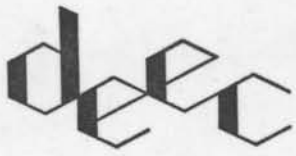


DEPARTAMENTO DE

ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

ESPECIFICAÇÃO E ESTRATÉGIA DE IMPLEMENTAÇÃO DE UM SISTEMA DE GERAÇÃO E DETECÇÃO DE TRÁFEGO

FACULDADE DE ENGENHARIA

UNIVERSIDADE DO PORTO

Rua dos Bragas, 4099 Porto Codex – PORTUGAL

Ref. 620

TESE DE MESTRADO

ESPECIFICAÇÃO E ESTRATÉGIA DE IMPLEMENTAÇÃO DE UM SISTEMA DE GERAÇÃO E DETECÇÃO DE TRÁFEGO

UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA
N.º 20130-Ex
CDU 681.3(043)
Data 22/11/1995
N.N. 34781

MANUEL ALBERTO PEREIRA RICARDO

1991

043 19
R376e
ex.2

Tese realizada sob supervisão do
Prof. Doutor Eurico Manuel Carrapatoso
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto

À Olga

Agradecimentos

Ao Prof. Doutor Eurico Manuel Carrapatoso pela disponibilidade com que sempre me atendeu, pelas revisões atentas desta tese e pela amizade com que me preveligiou.

Ao Prof. Doutor José Ruela Simões Fernandes por tudo o que me ensinou sobre redes.

Ao Prof. Doutor Artur Pimenta Alves pela confiança que sempre depositou em mim.

Ao meu amigo Eng. Clemente Aguiar, pelas longas horas de discussão que tanto influenciaram o Sistema na sua forma final.

A todos os colegas da equipa do SGDT pela prontidão com que sempre me auxiliaram.

À JCNIT pela bolsa de estudos concedida.

Ao INESC pelas óptimas condições de trabalho proporcionadas.

ÍNDICE

Índice i

Abreviaturas 1

1. Introdução 3

2. Enquadramento 7

3. Metodologia 13

3.1. Planeamento 13

3.2. Desenvolvimento 14

3.2.1. Projecto preliminar (FD1) 15

3.2.2. Projecto detalhado (FD2) 16

3.2.3. Execução e teste dos módulos (FD3) 16

3.2.4. Teste combinado dos módulos (FD4) 16

3.2.5. Integração (FD5) 17

4. Especificações do SGDT 19

4.1. Requisitos 19

4.2. Especificações 20

4.2.1. Descrição genérica 20

4.2.2. Conceitos básicos 23

4.2.2.1. Configuração 23

4.2.2.2. Canal de pacotes 24

4.2.2.3. Canal de sinalização 27

4.2.2.4. Sessão 28

4.2.2.5. Grupo 29

4.2.3. Modos de funcionamento 29

4.2.3.1. Avaliação de desempenho 29

4.2.3.2. Teste 30

4.2.4. Modelo hierárquico do software 31
4.2.5. Arquitectura do sistema 33
4.2.6. Arquitectura da unidade de processamento 33

5. Projecto de software **35**
5.1. Linguagem 36
5.2. Descrição do sistema 39
5.2.1. Estação de trabalho 42
5.2.2. Unidades de Processamento 46
5.2.3. Temporizações 53
5.2.4. Situações de erro 54

6. Requisitos de hardware **57**

7. Estratégia de implementação do software **63**
7.1. Assuntos não especificados em SDL 64
7.2. Máquina de estados 66
7.3. Estação de trabalho 68
7.3.1. Interfaces 69
7.3.2. Temporizador 71
7.3.3. Interface humana 71
7.3.4. Executor de sessões 72
7.4. Unidade de processamento 74
7.4.1. Interfaces 77
7.4.2. Temporizador 80
7.4.3. Supervisor 80
7.4.4. Placa C25 82
7.4.4.1. Interfaces 84
7.4.4.2. Temporizador 84
7.4.4.3. Gerador 85
7.4.4.4. Detector 86
7.4.4.5. Origem de chamadas 87

8. Conclusões **89**
8.1. Actividade 89

8.2. SGDT 90
8.3. Futuros Desenvolvimentos 92

9. Referências **95**

APÊNDICE A - Parâmetros de tráfego **A - 1**

APÊNDICE B - Artigos publicados **B - 1**

APÊNDICE C - Especificação formal **C - 1**

ABREVIATURAS

AFAP - "As Fast As Possible"
ATM - "Asynchronous Transfer Mode"
BCPN - "Business Costumer Premises Network"
BOD - "Bandwidth on Demand"
BSD - "Berkeley Software Distribution"
C25 - Processador de sinal digital TMS320C25, Texas Instruments
CCITT - "Comité Consultatif International pour le Télégraphe et le Téléphone"
CEE - Comunidade Económica Europeia
COFF - "Commom Object File Format"
D - Detector
DAMS - "Dynamically Adaptable Multiservice System"
DQAC - "Distributed Queue Access Control"
DSP - "Digital Signal Processor"
EPLD - "Erasable Programmable Logic Device"
ESPRIT - "European Strategic Program for Research in Information Technology"
FD i - Fase de Desenvolvimento i
FDDI - "Fiber Distributed Data Interface"
FIFO - "First In First Out"
FTP - "File Transfer Program"
G - Gerador
GR - Sintaxe gráfica do SDL
IEEE - "The Institute of Electrical and Electronics Engineers"
INESC - Instituto de Engenharia de Sistemas e Computadores
IO - "Input/Output"
IP - "Internet Protocol"
LAN - "Local Area Network"
MAC - "Medium Access Control"
MAN - "Metropolitan Area Network"
O - Origem de chamadas

PAL	- "Programmable Array Logic"
PPCA	- Posto Privado de Comutação Automática
PR	- Sintaxe textual do SDL
RACE	- "R & D in Advanced Communications Technologies for Europe"
RAM	- "Random Access Memory"
RDIS	- Rede Digital com Integração de Serviços
RISC	- "Reduced Instruction Set Computer"
ROM	- "Read Only Memory"
RPC	- "Remote Procedure Call"
SDL	- "Specification and Description Language"
SGDT	- Sistema de Geração e Detecção de Tráfego
STM	- "Synchronous Transfer Mode"
TCP	- "Transmission Control Protocol"
UCOL	- "Ultrawideband Coherent Optical LAN"
UDP	- "User Datagram Protocol"
UP	- Unidade de Processamento
VME	- "Versa Module Europe"
WS	- "Workstation"

1. INTRODUÇÃO

Esta tese situa-se na área das Redes de Telecomunicações de Banda Larga com Integração de Serviços e é o resultado do interesse do autor pelo conhecimento deste tipo de redes, particularmente nas suas componentes de sistemas e software de comunicações. Teve por base o trabalho de concepção de um sistema de geração e detecção de tráfego para teste e validação de redes de banda larga, que surgiu como consequência da não existência no mercado de ferramentas com estas características.

No âmbito de vários projectos europeus de investigação estão a ser desenvolvidas redes de tipo metropolitano, que se caracterizam pela oferta de novos serviços de telecomunicações, só possível pelo facto de usarem fibra óptica como meio de transmissão e de serem capazes de comutar grandes quantidades de tráfego.

O Sistema de Geração e Detecção de Tráfego (SGDT) é um dispositivo distribuído, fisicamente constituído por uma estação de trabalho, interligada através de uma rede local a 3 unidades que geram, detectam e analisam pacotes de dados e mensagens de sinalização destinados ao controlo de chamadas.

Os pacotes de dados, enviados para a rede com informação sem significado e em tempo real, são gerados de acordo com processos estatísticos que definem o intervalo entre geração de pacotes consecutivos e o comprimento dos pacotes. Os intervalos de tempo são controlados de modo a que fontes de tráfego reais, individuais ou multiplexadas, bem como funções de distribuição estatística usadas normalmente em estudos de tráfego possam ser simuladas, podendo cada unidade geradora atingir valores de pico de cerca de 60 Mbit/s. Pela detecção destes pacotes na mesma ou noutra unidade, depois de transportados pela rede, efectuam-se medidas que serão analisadas posteriormente, para avaliação do desempenho da rede. Medidas possíveis são o atraso dos pacotes, a variabilidade desse atraso, o número de pacotes perdidos e pacotes com erros.

As mensagens de sinalização são enviadas para pedidos à rede de estabelecimento e terminação de chamadas, com o objectivo de medir o tempo de resposta da rede e dos

dispositivos associados ao controlo das chamadas. Neste caso, a estratégia de teste consiste em definir, também por processos estatísticos, a taxa de chegada de novas chamadas bem como a sua duração. As medidas efectuadas sobre este tipo de tráfego são os atrasos de estabelecimento e terminação de chamadas bem como as taxas de insucesso dos pedidos.

Para que possa ser usado em duas situações distintas (caracterização estatística das redes e gerador de cargas controladas sem objectivos de medida), optou-se por dois modos de funcionamento para o sistema: avaliação de desempenho e teste. No primeiro modo as sessões de geração e detecção do sistema, normalmente de curta duração, são executadas como experiências não interrompíveis. No segundo, o sistema funciona como fonte de tráfego sobre o qual não são efectuadas medidas, com sessões de duração não determinada à partida, e podendo as cargas geradas ser alteradas em qualquer momento. Pelo facto de gerar e detectar em pontos físicos distintos, o sistema poderá ser classificado como distribuído. Pela necessidade de gerar, detectar e efectuar medidas dentro de intervalos de tempo reais, poderá também ser classificado como de execução em tempo real.

Para este trabalho de tese estabeleceram-se como objectivos a especificação e a definição da estratégia de implementação do SGDT. Nesse sentido, e numa fase inicial, foram feitos trabalhos de modelização de fontes de tráfego e de estudo e caracterização de parâmetros de qualidade de serviços. Foram, na fase seguinte, identificados os requisitos de um sistema com as características do SGDT. A especificação do sistema, com a identificação dos perfis de tráfego a gerar, medidas a efectuar, definição de estratégia conjunta de geração e detecção e definição da arquitectura básica do sistema, foi o passo seguinte. Já na fase de desenvolvimento, o sistema foi formalmente especificado usando uma linguagem do CCITT, o SDL ("Specification and Description Language"). O trabalho apresentado nesta tese foi dado como concluído com a definição da estratégia de implementação do software.

Relativamente à estrutura da tese começar-se-à, no capítulo 2, por enquadrar o Sistema no cenário actual das Telecomunicações, justificando-se a sua oportunidade. No capítulo 3 introduz-se a metodologia usada no seu desenvolvimento. No capítulo 4, são descritos os requisitos encontrados bem como as especificações do Sistema. No capítulo 5 é apresentado o projecto do Sistema usando uma linguagem formal e, no capítulo 6, os requisitos para o hardware. No capítulo 7 é definida a estratégia de implementação do software através da

descrição dos processos contituintes, funções e interfaces necessárias. Finalmente, no capítulo 8, são apresentadas as conclusões e perspectivados futuros desenvolvimentos.

São ainda incluídos como apêndices resultados de estudos efectuados sobre modelização de fontes de tráfego e parâmetros de qualidade de serviço, artigos publicados em conferências e a especificação formal em SDL.

Os trabalhos de implementação e teste do Sistema, que foram a continuação da actividade discutida nesta dissertação, encontram-se neste momento em fase de conclusão. Não são, no entanto, incluídos pelo facto de terem sido **executados por outros investigadores.**

2. ENQUADRAMENTO

Num cenário tradicional de Telecomunicações as redes podem ser classificadas, pelo tipo de proprietário, em públicas e privadas. As primeiras são normalmente propriedade de uma companhia, que terá ou não o monopólio da transmissão num dado espaço geográfico, e podem ser usadas por qualquer utilizador. As segundas pertencem normalmente a entidades privadas, que as usam apenas para o transporte do seu tráfego interno, podendo eventualmente estar ligadas à rede pública. Ambos os grupos podem ainda ser divididos, segundo o tipo de tráfego transportado, noutros dois: de voz e de dados.

Ao redes públicas de voz apresentam-se topologicamente como centrais de comutação fisicamente distribuídas, às quais se ligam terminais de voz (telefones), sendo a ligação entre centrais feita através de cabos ou fibras ópticas que, com a gradual digitalização destas redes, passam a suportar hierarquias de transmissão síncronas (STM) [1]. As redes privadas de voz são normalmente associadas a um comutador local (PPCA) ao qual se encontram ligados os telefones. Em ambos os casos a comutação é de circuitos, isto é, no estabelecimento de uma chamada são reservados os recursos necessários, que serão libertados apenas no fim da chamada. O tráfego na rede associado a uma chamada de voz é constante [2].

As redes públicas de dados tem normalmente topologias simples: alguns nós de comutação interligados entre si, aos quais se ligam os terminais de dados (computadores) através de acesso directo ou, aproveitando recursos já existentes, através de acesso comutado suportado pelas redes públicas de voz [3]. As redes privadas, normalmente designadas por redes locais, têm topologias diversas sendo as redes em forma de anel, barramento e estrela as mais usadas [4]. As suas capacidades de comutação vão normalmente até às dezenas de Mbits/s podendo no entanto, quando as necessidades de comutação do proprietário aumentam, ser usadas outras redes normalmente designadas de "backbones", às quais são ligadas as primeiras e computadores com maiores requisitos de comutação. Estes "backbones" são normalmente ópticos e comutam na ordem das dezenas-centenas de Mbit/s. A comutação nas redes de dados é feita à base do pacote, isto é, os recursos da rede

são reservados pacote a pacote pelo que, se o conceito de chamada existir, esta é normalmente virtual, feita pela associação lógica de um conjunto de pacotes com a mesma origem e o mesmo destino. O aparecimento do pacote como entidade comutada deve-se às características do tráfego de dados: irregular e com possibilidade de ser muito intenso em intervalos de tempo curto (alguns segundos) - grande "burstiness" [4].

Com a crescente digitalização nos últimos anos das redes públicas nos países industrializados, quer na comutação quer na transmissão entre centrais, e com o esforço desenvolvido na adaptação do lacete de assinante, começa neste momento a ser possível que cada assinante da rede pública de voz tenha uma interface digital. Com uma interface deste tipo passa a ser viável, através do mesmo suporte físico - os pares condutores de cobre que já hoje nos chegam a casa, transmitir tráfego de voz e de dados de uma forma integrada. A remodelação da rede existente dá, desta forma, origem à Rede Digital com Integração de Serviços - RDIS, que já se encontra parcialmente implementada em alguns países como o Japão, a França e a Alemanha). Uma interface RDIS normal, também chamada de básica, poderá suportar simultaneamente duas chamadas de voz, (cada uma com 64 kbit/s bidireccionais), a transmissão de pacotes até 16 kbit/s (unidireccionais) e a recepção de pacotes até 16kbit/s (unidireccionais), ou seja 144kbit/s bidireccionais. Pelo facto da rede suporte da RDIS ser a rede pública de voz, a comutação na primeira fase da RDIS é orientada apenas aos circuitos, isto é, a transmissão de dados tem que ser feita nos canais de voz (canais B, de 64 kbit/s), sendo o canal D utilizado apenas para a sinalização. Na segunda fase passa no entanto a ser possível a integração dos dois modos de comutação, ou seja, será possível a transmissão de pacotes de dados no canal D.

Da mesma forma que nas redes públicas, a integração começa também a surgir nas redes privadas. Centrais de comutação privadas RDIS são já um facto (assimilação dos dados pelas redes de voz). Evoluções das redes locais de dados no sentido de suportar voz são também uma realidade. O FDDI, uma rede em forma de anel que comuta pacotes, evoluiu no sentido da assimilação da voz dando origem ao FDDI II [5]. Este anel pode comutar voz sob a forma de circuitos e dados sob forma de pacotes.

Desenvolvimentos obtidos na última década na integração da eléctrónica, conjugados com a enorme largura de banda oferecida pela fibra óptica [6] e novos conhecimentos em sistemas distribuídos [7], levam a que sonhos antigos dos operadores das redes públicas

Desta forma, para que as estações possam comunicar num canal foi desenvolvido um protocolo próprio, o DQAC ("Distributed Queue Access Control"), que regula o acesso ao centro da estrela e permite a comutação de células ATM.

O DAMS [17] ("Dynamically Adaptable Multiservice System") tem como objectivo o desenvolvimento de um comutador integrado de voz e dados, capaz de fazer a transição para o ATM. Este comutador, quando ligado a um "backbone" FDDI II, pode transmitir dados para outros comutadores idênticos. Cada unidade pode comutar até 60 Mbit/s e oferece para o exterior interfaces de voz, RDIS, Ethernet e BOD ("Bandwidth on Demand" - IEEE 802.9). Uma das inovações do DAMS é a existência de um sistema operativo distribuído, que oferece primitivas de sinalização, gestão, transmissão de pacotes, além das facilidades normais num sistema de rede tais como "name server", "file server", etc.

Ambos os projectos sentiram a necessidade de um sistema que gerasse grandes quantidades de tráfego com características semelhantes às dos novos serviços e capaz de efectuar medidas de grandezas com significado para a validação e caracterização da qualidade dos serviços oferecidos pela rede. No UCOL, este sistema seria usado na validação do protocolo de acesso ao centro da estrela e para, na demonstração de rede, servir de sistema de geração e detecção de cargas controladas. O sistema deveria ser constituído por 3 unidades de geração/detecção capazes de gerar e detectar simultaneamente até 20 Mbit/s em células ATM.

No DAMS, seria necessário um sistema para a validação do comutador capaz de gerar também de cargas elevadas para o demonstrador. Como o DAMS é um sistema que comuta simultaneamente pacotes e circuitos, seria necessário que o sistema gerasse pacotes de dados com um comprimento máximo 2 Koctetos e mensagens de sinalização para estabelecer e terminar os circuitos. Seriam necessárias para o demonstrador 3 unidades geradoras/detectoras, cada uma com a capacidade de gerar e detectar até 60Mbit/s e tratar do estabelecimento e terminação de 500 chamadas simultâneamente.

Quando estes projectos se iniciaram, era conhecido um gerador/detector de tráfego [18] desenvolvido pela Universidade de Liége no âmbito do programa ESPRIT I, que injectava tráfego numa Ethernet a ritmos controlados. Existiam 5 processos distintos de geração e 1 de detecção que poderiam ser executados concorrentemente efectuando cada um deles algumas medidas. No entanto, e do ponto de vista dos projectos descritos, este sistema

apresentava limitações em termos de volume de tráfego gerado, interface fornecida e número de pontos de acesso à rede.

Foi assim decidido, em cada um dos projectos, realizar um sistema de geração e detecção de tráfego que, de acordo com a Comissão das Comunidades Europeias, seria comum, desenvolvido por uma instituição de investigação, o INESC, e que será detalhadamente apresentado nos capítulos que se seguem.

3. METODOLOGIA

Atendendo à incompleta definição do sistema a desenvolver, à sua previsivel complexidade, aos tempos de desenvolvimento e à importância para os projectos que o usam, sentiu-se a necessidade de introduzir um método para o seu desenvolvimento.

A introdução ou adopção de um método traduz-se na definição de um conjunto de passos que regulem todo o desenvolvimento do processo que se pretende sistematizar. A definição deste conjunto de passos ou regras passa por perceber o problema a resolver, encontrar situações onde se tenham verificado problemas idênticos, estudar as soluções encontradas e, se satisfatórias, adoptá-las com a consequente adaptação ao problema. A experiência colhida ao longo do processo de desenvolvimento será a realimentação desejada para a melhoria dessas regras.

A metodologia usada no SGDT [19] consistiu em dividir o seu processo evolutivo em duas fases: fase de planeamento e fase de desenvolvimento.

3.1. PLANEAMENTO

Na fase de planeamento, o sistema é identificado e definido de uma forma genérica. São assim definidos 2 passos nesta fase: identificação dos requisitos e elaboração das especificações.

A identificação dos requisitos consiste na formulação completa e clara do problema a resolver, ou seja na caracterização do sistema. A duração desta fase depende da forma como o cliente tiver elaborado os seus requisitos: quanto mais vagos forem, maior será o trabalho de formulação. É, no entanto, uma fase que tem sempre de existir, quanto mais não seja para que as necessidades do cliente sejam completamente entendidas e sem ambiguidades. No caso do SGDT, os requisitos eram muito vagos, pelo que existiu um grande trabalho na identificação do que deveria de facto fazer um sistema desta natureza.

A elaboração das especificações consiste na obtenção e descrição de uma solução para o problema. No caso do SGDT deverão ser descritos, por exemplo, quais as funções de tráfego a gerar, como serão caracterizadas, qual a precisão das medidas, que dados serão fornecidos ao utilizador, quando o serão, com que forma, etc. Devem ser também ser delineada, de uma forma simples, a divisão entre funções de hardware e software, bem como as escolhas de carácter genérico do sistema tais como hierarquia de computadores, uso de LANs, etc. As estimativas de custos e equipamentos a comprar devem também ser feitas.

A solução encontrada nesta fase será determinante no sucesso do sistema, uma vez que a qualidade deste será sempre o reflexo da especificação.

3.2. DESENVOLVIMENTO

Na fase de desenvolvimento (Fig. 3.1), e em função das especificações produzidas na fase de planeamento, o sistema começa a ganhar forma. Os projectos de hardware e software, que avançam em simultâneo com validação mútua, são divididos em 5 passos: projecto preliminar (FD1 - Fase de Desenvolvimento 1), projecto detalhado (FD2), execução e teste dos módulos (FD3), teste combinado dos módulos (FD4) e integração (FD5).

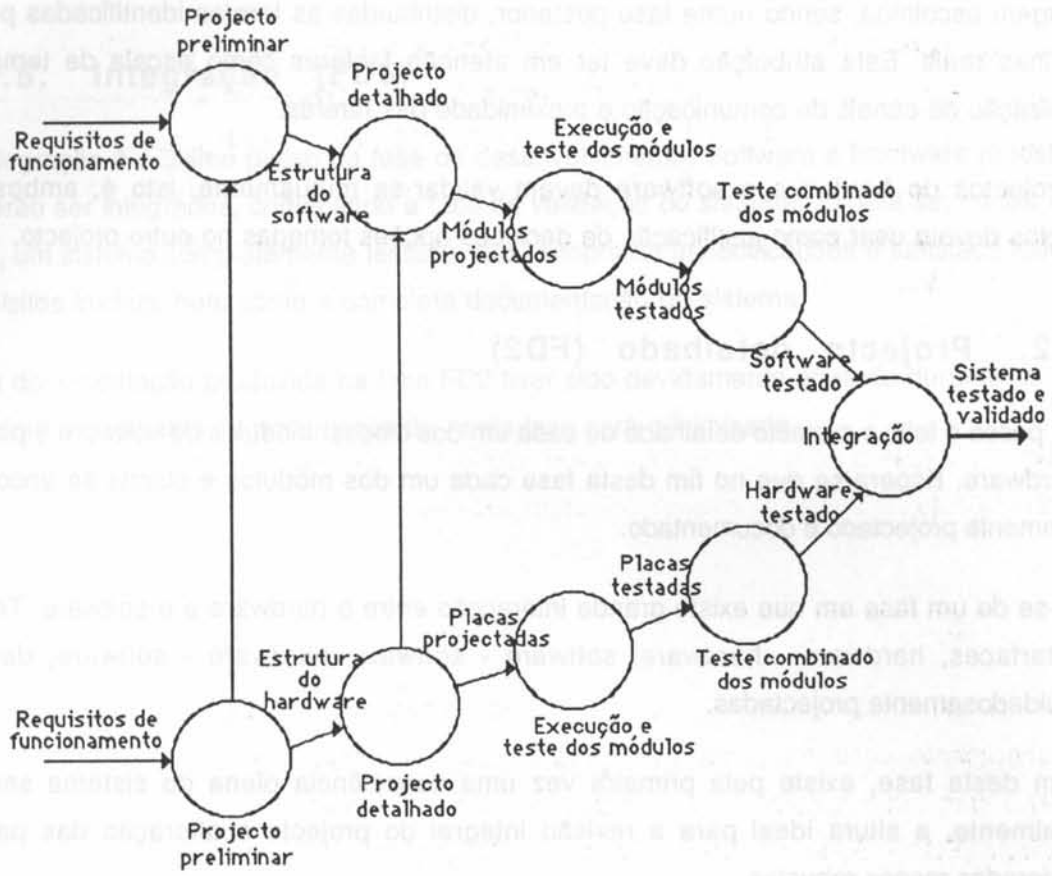


Figura 3.1 - Fase de desenvolvimento

3.2.1. Projecto preliminar (FD1)

Neste passo são executados os projectos preliminares do hardware e do software e são esperados, como resultados, as respectivas estruturas que deverão ser descritos em dois documentos de projecto preliminar.

Enquanto que o processo de estruturação do hardware é normalmente função da experiência do projectista, no software, para a execução do projecto preliminar, pode ser seguida uma estratégia mais bem definida. Esta estratégia passa pelo uso de uma linguagem que possua construções semânticas que permitam tratar os problemas de estruturação, paralelismo e comunicação necessários à modelização do sistema. Assumindo-se construído sobre uma única máquina - a máquina virtual, o sistema é modelizado na

linguagem escolhida, sendo numa fase posterior, distribuídas as tarefas identificadas pelas máquinas reais. Esta atribuição deve ter em atenção factores como escala de tempos, minimização de canais de comunicação e proximidade das tarefas.

Os projectos de hardware e software devem validar-se mutuamente, isto é, ambos os projectos devem usar como justificação de decisões opções tomadas no outro projecto.

3.2.2. Projecto detalhado (FD2)

Neste passo é feito o projecto detalhado de cada um dos blocos: módulos de software e placas de hardware. Espera-se que no fim desta fase cada um dos módulos e placas se encontre devidamente projectado e documentado.

Trata-se de um fase em que existe grande interacção entre o hardware e o software. Todas as interfaces, hardware - hardware, software - software, hardware - software, devem ser cuidadosamente projectadas.

No fim desta fase, existe pela primeira vez uma consciência plena do sistema sendo, normalmente, a altura ideal para a revisão integral do projecto e alteração das partes consideradas menos robustas.

3.2.3. Execução e teste dos módulos (FD3)

Cada um dos elementos a executar (módulos de software e placas de hardware) deve ser construído e testado nesta fase. No fim espera-se que todos os módulos se encontrem em boas condições de funcionamento.

3.2.4. Teste combinado dos módulos (FD4)

É feita a integração do software e hardware separadamente. Espera-se que no fim, tanto o software como o hardware esteja em boas condições de operacionalidade.

3.2.5. Integração (FD5)

A integração é o último passo da fase de desenvolvimento. Software e hardware já testados deverão ser integrados, começando a fase de validação do sistema. Espera-se, no fim desta fase, um sistema completamente testado, que cumpra as especificações e satisfaça todos os requisitos iniciais, bem como a completa documentação do sistema.

Se a documentacão produzida na fase FD2 tiver sido devidamente anotada durante as fases seguintes, o trabalho de documentação nesta fase será minimizado.

4. ESPECIFICAÇÕES DO SGDT

Nesta capítulo é descrita a fase de planeamento do SGDT, sendo identificados e uniformizados os requisitos dos dois projectos e apresentadas as especificações.

4.1. REQUISITOS

Ambos os projectos esperavam que o SGDT fosse um dispositivo que pudesse ser usado na validação de partes específicas das redes e que gerasse, para os respectivos demonstradores, quantidades elevadas de tráfego com características da carga real.

No UCOL, este sistema deveria efectuar medidas que permitissem caracterizar o protocolo usado em cada canal para controlar o acesso ao centro da estrela. Para o demonstrador seria necessário que cada uma das 3 unidades fosse capaz de gerar 20Mbit/s em células ATM, ou seja cerca de 50Kcélulas/s, com características de carga real.

No DAMS, o sistema deveria efectuar medidas para caracterizar o comutador. Como o DAMS é um sistema que comuta simultaneamente pacotes e circuitos, seria necessário que o sistema gerasse pacotes de dados com um comprimento máximo de 2 Koctetos e mensagens de sinalização para o estabelecimento e terminação de circuitos. Para o demonstrador seriam necessárias 3 unidades geradoras/detectoras, cada uma com a capacidade de gerar e detectar até 60Mbit/s em pacotes de dados e tratar do estabelecimento e terminação de 500 chamadas.

Atendendo às condições impostas pelas duas redes, os requisitos do SGDT poderão ser resumidos da seguinte forma:

- Topológicamente, o sistema deve ser constituído por 3 unidades geradoras e detectoras de tráfego. Cada unidade deve ser capaz de suportar duas interfaces físicas, para acesso a cada uma das redes (UCOL e DAMS). No sistema deverá existir apenas uma interface com o utilizador, de preferência gráfica e a cores, que permita a configuração do sistema, a

definição do tráfego a gerar e medidas a efectuar, e a visualisação fácil dos resultados obtidos.

- A geração deve poder ser feita em cada uma das três unidades. O tráfego deve ser de dois tipos: pacotes de informação ou mensagens de sinalização. Os pacotes gerados, que contêm informação sem significado para a rede, devem poder ter comprimento variável entre 48+5 e 2048 octetos. As variáveis de controlo deste tipo de tráfego devem ser a taxa de geração de pacotes e o comprimento dos mesmos, devendo cada unidade ser capaz de gerar até 50 000 pacotes por segundo. As mensagens, bem como os procedimentos de sinalização, deverão ser as normalmente usadas no DAMS para o estabelecimento e terminação de chamadas [20], ao nível 3 do modelo de referência OSI. As variáveis de controlo do tráfego sinalização deverão ser a taxa de estabelecimento e a duração das chamadas, devendo cada unidade ter a capacidade de tratar até 500 chamadas simultâneas. Devem ser feitas as medidas possíveis relativamente aos processos de geração dos dois tipos de tráfego.

- A detecção deve também ser feita em cada uma das três unidades, podendo o tráfego recebido ser de dois tipos: os pacotes de informação gerados pelo SGDT, depois de transportados pela rede, e as mensagens de sinalização enviadas pela entidade de controlo de chamadas da rede, em resposta aos pedidos de estabelecimento e terminação de chamadas. As medidas a efectuar deverão ser as possíveis e oferecer informação útil para o teste de validação das redes.

4.2. ESPECIFICAÇÕES

A descrição genérica do SGDT e sua topologia é apresentada nesta secção, seguida da descrição dos conceitos básicos introduzidos e dos modos de funcionamento. É ainda apresentada a estrutura hierárquica do software bem como a arquitectura básica do hardware.

4.2.1. Descrição genérica

O Sistema de Geração e Detecção de Tráfego é topologicamente constituido por três Unidades de Processamento (UPs) e uma "Workstation" (WS) interligadas por uma rede de dados (Fig. 4.1).

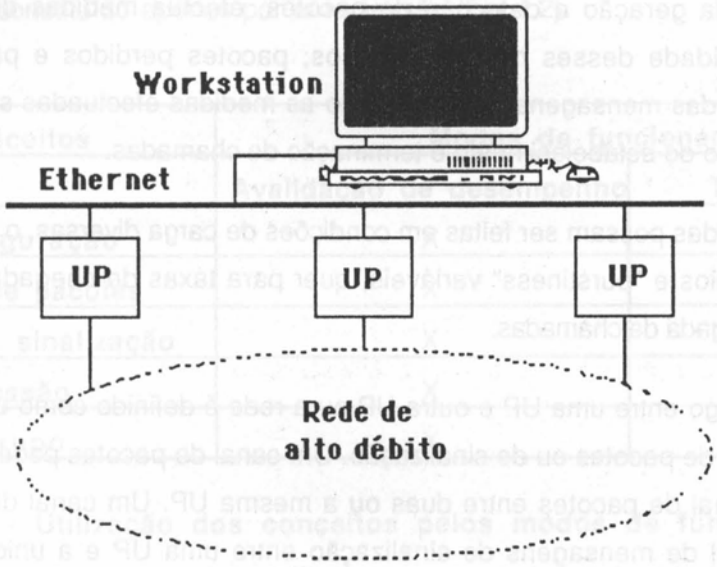


Figura 4.1 - Sistema de Geração e Detecção de Tráfego

As UPs, que se encontram também fisicamente ligadas à rede a testar, são as entidades físicas responsáveis pela geração e detecção dos pacotes de dados e das mensagens de sinalização. A WS é o dispositivo usado pelo utilizador para configurar o sistema, definir as experiências e visualizar os resultados.

De acordo com a configuração das tarefas de cada UP, a WS descarrega para estas o software necessário à sua execução. Numa fase seguinte, depois de passadas as características das experiências a executar, as UPs por ordem da WS começarão a gerar e a detectar pacotes de dados e/ou mensagens de sinalização. A visualização dos resultados será feita na WS, também por ordem explícita desta. Como modelo de referência de funcionamento para o SGDT foi usado o modelo cliente - servidor (WS - PUs): o primeiro dá ordens que serão executadas remotamente nos últimos.

Para o SGDT foram escolhidos dois modos de funcionamento: avaliação de desempenho e teste.

Avaliação de desempenho é o modo de funcionamento básico. Neste modo, o sistema carateriza a influência da rede no tráfego gerado. Sendo a rede ideal uma cujo tráfego no destino tenha as mesmas características que na origem e com um atraso nulo, o SGDT fará

medidas no sentido de avaliar o desvio da rede sob teste em relação à rede ideal. Para tal o SGDT, através da geração e detecção de pacotes, efectua medidas de atrasos extremo extremo, variabilidade desses mesmos atrasos, pacotes perdidos e pacotes com erros. Quando são geradas mensagens de sinalização as medidas efectuadas são os atrasos e as taxas de insucesso do estabelecimento e terminação de chamadas.

Para que as medidas possam ser feitas em condições de carga diversas, o SGDT gera tráfego com valores médios e "burstiness" variáveis, quer para taxas de chegada de pacotes, quer para taxas de chegada de chamadas.

Um fluxo de tráfego entre uma UP e outra UP ou a rede é definido como um canal, que pode ser de dois tipos: de pacotes ou de sinalização. Um canal de pacotes pode ser visto como um fluxo unidireccional de pacotes entre duas ou a mesma UP. Um canal de sinalização é um fluxo bidirectional de mensagens de sinalização entre uma UP e a unidade de gestão de chamadas na rede. Canais dos dois tipos podem coexistir.

Uma experiência durante a qual é gerado e detectado tráfego nos canais definidos é chamada de "sessão". Durante uma sessão as características dos canais não podem ser alteradas para que não se modifiquem as condições de geração, porque só desta forma faz sentido que as medidas sejam associadas a processos bem caracterizados estatisticamente. Assim as sessões são definidas, executadas e só no fim os resultados serão disponibilizados. A duração típica de uma sessão neste modo é de poucos segundos e função das taxas de geração. Se cada pacote for considerado uma amostra do estado da rede, se for considerado que o processo de medida fica bem caracterizado com, por exemplo, 100.000 amostras e se a taxa de geração for de 20 Kpacote/s, então a duração da sessão deverá ser de 5 segundos.

Um conjunto de sessões que usa a mesma configuração do sistema podem ser associadas num "grupo". As sessões que constituem um grupo podem ser executadas sequencialmente sem o envolvimento do utilizador.

No seu segundo modo - teste, o SGDT funciona como um um sistema versátil de geração de tráfego, podendo as características de geração ser alteradas durante a execução de uma sessão, não sendo, portanto, válidas as medidas de carácter estatístico. Os canais continuam a ser definidos da mesma forma que na avaliação de desempenho mas não tendo a

sessão uma duração prédefinida. O conceito de grupo também não é aplicável - de facto cada grupo será constituído apenas por uma sessão (Fig 4.2).

Conceitos	Modos de funcionamento	
	Avalidação de desempenho	Teste
Configuração	X	X
Canal de pacotes	X	X
Canal de sinalização	X	X
Sessão	X	X
Grupo	X	- -

Figura 4.2 - Utilização dos conceitos pelos modos de funcionamento

4.2.2. Conceitos básicos

Os conceitos básicos apresentados em 4.2.1 foram introduzidos com o objectivo de facilitar ao utilizador a percepção e modelização do sistema, pela criação de objectos virtuais facilmente identificáveis e manipuláveis. Estes conceitos são apresentados nos pontos seguintes, pela descrição dos seus tipos.

4.2.2.1. Configuração

A configuração do sistema consiste na atribuição a cada UP das tarefas a desempenhar. Cada UP poderá desempenhar simultaneamente até duas das seguintes tarefas: geração de pacotes (G), detecção de pacotes (D) ou origem de chamadas (O).

Assumindo que o estado de não funcionamento é representado por mais uma tarefa (N), cada UP poderá suportar até $C_s^t + [t! - P_s^t]$ configurações diferentes, sendo C o número de combinações, P o número de permutações, t o número de tarefas e s o número de tarefas simutâneas. Com t=4 e s=2 serão possíveis 10 configurações em cada UP (GG, GD, GO, GN, DD,DO, DN, OO, ON, NN). Assumindo ainda o príncipio da indepêndencia das UPs, serão possíveis para o SGDT 10^3 configurações diferentes.

4.2.2.2. Canal de pacotes

Um canal de pacotes pode ser visto como um fluxo unidireccional de pacotes de uma UP para a mesma UP, outra UP ou nenhuma UP. Neste último caso, o tráfego gerado não estará a ser detectado, servindo apenas para carregar a rede. Um canal deste tipo, cujos pacotes são de informação sem significado (Fig. 4.3), é caracterizado pelos seguintes elementos: origem, destino, perfil de tráfego e resultados.

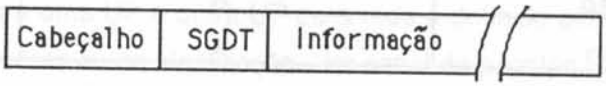


Figura 4.3 - Pacote típico transportado num canal

A origem, que é a identificação da UP que gera o tráfego, poderá ser qualquer UP. Quanto ao destino, trata-se da identificação da UP que detecta o tráfego do canal e que poderá ser nenhuma ou uma UP qualquer.

Perfil de tráfego é a informação que descreve as características dos ritmos de tráfego a gerar no canal. Estes ritmos podem ser de dois tipos: um estado ou dois estados.

O tipo de perfil com um estado é caracterizado pelo comprimento de pacote e intervalo entre pacotes consecutivos. Se também for definido o tempo de permanência do gerador nesse estado antes de comutar para outro, e este segundo for também definido pelos mesmos parâmetros, obtemos o tipo de geração com dois estados. Assim, no modelo de geração de dois estados, os parâmetros usados para a caracterização de cada estado são: intervalo entre pacotes, comprimento de pacote e número de pacotes gerado nesse estado.

Se não existir geração num dos estados, obtem-se um perfil geração-pausa, sendo a permanência no estado de pausa definida em unidades de tempo.

Cada um destes 3 ou 6 parâmetros, conforme o tipo de tráfego gerado, é descrito de uma forma estatística, por uma função distribuição de probabilidade, com um dado valor médio e uma variância, se necessário. As funcões distribuição de probabilidade usadas são

Poisson, Gaussiana e Uniforme. Valores contantes para os parâmetros descritos são também permitidos.

Apesar da definição do perfil de tráfego poder ser completamente definida pelo utilizador, são pré-definidas 6 funções de geração. Deterministica, Markoviana e Gaussiana para o tipo um estado, "Burst determinístico" e "Burst de Poisson" para o tipo de 2 estados e "AFAP" ("As Fast As Possible").

As funções Determinística, Markoviana e Gaussiana são caracterizadas por comprimento de pacote constante e intervalo entre pacotes, respectivamente, constante e com distribuição exponencial e gaussiana.

As funções de "burst" são, para cada um dos dois estados, caracterizadas por um comprimento de pacote constante e intervalo entre pacotes e tempo de permanência em cada estado constantes para o determinístico e exponencialmente distríbuidos para o de Poisson. Nestas funções, como em todas as funções do tipo 2 estados, o gerador alterna entre a geração no estado 1 e a geração no estado 2: o gerador começa, por exemplo, no estado 1 e, após a geração de N_1 pacotes, passa para o estado 2, de onde só sairá outra vez para o estado 1 após ter gerado N_2 pacotes. O ritmo em cada estado processa-se de acordo com as distribuições e valores médios respectivos. O processo de geração repete-se até que a sessão termine.

A função "AFAP", que não é de um nem de dois estados, caracteriza-se pela geração pela UP do maior caudal de tráfego possível.

O último elemento caracterizador de um canal de pacotes são os resultados, que são divididos em medidas na geração ou origem e medidas na detecção ou destino. As medidas a efectuar em cada um dos pontos físicos podem ser de dois tipos: histogramas e contagens. Os histogramas são caracterizados pelo número de intervalos, largura dos intervalos e valor mínimo.

Uma vez obtidos os resultados, os valores deverão ser associados ao canal. O que se espera relativamente aos histogramas é um vector de ocorrências com o número de acontecimentos para cada intervalo e um valor máximo. Com base na informação do

histograma são obtidos o valor médio e variância. Correlações com curvas de distribuição de probabilidade conhecidas podem também ser obtidas (Fig. 4.4).

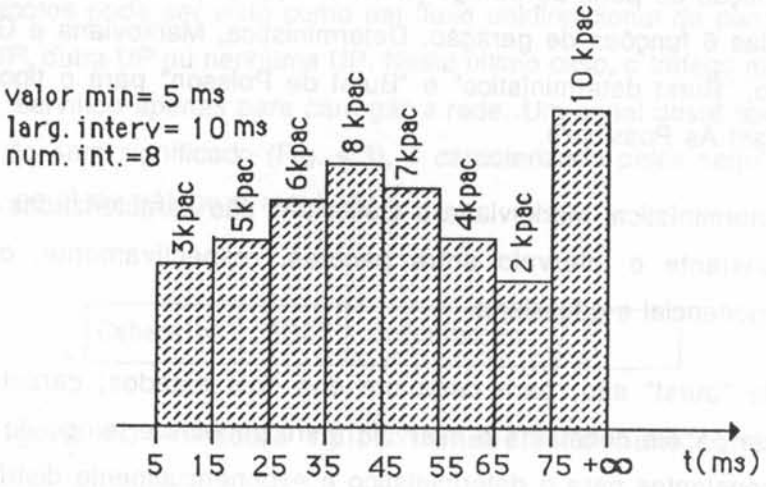


Figura 4.4 - Informação obtida através de um histograma associado ao atraso extremo a extremo dos pacotes gerados num canal.

As contagens associadas ao canal de pacotes efectuadas na origem são o número de pacotes gerados. Os histogramas são: intervalos entre pacotes consecutivos e comprimentos de pacotes.

As contagens associadas à detecção são: número de pacotes detectados, número de pacotes com erros, número de pacotes perdidos e número de pacotes incompletos. Os histogramas são: atrasos extremo a extremo (da origem ao destino) e intervalos entre pacotes.

A definição da origem e destino de um canal de pacotes deve ter em consideração a configuração escolhida para o sistema: as UPs não podem executar tarefas para as quais não estejam configuradas nem ser ocupadas por mais do que um canal.

Na atribuição de um canal de pacotes acrescente-se ainda que a definição da sua origem e destino deve ter em consideração a configuração escolhida para o sistema: as UPs devem estar configuradas com as tarefas necessárias e não podem estar ocupadas por outros canais.

4.2.2.3. Canal de sinalização

Um canal de sinalização é definido como um fluxo bidireccional de mensagens de sinalização entre uma UP e a entidade da rede responsável pela de gestão de chamadas. As mensagens transportadas são as necessárias ao estabelecimento e terminação de uma chamada (Fig. 4.5 e Fig. 4.6). Todas as chamadas estabelecidas correspondem a circuitos bidireccionais de 64 kbit/s.

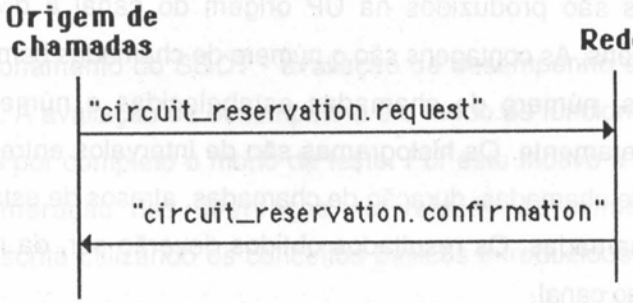


Figura 4.5 - Mensagens para o estabelecimento de uma chamada

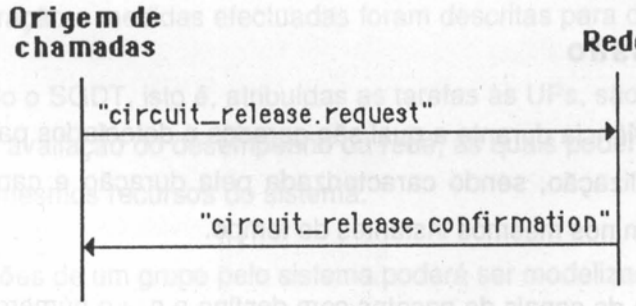


Figura 4.6 - Mensagens para a terminação de uma chamada

Um canal de sinalização é caracterizado pelos seguintes elementos: origem, perfil de tráfego e resultados.

Origem é a identificação da UP que gera as mensagens de sinalização enviadas para a rede e detecta as mensagens enviadas pela rede. Poderá ser qualquer UP.

Perfil de tráfego descreve as taxas de estabelecimento e terminação de chamadas, que são caracterizadas por intervalo entre chamadas consecutivas e duração das chamadas. Da mesma forma que para os canais de pacotes, cada um destes elementos é descrito estatisticamente usando as mesmas funções de distribuição, sendo, no entanto, usado apenas o modelo de 1 estado.

Existe uma função pré-definida para a geração de chamadas Markoviana: intervalos entre início de chamadas consecutivas e duração de chamadas são exponencialmente distribuídos.

Todos os resultados são produzidos na UP origem do canal e dividem-se também em histogramas e contagens. As contagens são o número de chamadas bem sucedidas, número de chamadas rejeitadas, número de chamadas estabelecidas e número de chamadas não estabelecidas completamente. Os histogramas são de intervalos entre pedidos consecutivos de estabelecimento de chamadas, duração de chamadas, atrasos de estabelecimento e atrasos de terminação de chamadas. Os resultados obtidos deverão ser, da mesma forma que nos pacotes, associados ao canal.

A definição da origem de um canal de sinalização deve ter em consideração as tarefas com que a UP de origem está configurada e a sua eventual ocupação por outros canais.

4.2.2.4. Sessão

Sessão é uma experiência durante a qual são gerados e detectados pacotes de informação e mensagens de sinalização, sendo caracterizada pela duração e canais constituintes, que começam e terminam nos mesmos instantes de tempo.

Sendo n_{cd} o número de canais de pacotes com destino e n_{sd} o número de canais de pacotes sem destino atribuído ou de sinalização, o número de canais em cada sessão é $n_{cd} + n_{sd}$ com $n_{cd} = 0, 1, 2, 3$ e $n_{sd} = 0, 1, ..., 6-2*n_{cd}$. O limite máximo será de 6 canais na situação em que $n_{cd} = 0$ e $n_{sd} = 6$. É a situação em que o SGDT está configurado só com tarefas geradoras e originadoras de chamadas e cada tarefa é origem de um canal.

4.2.2.5. Grupo

Grupo é definido como um conjunto de sessões que requerem os mesmos recursos do SGDT, e normalmente com objectivos de teste idênticos. O grupo fica caracterizado pelas sessões constituintes e configuração do SGDT usada a qual deve permanecer fixa para todas as sessões do grupo

4.2.3. Modos de funcionamento

Os 2 modos de funcionamento do SGDT - avaliação de desempenho e teste - são descritos nos pontos seguintes. A avaliação de desempenho é o modo de funcionamento principal, pelo facto de conter quase por completo o modo de teste. Por este motivo a descrição do 2º modo será feita pela enumeração das diferenças em relação ao primeiro. A avaliação de desempenho será descrita utilizando os conceitos básicos introduzidos.

4.2.3.1. Avaliação de desempenho

Neste modo o SGDT avalia o desempenho da rede através da execução de sessões cujas características de geração e medidas efectuadas foram descritas para os canais.

Depois de configurado o SGDT, isto é, atribuídas as tarefas às UPs, são definidas as sessões que serão usadas na avaliação do desempenho da rede, as quais podem ser associadas num grupo se usarem os mesmos recursos do sistema.

A execução das sessões de um grupo pelo sistema poderá ser modelizada através de uma fila de espera FIFO, na qual são colocadas as sessões do grupo (de uma só vez ou não) por ordem explícita do utilizador. Estas sessões irão sendo executadas ou consumidas sequencialmente pelas UPs, pela ordem de chegada ou de colocação na fila, sendo a ordem de fecho da fila dada também pelo utilizador.

Assim, neste modo de funcionamento, uma sessão passa por 4 estados: definição, espera, execução e completa (Fig. 4.7).

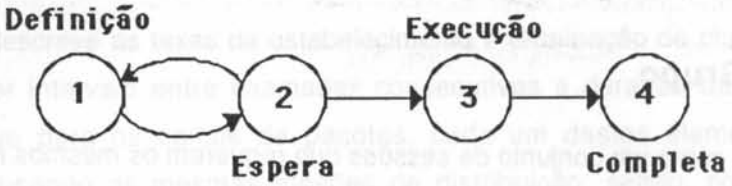


Figura 4.7 - Diagrama de estados de uma sessão no modo de avaliação de desempenho

Quando uma sessão está em definição são escolhidos o número de canais, características de cada canal e a duração. A transição para o estado de espera dá-se por ordem explícita do utilizador.

No estado de espera a sessão encontra-se numa fila, esperando pela sua vez para ser executada. A passagem ao estado de execução dá-se quando a sessão anterior tiver sido executada, ficando o sistema apto para nova execução. Por ordem explícita do utilizador, a sessão em estado de espera pode voltar ainda ao estado de definição.

No estado de execução, a sessão é retirada da fila, interpretada e executada pelas UPs que começarão a gerar e a detectar tráfego. Findo o tempo de execução da sessão os resultados obtidos são adicionados às estruturas de informação da sessão dando-se a sua pasagem ao estado de completa. Uma vez completa, os resultados da sessão podem ser analisados através de ferramentas gráficas especialmente concebidas para o efeito.

Neste modo de funcionamento, os parâmetros de geração se uma sessão não podem ser alterados quando esta se encontra em execução. Seria errado associar os resultados dos histogramas aos dois processos de geração (antes e depois da alteração). A alteração dos ritmos faz-se através da escolha adequada das funções de geração e suas distribuições ou pela definição de mais do que uma sessão.

4.2.3.2. Teste

No seu segundo modo - teste, o SGDT funciona como um um sistema versátil de geração de tráfego. Depois de configurado, a sessão é definida pela caracterização dos seus canais. A diferença em relação ao modo de funcionamento básico é que não podem ser obtidos histogramas, não sendo também pré-definida a duração de uma sessão. O conceito de grupo

e fila de espera associada também não são aplicáveis. Por esse facto, os estados de uma sessão resumem-se a definição, execução e completa (Fig. 4.8).

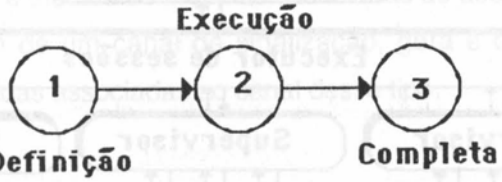


Figura 4.8 - Diagrama de estados de uma sessão no modo de teste

Na definição a sessão é definida pelo número de canais e características de cada canal. A transição para o estado de execução faz-se por ordem explícita do utilizador.

No estado de execução, a sessão é interpretada e executada pelas UPs, que começarão a gerar e a detectar tráfego. A execução da sessão pode ser interrompida para alteração de parâmetros de geração, ou para pedido de envio de mensagens específicas de teste para a rede. A informação relativamente à geração e detecção deve poder ser enviada em tempo real para a interface com o utilizador. A sessão passa ao estado de completa também por ordem do utilizador. Neste estado a sessão termina sem que seja gravada outra informação além da definição da sessão.

Por uma questão de calendarização do SGDT em relação aos projectos, indefinição das mensagens de teste a enviar para a rede e procedimentos relacionados, e por razões de planeamento do trabalho foi decidido implementar numa primeira fase só o modo de funcionamento básico fazendo-se, numa segunda fase, a extensão do sistema ao teste. Assim o trabalho de projecto preliminar e detalhado apresentados nos pontos seguintes desta tese dirão apenas respeito ao modo de funcionamento básico ou de avaliação de desempenho.

4.2.4. Modelo hierárquico do software

As tarefas principais de software são: Interface humana, Gestor de sessões, Executor de sessões, Supervisor, Gerador, Detector e Origem.

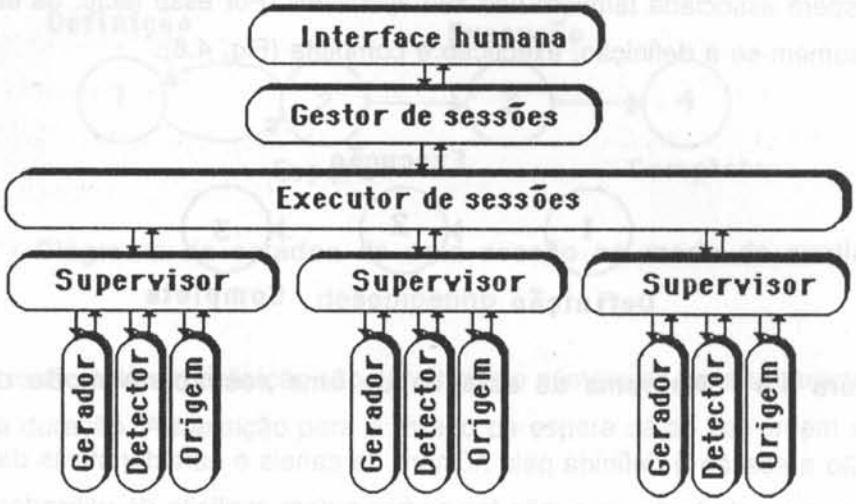


Figura 4.9 - Modelo hiérarquico do software do SGDT

O software apresenta dois tipos de estrutura: vertical e horizontal. A estrutura vertical é baseada em níveis que oferecem serviços aos níveis superiores, sendo a horizontal usada para, num dado nivel, dividir o software em blocos mais pequenos. Estes blocos são independentes e estão agrupados pela proximidade de funções que oferecem ou por proximidade física. A estrutura do software do SGDT, com 3 UPs, pode ser observada na Fig. 4.9.

A Interface humana permite a definição e análise gráfica das sessões, a visualização e a manipulação da fila de espera do grupo, a configuração do sistema e o armazenamento de dados e resultados de sessões em directórios próprios. Transforma ainda a informação característica dos conceitos virtuais, orientados ao homem, em informação orientada à máquina.

O Gestor de sessões é a entidade que gere a fila de espera de sessões, a qual deverá ser implementada fisicamente. Com a introdução de uma fila de facto poder-se-á isolar a Interface humana do resto do sistema, permitindo-lhes (interface e resto do sistema) que funcionem de forma independente e em paralelo.

O Executor de sessões é a entidade que executa as sessões, retirando uma de cada vez da fila. O Supervisor, que corre em cada UP, ordena a execução local das ordens recebidas do Executor de sessões.

O Gerador, que é a origem de um canal de pacotes, gera e efectua as medidas na origem de acordo com as características da sessão em execução. O Detector, destino de um canal de pacotes, detecta os pacotes e efectua as medidas associadas ao destino do canal. A Origem de chamadas, que é a origem de um canal de sinalização, gera e detecta as mensagens de sialização e efectua as medidas associadas ao canal desse tipo.

4.2.5. Arquitectura do sistema

O SGDT consiste numa "Workstation" e 3 Unidades de Processamento interligados entre si por uma rede local de dados (Fig. 4.1).

A WS, cuja função básica é permitir que o acesso ao sistema seja centralizado, suportará as tarefas de Interface humana, Gestor de sessões e Executor de Sessões. Para tal é necessário um sistema operativo multitarefa com ambiente de janelas que permita a construção da interface gráfica. Mecanismos de comunicação entre processos são também desejáveis. Interfaces de hardware à rede e software para desenvolvimento de aplicações de rede são também necessários.

A rede de dados será o meio usado para a transferência de dados entre a WS e as UPs. Para que o preço das placas de interface e pacotes de software de rede fosse baixo e o software de rede a desenvolver não oferecesse dificuldades de maior foi escolhida uma Ethernet.

As UPs serão as entidades que geram e detectam o tráfego real. As tarefas suportadas nesta unidades, que serão todas iguais, são o Supervisor, o Gerador, o Detector e a Origem.

4.2.6. Arquitectura da unidade de processamento

A UP é constituída por uma placa genérica baseada num CPU 68030 da Motorola e duas placas usadas para a geração e detecção de tráfego baseadas num processador de sinal digital, o TMS320C25 (C25) da Texas Instruments, interligados por um barramento standard VME (Fig. 4.10).

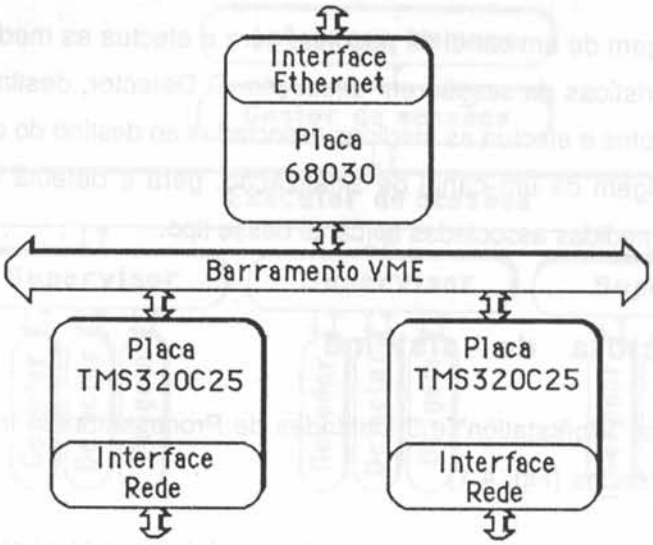


Figura 4.10 - Arquitectura da unidade de processamento

As placas baseadas no C25 são placas dedicadas às tarefas de geração, detecção e origem de chamadas. Estas placas, que serão desenvolvidas, devem suportar as interfaces físicas de acesso às duas redes (DAMS e UCOL).

Na placa 68030 é usado o OS9, um sistema operativo multitarefa com características de tempo real.

5. PROJECTO DE SOFTWARE

Neste capítulo é apresentado o projecto preliminar do software o qual corresponde à fase de desenvolvimento 1 da metodologia adoptada.

O projecto preliminar, que tem como objectivo a definição do sistema, é feito pela identificação quer das tarefas constituintes e respectivas interfaces, quer dos requisitos de cada tarefa.

Por identificação de interfaces entende-se a definição dos modos de comunicação entre as tarefas: protocolos e mensagens, funções com valores de retorno ou informação partilhada. Por identificação dos requistos entende-se a identificação das necessidades de tempo, de espaço de armazenamento e, consequentemente, de recursos necessários: requisitos especiais para as placas de hardware ("timers", FIFOs), quantidades de memória, implementação directa de tarefas em hardware ("PALs, EPLDs"), criação de novos "device drivers" ou outras infraestruturas de comunicação, etc.

A dificuldade sentida na abordagem destes problemas, isto é, em expressar um sistema da natureza do SGDT em linguagem corrente ou natural, fez sentir a necessidade de uma linguagem com uma semântica própria para este tipo de problemas. Aspectos de estruturação, paralelismo, comunicação e controlo, deveriam poder ser descritos de forma não ambígua. A descrição do funcionamento básico de cada uma das tarefas, tomando em consideração os objectivos a atingir e a arquitectura básica delineada para o sistema, deveria também ser possível. Desta forma, além da definição das interfaces e da identificação dos requisitos, poder-se-ia também obter uma referência útil para a fase de implementação.

A linguagem escolhida para se expressar formalmente o SGDT foi o SDL - "Specification and Description Language", do CCITT [21]. Esta linguagem tem sido tradicionalmente usada para modelização de conceitos e especificação e descrição de protocolos por organismos de normalização do CCITT, e para especificação e projecto detalhado de sistemas de comutação na indústria das telecomunicações. O seu uso nos últimos tempos tem vindo a alargar-se

[22]. Esta linguagem, que globalmente satisfaz os requisitos identificados, foi definitivamente escolhida pela existência no INESC de uma ferramenta que permite a edição e localização gráfica de erros sintácticos e semânticos, bem como a simulação dos sistemas descritos em SDL [23].

No ponto seguinte será feita uma introdução à linguagem pela apresentação dos seus 4 aspectos principais - funcionamento, comunicação, estruturação e representação de dados - sendo para cada um deles descrito o seu modelo e as construcções básicas. A descrição formal do SGDT será feita em 5.2.

5.1. LINGUAGEM

O SDL, que sintacticamente se pode apresentar numa forma gráfica (GR) ou textual (PR), pode ser apresentado por 4 aspectos principais [24]: funcionamento, comunicação, estruturação e representação de dados.

O modelo assumido para o funcionamento é o de uma máquina de estados finitos alargada [25]. Esta máquina é uma entidade que processa dados na transição entre estados e fica adormecida enquanto permaneça num estado. O estado atingido após uma transição depende do estado inicial, dos dados chegados e das condições locais.

Esta máquina é descrita em SDL por um "PROCESS". Um "PROCESS" é, de uma forma simplista, definido pelo número de instâncias ou máquinas de facto permitidas e pelo seu funcionamento. As contruções básicas usadas na descrição do funcionamento são: "START", "STATE", "DECISION", "TASK" e "STOP". "START", "STOP" e "STATE" são usadas para descrição de estados, sendo "START" usada para o primeiro estado da máquina e "STOP" para descrever o último que, uma vez atingido, fará com que a máquina deixe de existir. "DECISION" é a construção que permite a tomada de decisões na máquina com base em informação contida em variáveis. "TASK" é usada para a execução de operações sobre variáveis.

Uma construção que pode ser usada pelo "PROCESS", o "TIMER", permite modelizar um "relógio despertador" no qual podem ser colocados tempos que, quando atingidos, dão origem a que "TIMER" envie um sinal ao "PROCESS". Estes tempos são colocado através da

construção "SET", que também inicia a actividade do "TIMER", podendo a sua desactivação ser feita por um "RESET".

A comunicação entre 2 máquinas de estados assíncronas (instâncias de processo) pode ser feita por 3 formas: envio explícito de um sinal, partilha de informação ou no acto de criação da segunda máquina pela primeira.

Cada instância tem uma fila de espera associada, na qual são depositados pela ordem de chegada todos os sinais provenientes de outras máquinas, independentemente do canal de transmissão usado. A recepção ou consumo dos sinais é feito por ordem explícita da máquina. Quando esperado, o sinal é normalmente o mecanismo que despoleta o processamento da máquina, fazendo-a passar de um estado a outro. Durante este processamento poderão ser enviados sinais a outras máquinas, contendo qualquer tipo de informação. Se não esperado, o sinal poderá ser retirado da fila e eliminado ou continuar na fila para tratamento posterior. As construções básicas associadas aos procedimentos de comunicação por sinais são: "SIGNAL", "SIGNALSET", "INPUT", "SAVE" e "OUTPUT". "SIGNAL" é a construção associada à informação transferida por sinais e "SIGNALSET" é usada para a descrição do conjunto de sinais passíveis de serem recebidos por um processo. "INPUT" é usada para a recepção de um sinal, podendo a manutenção na fila de um sinal indesejado ser feita por um "SAVE". "OUTPUT" é o mecanismo usado para o envio de sinais a outras instâncias do mesmo ou de outros processos.

Na segunda forma de comunicação, a informação é partilhada e pode ser acedida por várias instâncias de processos. Em SDL, toda a informação pertence a uma e uma só instância de um processso, a única com o direito de mudar o valor dessa informação. No entanto, através de dois pares de construções ("REVEALED"-"VIEW" e "EXPORT"-"IMPORT"), essa informação poderá ser vista (lida) por outras instâncias.

A primeira construção permite que determinada informação "REVEALED", pertencente a uma instância, possa ser vista por outra instância, sob ordem explícita de "VIEW". O valor observado pela segunda instância traduz sempre o valor da informação no momento da ordem. Esta construção é usada apenas por processos contidos no mesmo ambiente. Na segunda construção, o valor de determinada informação pode ser exportado pela instância proprietária através de uma ordem de "EXPORT" e visto por outras instâncias através de "IMPORT". Ao contrário da primeira construção, quando a instância proprietária muda o

valor da informação sem fazer novo "EXPORT", as outras instâncias não se apercebem da mudança, continuando, em nova ordem de "IMPORT", a observar o último valor exportado.

A terceira forma de comunicação entre instâncias dá-se pela criação do receptor pelo emissor da informação. Isto significa que, podendo uma instância criar outra do mesmo ou de outro processo através da construção "CREATE", a informação poderá ser transferida no momento da criação.

A estruturação em SDL é feita pela decomposição do sistema em partes e pela especialização das partes. Aspectos passíveis de estruturação em SDL são a representação, o funcionamento e as comunicações.

A representação é estruturada pela demarcação do sistema do seu ambiente e identificação das suas partes características. O sistema é caracterizado pela sua fronteira, tipos de informação usados e estrutura interna. A estrutura interna é, por sua vez, caracterizada pelos blocos constituintes, comunicações e funcionamento. As construções usadas em SDL na estruturação da representação do sistema são: "SYSTEM", "BLOCK", "PROCESS", "SERVICE", "PROCEDURE" e "CHANNEL SUBSTRUCTURE". "SYSTEM" demarca o sistema do seu ambiente externo, "BLOCK" delimita uma parte constituinte do sistema, "PROCESS", "SERVICE" e "PROCEDURE" representam aspectos de funcionamento e "CHANNEL SUBSTRUCTURE" canais de comunicação não ideais. Todas estas construções têm o conceito de fronteira. Este conceito não só permite o encapsulamento da entidade, com a definição das regras de acesso, como também serve para a implementação dos conceitos de herança de propriedades, como por exemplo visibilidade de variáveis e tipos.

A estruturação do funcionamento é feita pelo recurso às construções "SERVICE" e "PROCEDURE". O "PROCESS" modeliza o funcionamento de um conjunto de entidades (as suas instâncias), funcionando cada uma delas em simultâneo com as outras. Se a divisão do funcionamento de um processo é desejável, devem ser usados "SERVICEs". Estes têm a propriedade de não funcionarem simultaneamente, mas apenas um de cada vez no tempo. A "PROCEDURE" pode ser usada pelo "PROCESS" e pelo "SERVICE" para delimitar o funcionamento de um grupo de estados com acesso restricto.

A comunicação é estruturada através de duas construções: "ROUTE" e "CHANNEL". "ROUTE" define percursos de acesso dos sinais ao processo e é equivalente a "SIGNALSET".

"CHANNEL" é usado para modelizar um canal de comunicação entre partes constituintes do sistema ("BLOCKs"). O canal SDL é assumido ideal, no sentido em que todos os sinais chegam ao destino pela ordem de entrada no canal e não há perdas de sinais nem atrasos significativos do ponto de vista da especificação. Pode, no entanto, ser modelizado como não ideal pelo uso da construcção de "CHANNEL SUBSTRUCTURE", que permite a especificação do canal como se de um novo sistema se tratasse.

Dados, em SDL, são items de informação que assumem valores e são manipuláveis por operadores próprios. Estes dados são declarados e usados em "PROCESSes" e "PROCEDUREs". Existem tipos de dados ("SORTs", em SDL) pré-definidos, que poderão ser usados nas declarações: "INTEGER", "REAL", "NATURAL", "BOOLEAN", "CHARACTER", "PID", "DURATION", "TIME", "REAL" e "CHARSTRING". A criação de novos tipos com base em tipos já existentes é conseguida de uma forma simples usando as construcções "SYNTYPE", "ARRAY", "STRING", "POWERSET" e "STRUCT". Tipos novos podem também ser definidos, sendo para tal necessário definir o seu nome, valores assumidos, operadores (todos os que usem o valor do "SORT" como entrada - argumento, ou saída - resultados), e axiomas sobre os quais é construído o novo tipo.

5.2. DESCRIÇÃO DO SISTEMA

O sistema apresentado neste ponto e descrito em SDL, não inclui a Interface humana pelo facto de a linguagem não ser indicada para a representação deste tipo de aplicações. Por este motivo, a modelização de canais de pacotes e canais de sinalização também não será feita pois, fazendo sentido o uso destes conceitos apenas do ponto de vista do utilizador, são implementados na Interface humana. A descrição de uma sessão no sistema especificado em SDL é feita através da sua duração e dados necessários a cada uma das duas tarefas configuradas em cada UP. A tradução ou alteração da representação destes dados em canais é feita na Interface humana.

Devido à origem do projecto, a descrição formal teve que ser feita em Inglês, a lingua oficial usada nos projectos DAMS e UCOL. No entanto, sempre que um nome introduzido esteja relacionado com algo já apresentado nesta tese, será feita a respectiva associação. Todos os outros nomes, que não os introduzidos na descrição formal, serão apresentados em Portugês.

O sistema considerado será, basicamente, o SGDT no modo de funcionamento de avaliação de desempenho e nas suas componentes consideradas de maior complexidade de execução: controlo e comunicação. Aspectos de funcionamente considerados não importantes deste ponto de vista poderão ser pontualmente encapsulados em rotinas, com interfaces devidamente identificadas.

O sistema é constituído por 4 blocos, escolhidos de acordo com a configuração física desejada para o SGDT (Fig. 5.1): "ws", "pn1", "pn2", "pn3".

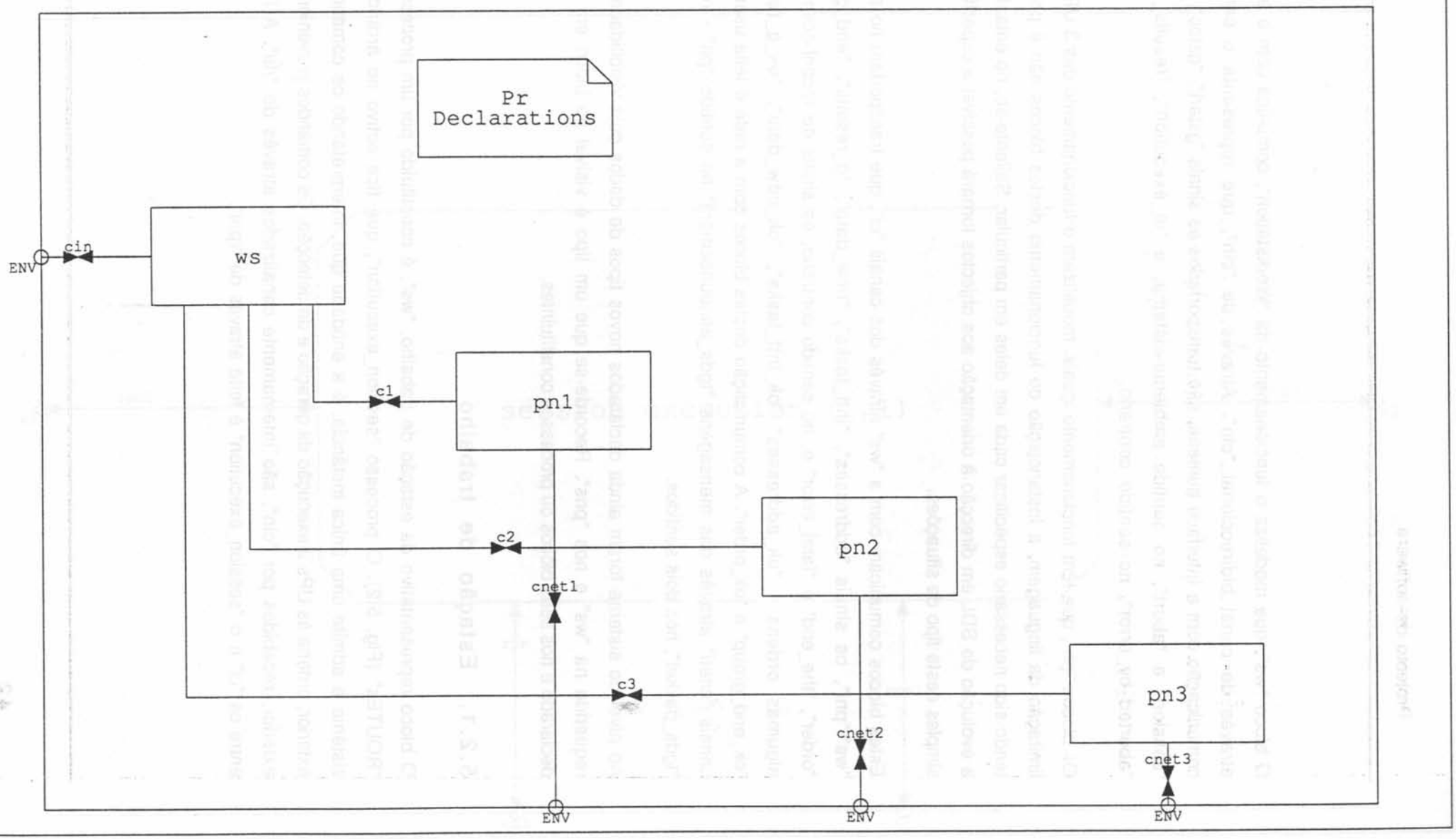


Figura 5.1 - Blocos constituintes do SGDT

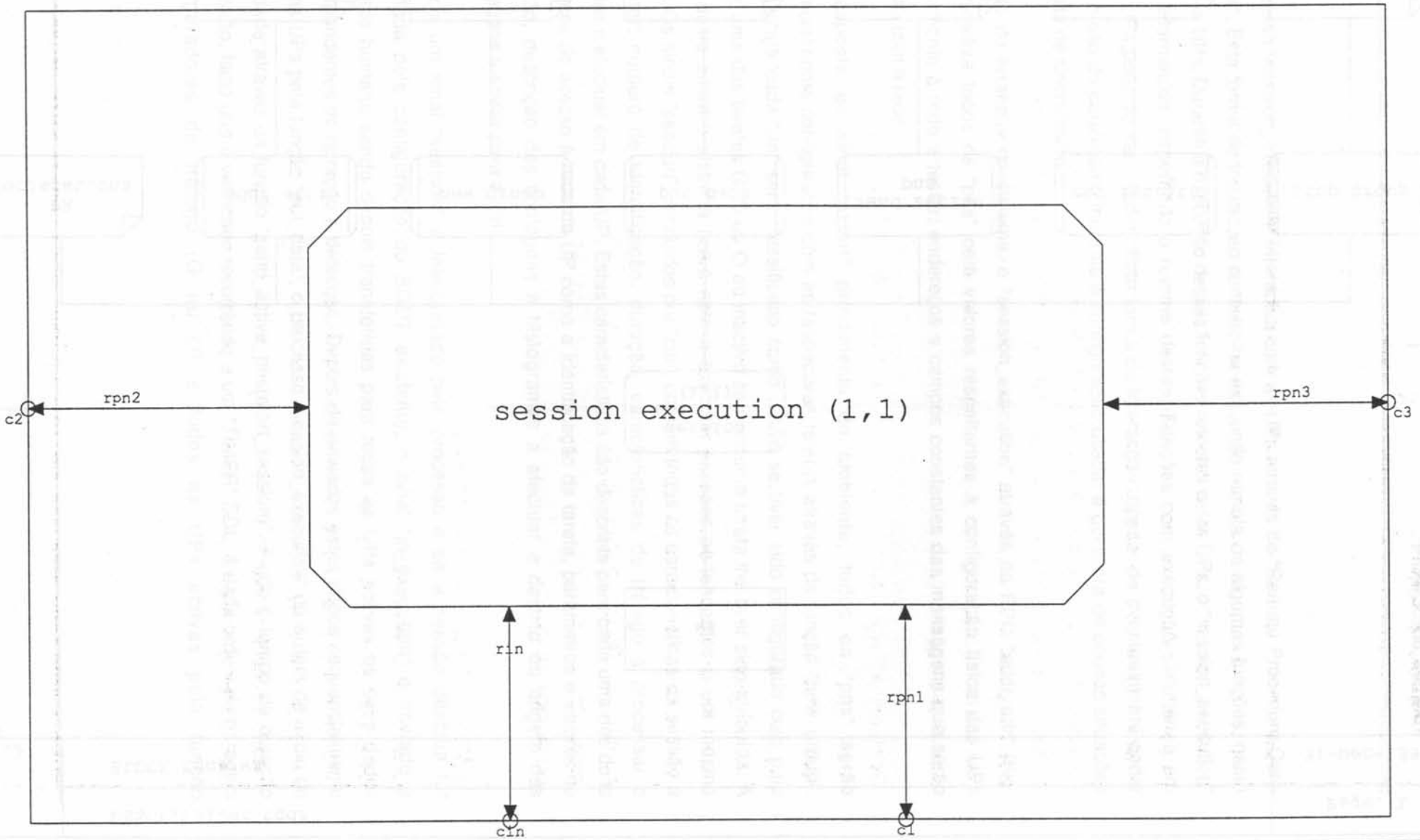
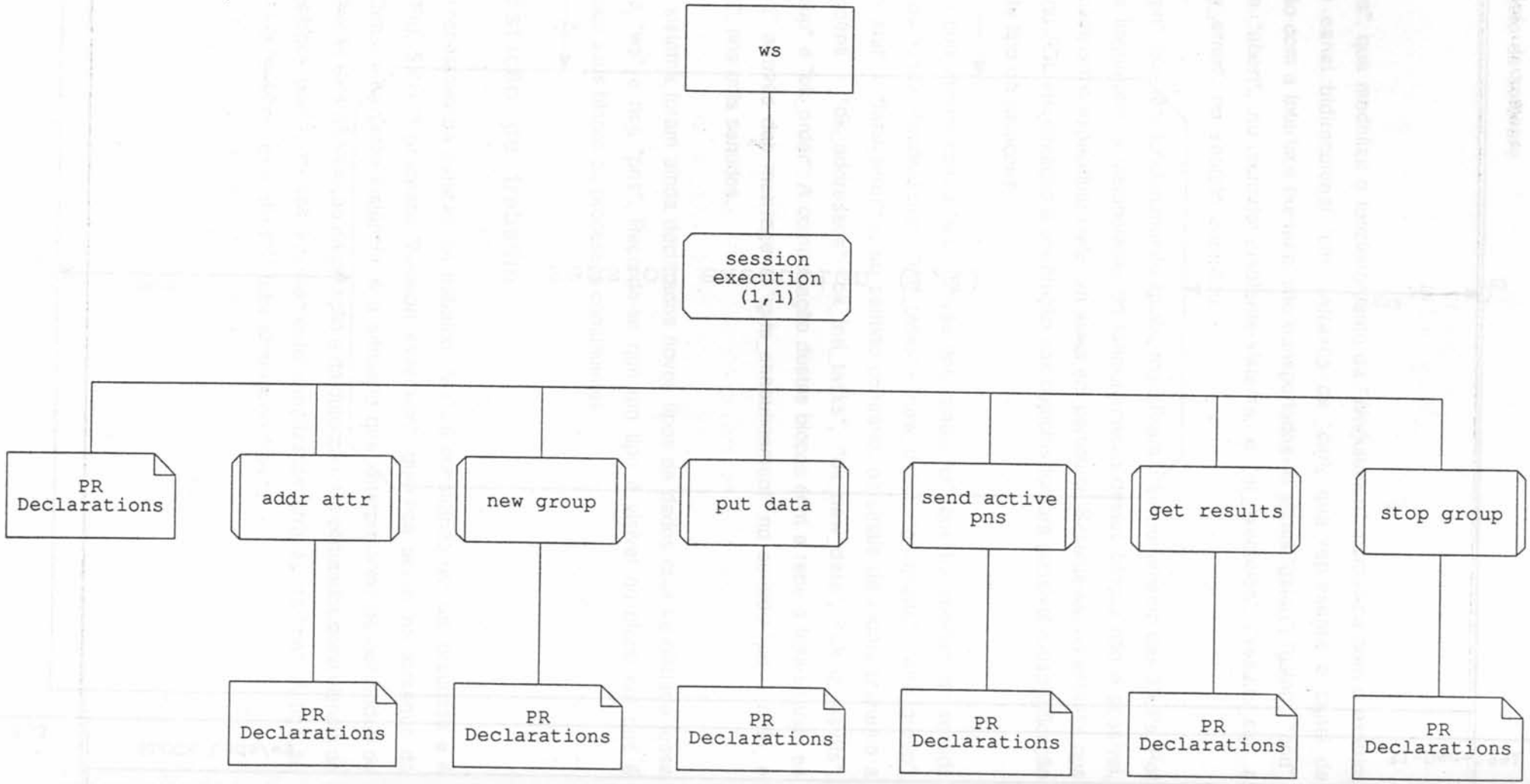


Figura 5.2 - Processo "session_execution" da WS


ws
session execution (1,1)
PR Declarations
addr attr
new group
put data
send active pns
get results
stop group
PR Declarations
PR Declarations
PR Declarations
PR Declarations
PR Declarations
PR Declarations

O processo "session_execution" interactua com as UPs através de "Remote Procedure Calls - RPCs". Esta forma de interacção consiste na execução remota de algumas funções, neste caso nas UPs. Durante a execução dessas funções remotas pelas UPs, o "session_execution" ficará adormecido, esperando o retorno destas. Funções com execução simultânea em várias UPs poderão ser usadas. Este forma de interacção, apesar de eventualmente poder limitar o uso do paralelismo inerente à configuração usada, é geradora de poucas situações de perda de sincronismo.

Quando do arranque do sistema, o "session_execution" através da RPC "addr_attr" (Fig. 5.3) inicializa todos os "pns" com valores respeitantes à configuração física das UPs relativamente à rede a testar: endereços e campos constantes das mensagens que serão trocadas com a rede.

Em resposta ao sinal "gstart" proveniente do ambiente, todos os "pns" serão sequencialmente configurados com as respectivas tarefas através da função "new_group". Nesta altura cada "pn" será classificado como activo se tiver sido configurado com pelo menos uma das tarefas G, D ou O ou inactivo se nenhuma tarefa lhe tiver sido atribuída. A partir desta altura o sistema ficará apto a executar sessões pertencentes a um mesmo grupo. Os sinais "session", recebidos por "cin", contêm todas as características da sessão a executar: número de identificação, duração, características do tráfego a processar e medidas a efectuar em cada UP. Estas características são descritas para cada uma das duas interfaces de acesso à rede da UP como a identificação da tarefa, parâmetros e valores de geração, descrição das contagens e histogramas a efectuar e destino ou origem das mensagens a enviar para a rede.

Quando um sinal "session" é interpretado pelo processo e se a sessão descrita for suportada pela configuração do SGDT existente, o sinal "in_execution" é enviado à Interface humana sendo depois transferidas para todas as UPs activas os seus dados correspondentes de geração e detecção. Depois de enviados estes dados sequencialmente para as UPs pela função "put_data", o processo "session_execution" dá ordem de início de actividade através da função "send_active_pns(start_session)". Findo o tempo de duração da sessão, facto que é verificado recorrendo a um "TIMER" SDL, é dada ordem de paragem dos geradores de tráfego (G ou O) a todos as UPs activas pela função

"send_active_pns(stop_generators)". Após a paragem destes, serão parados os detectores por "send_active_pns(stop_detectors)".

Os resultados são obtidos por chamadas à função "get_results" e transferidos para o exterior pelo sinal "results_sig". Este sinal, para além da identificação da sessão a que se refere, contém ainda para cada interface de acesso à rede de cada UP, a identificação da tarefa e a descriminação das contagens e histogramas efectuados bem como os respectivos valores.

Após execução de uma sessão o sistema fica apto a executar novas sessões ou a ser novamente configurado. Em caso de ser pretendida nova configuração, a anterior deverá ser explicitamente terminada através do sinal "gstop". Caso tal aconteça, todos as UPs serão avisados pela função "stop_group", cujo objectivo é fazer voltar as UPs a uma situação pré-definida. O sistema termina ou deixa de ficar activo quando é interpretado pelo "session_execution" o sinal "end". As UPs, nesta situação, terminarão também a sua execução.

5.2.2. Unidades de Processamento

Pelo facto de as UPs, "pns" na descrição formal, terem funcionamento iguais apenas uma delas será descrita. O bloco "pn" é constituído por 4 processos - "supervision", "generation", "detection" e "call_control" e 7 "ROUTEs", que interligam os processso entre si, à "workstation" e à rede a testar (Fig. 5.4).

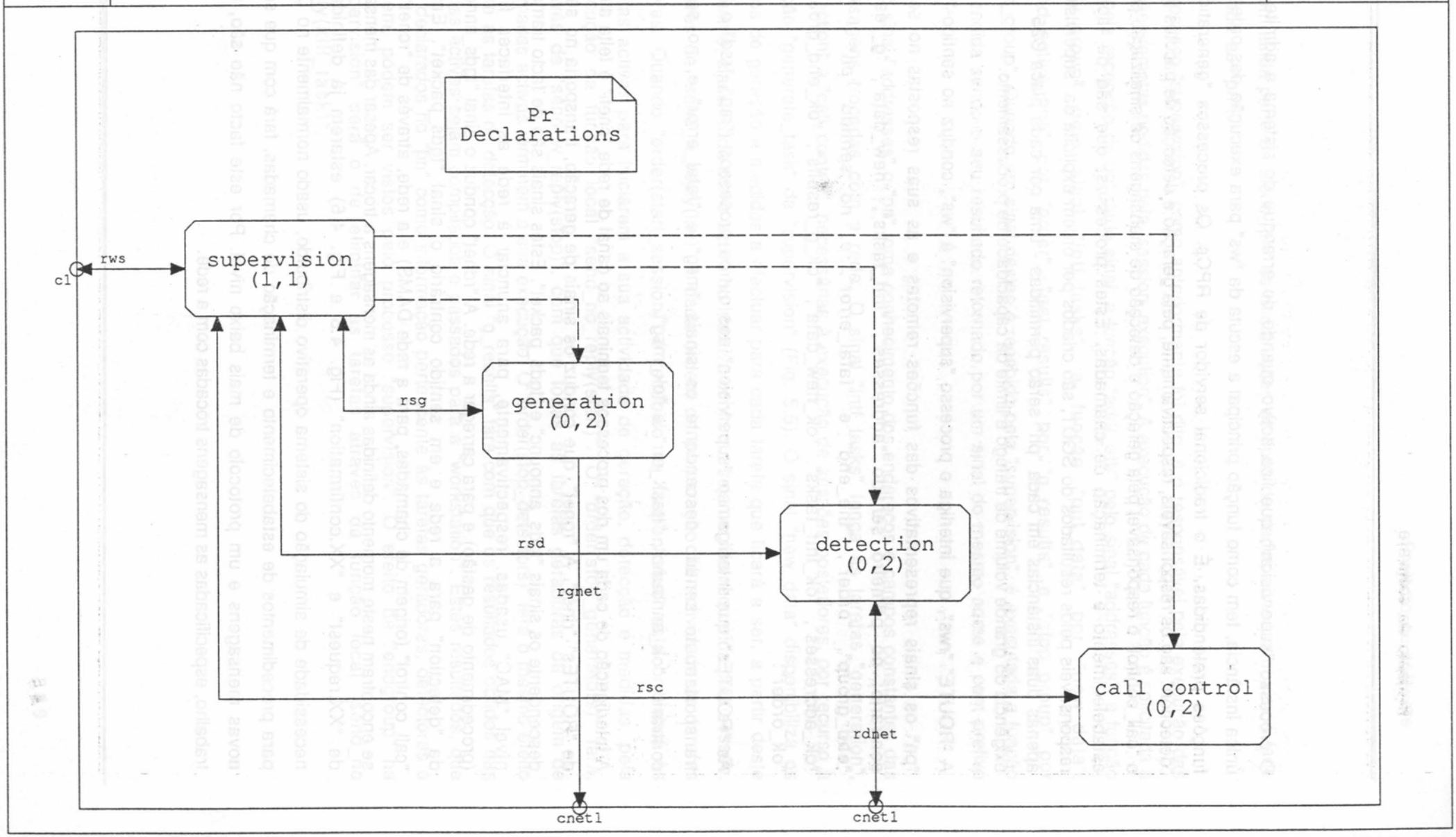


Figura 5.4 - Processos constituintes da UP

O processo "supervision", que fica activo quando do arranque do sistema e admite uma única instância, tem como função principal a escuta da "ws" para execução das ordens ou funções pretendidas. É o tradicional servidor de RPCs. Os processos "generation" e "detection" são os responsáveis, respectivamente, pela geração e detecção de pacotes sendo o "call_control" o responsável pela geração e detecção de mensagens de sinalização para o estabelecimento e terminação de chamadas. Estes processos, que são de facto os responsáveis pelos resultados do SGDT, são criados por ordem explícita da "supervision" e apenas duas instâncias em cada "pn" serão permitidas - uma por cada placa C25 - pela exigência do grande volume de tráfego e limite da capacidade de processamento.

A "ROUTE" "rws", que interliga o processo "supervision" à "ws", conduz no sentido "ws" - "pn" os sinais representativos das funcões remotas e as suas respostas no sentido contrário: no primeiro sentido "addresses", "intf_tasks", "new_data", "g_results", "end_group", "order", "the_end" e "fatal_error", e, no sentido "pn" - "ws", "ok_addresses", "ok_intf_tasks", "ok_new_data", "ok_g_results", "ok_end_group" e "ok_order".

As "ROUTEs" que interligam o "supervision" aos outros processos ("rsg", "rsd" e "rsc") transportam no sentido descendente os sinais "msg" e "fatal_error" e, no sentido contrário, "ok_announc", "ttask_err" e "ok_msg".

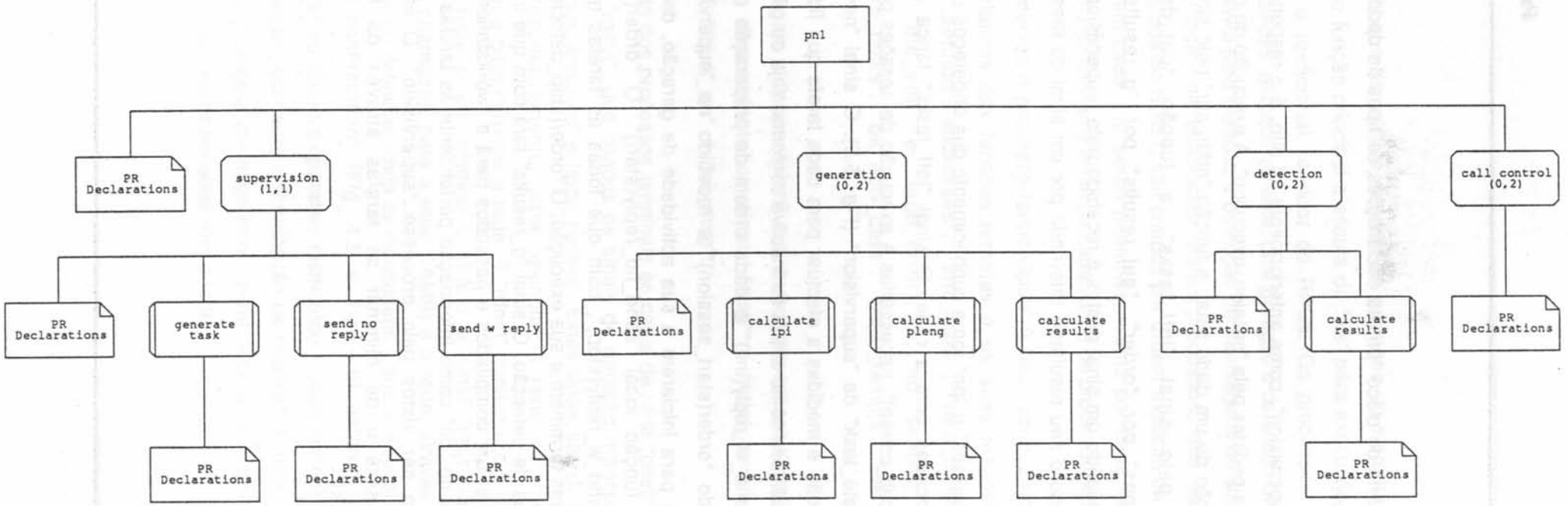
A interligação de cada um dos processos terminais ao canal de rede "cneti" é feita através de "ROUTEs" "rinet". A "rgnet", que conduz os sinais de geração, transporta no sentido descendente os sinais "tgds_announc" e "tgds_packet". Estes sinais são de facto tramas de nível "MAC" usadas, respectivamente, para anunciar à rede as interfaces físicas (procedimento de gestão) e para carregar a rede. A "rdnet" conduz o sinal "tgds_announc" da "detection" para a rede e em sentido contrário o sinal "tgds_packet". Entre o "call_control" (origem de chamadas, para a rede DAMS) e a rede, através de "rcnet", não se encontram neste momento definidas ainda as mensagens a trocar. Apesar das mensagens de "XX.request" e "XX.confirmation" (Fig. 4.5 e Fig. 4.6) estarem já definidas, a necessidade de simulação do sistema operativo distribuído, usado normalmente no DAMS para procedimentos de estabelecimento e terminação de chamadas, fará com que surjam novas mensagens e um protocolo de mais baixo nível. Por este facto não são, neste trabalho, especificadas as mensagens trocadas com a rede.

Fazem ainda parte do bloco "pn" as declarações de tipos de dados novos com visibilidade limitada ao "pn".

O processo "supervision", como anteriormente foi dito, é o responsável pela execução das funções remotas pedidas pela "session_execution". A execução de cada função é solicitada à UP pela recepção de um dado sinal: a função "addr_attr" pelo sinal "addresses", a função "new_group" pelo sinal "intf_tasks", a função "put_data" por "new_data", "send_active_pns" por "order", "get_results" por "g_results" e "stop_group" por "end_group". Quando um sinal destes é recebido pela "supervision", é executada a função pretendida sendo o seu resultado retornado por um sinal do mesmo nome e com prefixo "ok_".

Pelo sinal "addresses" o "pn" toma conhecimento dos endereços e campos constantes das mensagens trocadas com a rede. O sinal "intf_tasks" lança as tarefas "generation", "detection" e "call_control", necessárias à execução de sessões posteriores, por recurso à função "generate_task" da "supervision" (Fig. 5.5). O sinal "new_data" disponibiliza os dados de geração e medidas a efectuar para cada tarefa que ficará a ser, a partir deste momento, classificada como activa ou inactiva conforme seja ou não requerida pela sessão. Através de "send_w_reply(init)" é dada ordem de preparação de actividade às tarefas activas. Quando "order(start_session)" é recebido na "supervision" esta ordenará às tarefas activas para iniciarem a sua actividade de geração, detecção e medidas, pela execução da função local "send_no_reply(start)". O "order(stop_generators)" fará, através de "send_w_reply(stop)", com que todas as tarefas geradoras ou origem de chamadas activas terminem a sua execução. O "order(stop_detectors)" terá o mesmo efeito sobre as tarefas de detecção. O sinal "g_results" fará com que os resultados obtidos nas tarefas activas sejam compilados e passados para a "workstation". Estes resultados, que são declarados no "pn" como informação pertencente às tarefas geradoras, detectoras e origem, podem ser vistos pelo processo "supervision". O efeito de "endgroup" na "supervision" será o de eliminar as tarefas através da função local "send_no reply(kill_task)".

1.3	DESCRIPTION: tgds	Page: 1
	BLOCK tgds/pn1	31-Dec-1991



Como se depreende da exposição feita, a "supervision" serve-se de 3 rotinas para comunicação com as tarefas: "generate_task", "send_w_reply" e "send_no_reply". A diferença entre as duas últimas reside no facto de uma delas ("send_w_reply") adormecer o processo enquanto este espera pela resposta que, uma vez chegada, trará informação relativa ao sucesso da operação, enquanto que a outra apenas notifica as tarefas terminais do "pn".

O processo "generation" é a entidade responsável pela geração dos pacotes de acordo com os ritmos definidos pelo utilizador e pela execução das contagens e dos histogramas. A informação relativa à sua actividade, do tipo "INTERFACE_SESSION_DATA", descrita no apêndice C, é transferida para esta tarefa pelo "supervision" através de uma construção "View - Reveal". Este tipo de informação é constituída pela identificação da tarefa à qual os dados dizem respeito, características de geração, medidas a efectuar e destino físico dos pacotes gerados. As características de geração são descritas pela função de tráfego a gerar e pelos parâmetros correspondentes, sendo cada um destes caracterizados por identificação da função de distribuição, valores médio e desvio padrão. As medidas a efectuar indicam quais as contagens e histogramas que devem ser realizados durante a execução da sessão. Cada histograma é caracterizado pelo valor mínimo a partir do qual de efectuam contagens, largura de intervalo e número de intervalos. As unidades usadas na representação dos valores das grandezas de tempo e comprimento de pacote são, respectivamente, 0.1 μs e octeto.

A actividade do "generation" é regulada pelos sinais "msg(init)", "msg(stop)", "msg(kill)" e "msg(start)", sendo os dois primeiros enviados pelo "supervision" através da função "send_w_reply" e os últimos através de "send_no_reply". Assim, quando uma instância do processo é criada pela "supervision", é enviado para a rede o sinal "tgds_announc" usado para avisar a rede da futura actividade da interface física de suporte. Quando é recebido um sinal "msg(init)", todas as variáveis que irão conter informação das medidas são inicializadas. Ao sinal de "msg(start)" começarão a ser gerados os pacotes para a rede, representados em SDL pelos sinais "tgds_packet". Ao contrário das outras acções, que retornam depois da acção pretendida ter sido executada, no estado de geração o processo pode enviar assíncronamente ao "supervision" o sinal "ttask_err" com a indicação de erro de mau funcionamento.

Durante o estado de geração é determinado, de acordo com as leis estatísticas escolhidas, o próximo instante de lançamento de pacote para a rede. Este valor, enviado para um "Timer", fará com que o processo envie o sinal (pacote) para a rede no instante desejado, sendo nesta altura também calculado o seu comprimento e efectuadas as medidas definidas pelo utilizador. A extracção de valores de funções estatísticas é obtida pela fórmula $X=[F_X(z)]^{-1}$, sendo z um número aleatório com distribuição uniforme entre 0 e 1, e $F_X()$ a função cumulativa de uma função distribuição de probabilidade. As medidas são efectuadas, para cada pacote gerado, pelo incremento dos valores das contagens e dos histogramas.

Pela recepção de "msg(stop)" a actividade de geração terminará, ficando os resultados disponíveis para posterior recolha pelo "supervision" e o processo preparado para executar uma nova sessão - "msg(init)", ou para terminar - "msg(kill)". Os resultados são disponibilizados através da construção "VIEW - REVEAL" de uma variável do tipo "INTERFACE_RESULTS_DATA", descrita no apêndice C, que é constituída por um campo identificador da tarefa, um campo identificador das contagens efectuadas, o valor de cada contagem, identificador dos histogramas efectuados e, para cada um deles, identificadores de valor mínimo, largura de intervalo, número de intervalos, valor máximo e vector de 512 valores.

O processo "detection", cuja função principal é a detecção dos pacotes gerados por uma tarefa "generation", tem interfaces em tudo idênticas ao "generation". As mesmas mensagens de "msg(init)", "msg(stop)", "msg(kill)" e "msg(start)" são enviadas pelo "supervision" para regular a sua actividade, sendo os dados e os resultados de uma sessão disponibilizados através de construções "VIEW - REVEAL" de variáveis do tipo "INTERFACE_SESSION_DATA" e "INTERFACE_RESULTS_DATA". A interacção do "detection" com a rede é feita no sentido "pn" - rede pelo sinal "tgds_announc", anunciando à rede que mais uma entidade física estará activa gerando ou recebendo mensagens e, no sentido rede - "pn", pelo sinal "tgds_packet" que é o pacote enviado pelo "generation" através da rede. A detecção dos pacotes após um sinal "msg(start)" traduz-se em, para cada pacote recebido, ir incrementando valores das contagens e dos histogramas correspondentes.

O processo "call_control" apresenta uma interface com o "supervision" também semelhante à dos outros dois processos terminais, quer em termos de mensagens quer em

termos de dados e resultados. O funcionamento de origem e terminação de chamadas não é aqui especificado pela indefinição da interface com a rede (mensagens e protocolo). Acrescente-se, no entanto, que a sua integração no sistema será bastante simples, desde que se resuma a um único processo contendo um ou vários serviços, como se demonstrará no capítulo de implementação.

5.2.3. Temporizações

O conceito de temporização no SGDT é definido como o intervalo de tempo durante o qual é esperado um determinado acontecimento. Se, findo este intervalo, o acontecimento não tiver ocorrido, o processo em questão é declarado como em situação de mau funcionamento ou de erro. O objectivo último deste conceito é a detecção de situações de erro em processos vizinhos.

Um dos casos susceptíveis de temporização no SGDT é a espera de um sinal de confirmação por um processo que tinha previamente feito o pedido de execução de uma acção. A temporização será, neste caso, definida como o máximo intervalo de tempo admissível entre o envio do sinal de pedido e a recepção do correspondente sinal de confirmação. Outro caso é a espera de um pedido de execução de acção, sendo neste caso a temporização definida como o máximo intervalo entre dois pedidos de acções consecutivas.

Apesar de não especificadas formalmente, as situações que no SGDT devem ser temporizadas encontram-se identificadas. No "session_execution" devem ser temporizadas as 6 subrotinas usadas nas comunicações com as UPs. No caso do sinal de confirmação da acção pedida ("ok_xxx"), que além da informação de sucesso da acção serve também para sincronizar o processo, não ser recebido dentro de um intervalo de tempo pré-definido, a subrotina retorna erro de temporização, passando o processo ao estado de erro (ver 5.2.4). Da mesma forma, no "supervision" as subrotinas "generate_task" e "send_w_reply" também devem ser temporizadas pois, tal como as do processo anterior, também esperam sinal de confirmação de acção. Temporizações devem ainda ser aplicadas neste processo após a recepção dos sinais "new_data" e "order". Significa isto que, após os dados de uma sessão terem sido passados às UPs, devem ser definidos intervalos máximos de espera por sinais de início de actividade, fim de actividade e recolha de resultados. No caso de uma sessão poder vir a ter uma duração não previsível à partida, como acontecerá

na extensão do sistema especificado ao modo de funcionamento de Teste, será criado mais um sinal para sincronismo, que poderá ser encapsulado na "session_execution" numa subrotina de nome "continue()", e que será recebido periódicamente nas UPs durante o período de geração e detecção de tráfego. Garantir-se-à desta forma a detecção pela UP de uma falha de actividade na "ws" durante os períodos de interacção com a rede. Também no "supervision", uma situação de falha do acontecimento esperado durante o intervalo de tempo de espera conduzirá o processo ao estado de erro.

5.2.4. Situações de erro

Uma situação de erro no SGDT é definida como uma situação de falha ou mau funcionamento de alguma entidade, seja esta activa (processamento) ou passiva (transmissão). É sentida nas instâncias dos processos pela detecção de valores de erro pré-definidos.

O SGDT detecta situações de erro sem, pelo menos nesta versão, tentar recuperar delas. De facto a recuperação de erros seria um problema de complexidade tão grande como a do próprio sistema e que não faria sentido, atendendo aos fins a que se destina o SGDT. Não sendo necessário garantir baixas probabilidades de falhas do sistema, apenas é requerido que, quando obtidos os resultados, estes sejam fiáveis. Desta forma, o SGDT quando detecta um erro, avisa o utilizador do facto através de informação lhe seja útil (processo, bloco, interface e número do erro), terminando depois a sua actividade de uma forma controlada.

A estratégia usada para a implementação destes conceitos baseia-se nas características do próprio sistema, mais especificamente na sua estrutura de controlo. O controlo está estruturado em árvore: com o "session_execution" na raíz como processo pai, seguem-se 3 processos filhos "supervision", um em cada UP. Cada um destes poderá, por sua vez, ter até dois filhos que são duas quaisquer instâncias dos processos "generation", "detection" ou "call_control". Todos os processos filhos executam apenas as acções pedidas pelo pai que, regra geral, espera informação acerca do sucesso da operação. Em todo o sistema existe apenas uma excepção a esta regra: quando o "supervision" dá ordem de ínicio de actividade de geração e detecção através do sinal "msg(start)", pela rotina "send_n_reply()", os processos filhos não confirmam a execução desta acção, podendo no entanto enviar um sinal assíncrono de "ttask_err", durante a sua actividade.

Aproveitando o facto de existir sempre uma forma controlada de cada processo avisar o pai da ocorrência de uma situação de erro, a detecção de erros é feita como a seguir se descreve: cada processo tem uma váriavel local de erro ("errno") que em situações de funcionamento normal deverá ter valor zero. Quando o valor desta variável se altera, por ocorrência de um erro no processo, ou por mudança do seu valor através do sinal de confirmação de acção do filho, o processo avisa o seu pai, também na resposta de confirmação, e muda para o estado de erro, onde fará a averiguação da origem deste. Se o erro foi originado por si próprio, o processo, envia mensagem de "fatal_error" a todos os filhos e termina a sua execução. Se, pelo contrário, o erro tiver sido passado por um filho, o processo envia a mensagem de "fatal_error" aos outros filhos e termina a sua execução. A recepção de um sinal de "fatal_error", que pode ser sempre feita pelo processo e só é enviado pelo pai, faz com que o processo por sua vez envie o sinal de "fatal_error" a todos os seus filhos e termine.

A averiguação da origem do erro é feita com base num esquema de numeração especialmente criado -

P B I EE

- em que os dois algarismos menos significativos (**EE**) traduzem o número de erro num processo, **I** a interface física de acesso à rede, **B** o bloco onde este ocurreu e **P**, o processo que detectou o erro.

P	B	I	EE
"session_execution" - 1	"ws" - 0	sem significado - 0	0 0
"supervision" - 2	"pn1" - 1	"interface2" - 1	...
"generation" - 3	"pn2" - 2	"interface1" - 2	9 9
"detection" - 4	"pn3" - 3		
"call_control" - 5			

Como exemplo, o erro de número 9 da tarefa de geração, a correr na interface 1 da UP 2, será observado no exterior como o erro 32109. Da mesma forma o erro 26 da "session_execution" terá o código 10026.

6. REQUISITOS DE HARDWARE

Neste capítulo são identificados os requisitos básicos das placas dedicadas C25 (Fig. 4.10) em função das especificações produzidas no capítulo 4 e do projecto de software apresentado no capítulo 5. São também justificadas as opções tomadas e apresentado o diagrama de blocos da placa que servirá de suporte ao projecto da mesma. Estas tarefas, sendo apresentadas como o projecto preliminar de hardware cujo objectivo é a identificação da estrutura deste, são enquadradas na metodologia adoptada e descrita no capítulo 3 na fase de desenvolvimento 1.

As UPs, que se espera suportem simultaneamente até duas tarefas de geração, detecção ou origem de chamadas, têm requisitos temporais muito exigentes. De facto, verifica-se que, com taxas de pico para a geração e detecção de 50.000 pac/s, o intervalo de tempo disponível para tratamento de um pacote (geração, transmissão, medidas) é da ordem dos 20μs. A utilização de uma "placa mãe" tradicional, por exemplo baseado num 68030, com um sistema operativo simples (necessário para o suporte dos pacotes de software para a "Ethernet"), ainda que auxiliado por um coprocessador matemático, tornaria o processamento de tão elevadas taxas bastante problemático, senão mesmo impossível. A escolha de um processador com maior capacidade de processamento, por exemplo o RISC Am29000 da AMD ou ainda um "Transputer", tornaria o custo quer da placa quer do software elevado além de, mesmo assim, não serem garantidas as taxas de geração e detecção requeridas.

A solução adoptada e apresentada no capítulo 4 foi a escolha de paralelismo pelo uso de mais do que uma placa. Por um lado, poderia continuar a ser usada uma placa tradicional, barata, com sistema operativo para suporte das aplicações de rede e, por outro, através de placas dedicadas, poder-se-iam atingir as taxas de processamento desejadas desde que se escolhesse convenientemente o hardware constituínte. As placas dedicadas que, pelo número de tarefas simultâneas desejadas pelos projectos UCOL e DAMS, devem ser duas, deveriam ainda ser iguais por razões de versatilidade, flexibilidade e economia de desenvolvimento, e ser capazes de suportar interfaces a estas duas redes .

Também por razões de versatilidade e flexibilidade era desejável que a entidade activa de processamento desta placa fosse programável, para que várias tarefas pudessem ser executadas numa mesma placa. Um processador seria a solução desejável. De entre alguns com sistema de desenvolvimento disponível no INESC, o TMS320C25 (C25) da Texas Instruments demonstrou, pela execução de testes sobre programas de geração de pacotes [26], ser capaz de processar e tranferir para a rede pacotes às taxas desejadas. Este processador de sinal digital, usado normalmente em aplicações de processamento de sinal e comunicações, tem uma arquitectura de Harvard de 32 bits e interface externa de 16 bits. Endereça 256 kbytes de memória dividida entre dados e programa, 16 portas de IO sendo o seu ciclo de instrução de 100 ns [27]. A interface da placa dedicada com a placa mãe [..] deveria ser VME [28], tendo este barramento sido escolhido por ser um stantard muito usado na Europa e pelo facto de placas com este tipo de interface, bem como os circuitos para implementação da mesma, terem boas relações de qualidade/custo e simplicidade/custo. A comunicação física da placa dedicada com as redes a testar deveria ser feita pela implementação de interfaces próprias que seriam ligadas ao barramento local do processador. De facto, a alternativa de desenvolver uma interface que interligasse a rede ao barramento VME seria inviabilizada pelo facto de serem necessárias taxas de transferência de 60 Mbit/s com determinismo assegurado, impossíveis de cumprir quer pelo VME quer por qualquer outro barramento genérico.

Atendendo ao tipo de interfaces exigidas e à especificidade das tarefas a serem suportadas, definem-se da seguinte forma os requisitos internos da placa dedicada C25:

- A interface VME deverá ser definida como a de um periférico típico não inteligente, comportando-se a placa sempre como um escravo em relação às outras placas "master" do barramento [28]. Não necessitando a placa de se apoderar do VME para transmissão de dados, quando desejar comunicar com um "master" deverá gerar uma interrupção para que este tome as devidas acções. A consequência imediata deste requisito é a simplificação desta interface.

- As interfaces com as redes deverão ser ligadas ao barramento local do C25 e implementadas com base em dispositivos programáveis (PAL ou EPLD) ou usando, com as devidas adaptações, circuitos integrados para acesso a redes normalizadas, como sejam o conjunto de circuitos para o FDDI da família "Supernet" da AMD ou o controlador de

Ethernet da família Am7990 também do mesmo fabricante. Apenas uma das interfaces terá que ser suportada de cada vez pela placa.

- Deverá suportar ainda um temporizador para a implementação do "Timer" da geração e de previsíveis temporizações para o "call_control". Atendendo a que intervalos entre pacotes consecutivos, ou seja, intervalos de tempo colocados no "Timer" são da ordem de 20 µs (50.000 pac/s) e tendo por objectivo erros inferiores a 1%, uma resolução de 200 ns seria desejável. Não são de prever valores máximos para as temporizações superiores a 3 minutos, que poderiam eventualmente acontecer no "call_control" para a duração de chamadas.

- Deverá conter um relógio, ou seja, um contador de tempo para o qual será exigida uma resolução semelhante à do temporizador, para a medida de intervalos de tempo entre a geração de pacotes consecutivos. Atrasos extremo a extremo na rede são supostos superiores a 2ms [29], pelo que, mesmo assumindo o projecto para erros de 1% (20 µs), esta medida não é a limitativa para a resolução do relógio.

- A descarga de programas através do VME para a memória da placa dedicada, feita pela placa mãe, deverá ser possível.

A solução de base apresentada para a placa dedicada C25 (Fig. 6.1), e que serviu de referência ao seu projecto, é pois constituida por 6 blocos: interface VME, interface rede, temporizador, relógio, memória e C25.

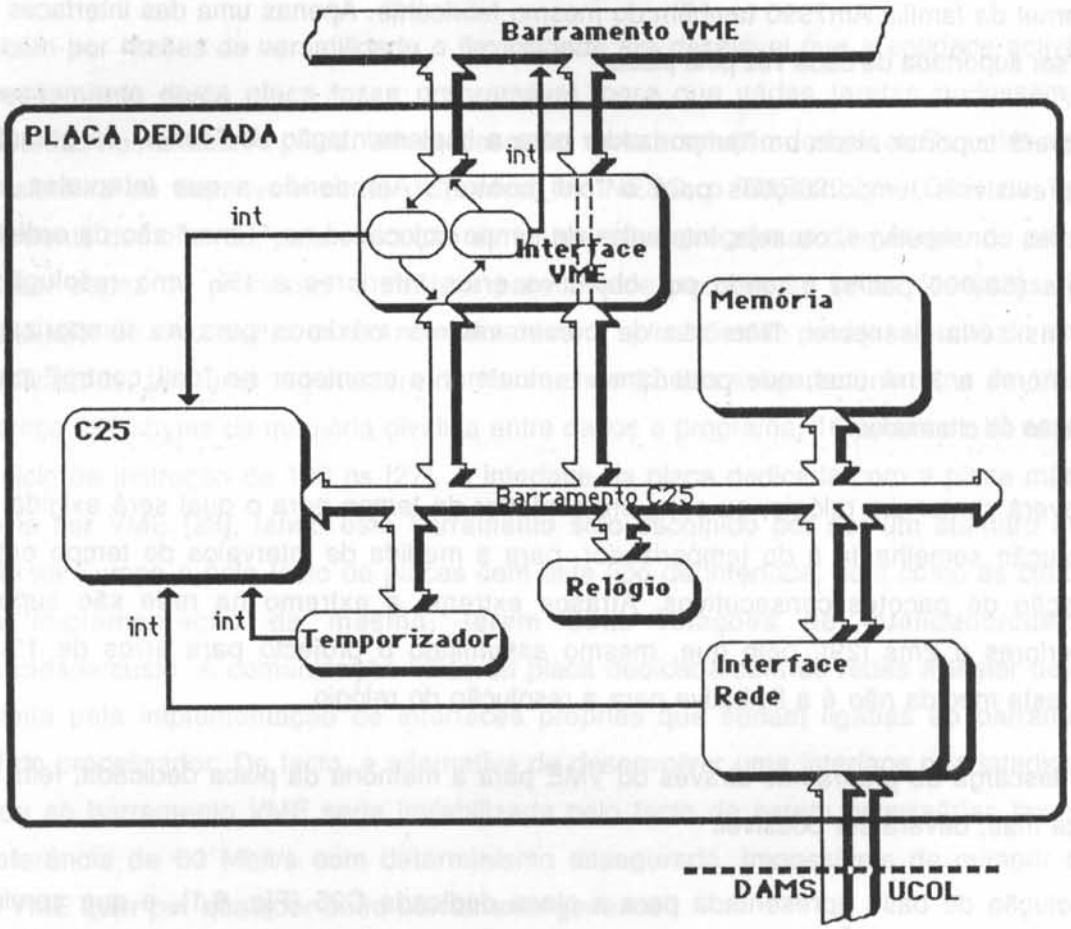


Figura 6.1 - Diagrama de blocos da placa dedicada C25

A interface VME, do ponto de vista da placa, deverá suportar dois modos de comunicação. No primeiro, e pela paragem da actividade do processador, a interface deve fazer com que toda a memória e portas do C25 possam ser acedidas através do VME. Isto permitirá quer a descarga dos programas e dados das sessões para a memória do C25, quer o acesso a portas, nas quais se encontra por exemplo o relógio que será acertado pela entidade superior, implementada na placa mãe (procedimento não descrito em SDL por se ter usado o conceito de tempo global "NOW"). Pelo segundo modo, será possível a comunicação com o C25 através de mensagens, por meio de duas zonas "dualport" unidireccionais. Quando uma mensagem é enviada pelo "master" do VME, o C25 será avisado por interrupção. Quando, em sentido contrário, o C25 pretender comunicar com um "master", escreve a mensagem na outra zona unidireccional, que gerará uma interrupção à placa mãe.

A interface rede, ligada ao barramento local e gerida, como todos os controladores, por registos de estado e controlo, avisará o C25 sempre que uma mensagem seja transmitida ou recebida para ou da rede.

O temporizador, com resolução mínima de 200 ns e 32 bits de largura, deverá também avisar o C25 sempre que uma temporização expire. O relógio terá também uma resolução mínima de 200 ns e 32 bits de largura.

A memória, que deverá ocupar todo o espaço de endereçamento do C25, deverá não prejudicar a sua performance, isto é, deve ser acedida sem gerar "wait states".

O C25, pelo facto de se encontrar numa placa cujo comportamento em relação ao VME é de escrava não inteligente, deverá oferecer o controlo do seu funcionamento (funcionamento ou não funcionamento) ao exterior, isto é, à placa "master" do VME.

7. ESTRATÉGIA DE IMPLEMENTAÇÃO DO SOFTWARE

Este capítulo pretende ser um guia não exaustivo para a implementação do SGDT e usa a descrição SDL apresentada no apêndice C como referência. Não é pretendida a apresentação dos algoritmos de todos os programas necessários, mas apenas sugeridas formas de implementar os aspectos de SDL considerados pelo autor como menos evidentes. Não é também pretendida a obtenção de um conjunto de regras de tradução do SDL para uma linguagem de programação estruturada, pelo facto do objectivo deste trabalho de tese ser a concepção de um sistema de geração e detecção de tráfego e definição da sua estratégia de implementação. Na metodologia descrita no capítulo 3, este guia poderá ser enquadrado na fase de desenvolvimento 2 (Projecto detalhado), cujo objectivo é a obtenção de cada um dos módulos de software e de hardware devidamente projectado e documentado. Pelo facto de não ser exaustivo e por não cobrir os aspectos de hardware, deverá ser entendido apenas como parte desta fase.

A tradução directa de uma especificação SDL para uma linguagem de programação, como por exemplo o C, não é normalmente possível pelo facto de assuntos como por exemplo, no caso do SGDT, a sincronização de relógios e descarga de software não terem sido abordados no projecto em SDL. Por outro lado, uma tradução universal, isto é, tradução para ambientes (sistemas operativos) diferentes também não é possível, pelo facto de cada um destes ter formas distintas de, por exemplo, implementar a comunicação entre processos. Atendendo a estes factos será usada, na exposição deste capítulo, a seguinte estratégia:

- Para o sistema, ou seja, para todo o SGDT, assumem-se as equivalências de "Block" a ambiente ou sistema operativo, "Process" ao tradicional programa de computador do qual se podem correr vários processos (instâncias em SDL) e "Procedure" à função usada numa linguagem de programação como por exemplo no C.

- Por ser de equivalência menos directa, é sugerida uma forma de implementar a máquina de estados usada na descrição do funcionamento do processo.

- Para cada ambiente, e atendendo às características próprias do SGDT, são sugeridas formas de implementação das "Routes" SDL, para comunicação entre processos, e dos canais para comunicação com os outros ambientes. São ainda sugeridas possíveis formas de implementação do "Timer" SDL.

- Para cada processo são indicadas formas de implementação das construções de "OUTPUT" e "INPUT" dos sinais.

- As outras construções SDL não serão abordadas por serem consideradas de tradução/ implementação menos problemática para um conhecedor de linguagens de programação com conhecimentos básicos de SDL.

Aspectos de carácter genérico não descritos na especificação SDL, como facilidades que os programas poderiam oferecer, ficheiros usados, sincronização de relógios, serão também apresentados em alturas apropriadas.

7.1. ASSUNTOS NÃO ESPECIFICADOS EM SDL

Dois assuntos não projectados na especificação formal por serem tipicamente característicos da fase de implementação, são a sincronização dos relógios das placas dedicadas, usados como dispositivos distribuidos com informação de tempo, e a descarga de software para as placas dedicadas. Embora não directamente relacionados com a aplicação, motivo pelo qual o SDL não permite o seu tratamento, os procedimentos relacionados com estes dois assuntos acabam, em última instância, por ser implementados na aplicação.

A sincronização de relógios, tão característica dos sistemas distribuidos de execução em tempo real, consiste em fazer com que dois ou mais relógios independentes apresentem, em qualquer instante, diferenças de tempo inferiores a um limite Δt. Isto significa que em qualquer instante, sendo t_i o tempo apresentado pelo relógio i e t_j o tempo apresentado pelo relógio j, t_i-t_j deverá ser sempre inferior a Δt, quaisquer que sejam i ou j. A ultrapassagem deste limite, ou seja, a perda de sincronismo, é devida normalmente a dois motivos[30]: o acerto do valor inicial dos relógios, que dificilmente é feito com o mesmo valor, e a deriva dos relógios que, por serem alimentados por cristais diferentes, apresentam ao fim de algum tempo valores diferentes. Isto significa que, mesmo que dois

relógios sejam acertados com o mesmo valor, ao fim de um determinado intervalo de tempo, os valores desses relógios apresentam uma diferença superior ao limite Δt.

No caso do SGDT, o conceito de tempo distribuído, ou seja, o uso de mais do que um relógio, aplica-se apenas na medida dos atrasos dos pacotes na rede. Nesta operação de medida, os pacotes são enviados para a rede pelo gerador com a informação do instante em que foram gerados. Na detecção, o atraso é calculado como sendo a diferença entre o instante de chegada, medido no relógio do detector, e o instante de geração veículado pelo pacote. Sendo esperados atrasos de pacotes de cerca de 2ms e se se pretender um erro de medida inferior a 1%, a diferença máxima admissível entre dois quaisquer relógios das placas C25 deverá ser inferior a 20μs. Uma vez que as interfaces com a rede utilizadas permitem que todos os relógios sejam alimentados por esta (derivas iguais), o acerto inicial dos relógios poderá ser feito com diferenças máximas de 20μs, não havendo necessidade de posteriores operações de ressincronização [31].

Esta operação de acerto dos 6 relógos do SGDT (um por placa dedicada) é feita pela difusão de um pacote pela Ethernet, que será recebido por todos as UPs em instantes que diferem entre si da diferença dos atrasos de transmissão, que deverão ser medidos e determinísticos. Quando o pacote chega à UP, esta, que já deverá estar preparada para a sua recepção, deve também garantir tempos determinísticos, que deverão ser medidos entre a chegada do pacote e a escrita dos valores iniciais nos relógios. Ao fim deste processo, e pela escolha adequada dos valores a escrever em cada um dos relógios, uma diferença máxima de 20μs entre dois quaisquer relógios deve ser garantida.

Apesar de nenhum estudo rigoroso ter sido feito, são sugeridas duas estratégias para a implementação da sincronização, a primeira por software e segunda por hardware. Ambas são baseadas no envio de um pacote de difusão pela Ethernet. A primeira estratégia assume que, na UP e uma vez chegado o pacote, qu é sentido pelo processador através de uma interrupção gerada pelo controlador de Ethernet, deverá ser chamada uma função que escreva os valores iniciais nos relógios da primeira e segunda placas C25. Na estratégia de hardware, o sinal de interrupção gerado pelo controlador de Ethernet é usado como interrupção directa a cada um dos C25, que farão o acerto dos relógios das suas placas. Ambas as estratégias assumem procedimentos prévios de aviso de acção de sincronização.

São apresentados nos pontos que se seguem as funções necessárias à sincronização pelo primeiro processo.

O segundo assunto mencionado neste ponto refere-se à descarga do software nas placas dedicadas C25. Como as tarefas a correr nestas placas - geração, detecção e origem de chamadas - são desenvolvidas em sistema próprio, é necessário transferir estes programas, armazenados em formato COFF [32], numa primeira fase para a placa mãe e numa segunda fase desta para as placas dedicadas. Na primeira transferência pode ser usado software de rede disponível para o efeito, como por exemplo o FTP [33], sendo, no entanto, necessário para a segunda fase a construção de um programa que faça a descarga dos ficheiros COFF de um disco, real ou virtual ("ramdisk"), da placa mãe para as placas dedicadas. Este programa será usado pelo supervisor, em alturas adequadas, para a operação de activação das tarefas terminais.

7.2. MAQUINA DE ESTADOS

A máquina alargada de estados finitos, tal como foi descrito no capítulo 5, é o modelo adoptado em SDL para a representação do funcionamento sendo usada nas contruções "Process", "Procedure" e "Service". Um exemplo de uma máquina deste tipo representando um "Process" com 4 estados é apresentado na Fig. 7.1. Nesta figura a representação dos estados é feita por circunferências e a representação das transições pelos ramos que as unem. Os ramos, que são orientados do estado inicial para o estado final, contêm na parte inicial as condições que originam a transição, seguidas de uma função na qual são encapsuladas as actividades a desenvolver pela máquina. Ao contrário do SDL que individualiza a representação do ramo, é apresentado na figura todo o processo de uma forma global.

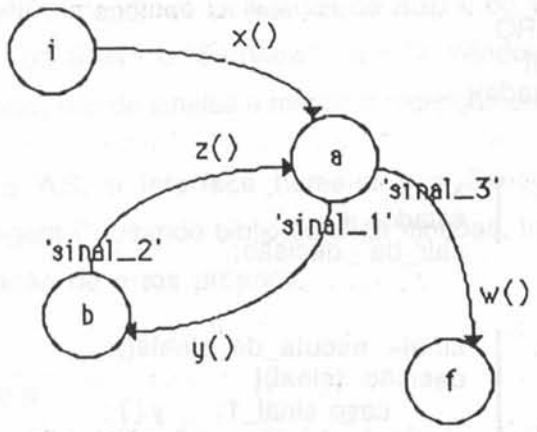


Figura 7.1 - Máquina de estados SDL

A implementação desta máquina, numa linguagem de programação estruturada, poderá ser feita através do uso de decisões múltiplas que, por exemplo na linguagem C, são implementadas com a construcção "switch". Um algoritmo para esta máquina pode ser o que a seguir se descreve:

```
estado= i;
activo= VERDADEIRO
enquanto (activo){
    decisão (estado){

        caso i:  | x ( ) ;                                  |
                 | estado= a;                               |
                 | sair_da _decisão;                        |

        caso a:  | sinal= escuta_de_sinais();               |
                 | decisão (sinal){                         |
                 |     caso sinal_1:  y ( ) ;               |
                 |                    estado= b;            |
                 |                    sair_da_decisão;      |
                 |     caso sinal_3:  w ( ) ;               |
                 |                    estado= f;            |
                 |                    sair_da _decisão;     |
                 |     }                                    |
                 | sair_da_decisão;                         |

        caso b:  | sinal= escuta_de_sinais();               |
                 | decisão (sinal){                         |
                 |     caso sinal_2:  z ( ) ;               |
                 |                    estado= a;            |
                 |                    sair_da_decisão;      |
                 |     }                                    |
                 | sair_da_decisão;

        caso f:  activo= FALSO;
                 sair_da_decisão;
        }
    }
```

As declarações limitadas por | | são semelhantes às que se encontram descritas nas especificação SDL, como pode ser observado na descrição no apêndice C, sendo apenas necessário, na implementação do algoritmo, a introdução das declarações não limitadas.

7.3. ESTAÇÃO DE TRABALHO

A estação de trabalho usada no SGDT para centralizar o controlo das unidades geradoras e detectoras de tráfego é uma "workstation" da SUN com sistema operativo próprio, o SunOS,

que é basicamente um UNIX integrando facilidades do BSD e do SYSTEM V. As interfaces gráficas oferecidas são a da SUN - o "SunView" - e o "X Windows", que se caracterizam pela iconização dos objectos, uso de janelas e menús e recepção de ordens através do rato.

O software a correr na WS, a Interface humana e o Executor de sessões, serão desenvolvidos, em linguagem C, usando bibliotecas de funções, tradutores, compiladores e auxiliares para identificação de erros próprios.

7.3.1. Interfaces

A definição da estratégia de implementação do SGDT na WS passa pela caracterização do tipo de interfaces usadas entre os 2 processos suportados (Interface humana e Executor de sessões), entre a Interface humana e o utilizador e entre o Executor de sessões e as UPs (Fig. 7.2). A caracterização destas interfaces consiste em definir os canais necessários à comunicação entre os processos e entre estes e o exterior, aproveitando as facilidades oferecidas pelo ambiente, ou seja, pelo sistema operativo.

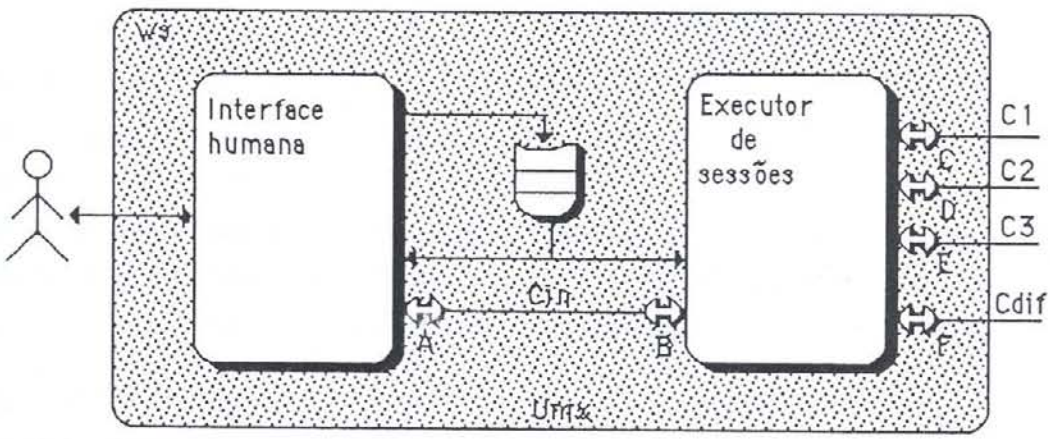


Figura 7.2 - Modos de comunicação na "WS"

A interface entre o utilizador e o processo Interface humana deve ser feita através de um monitor gráfico e colorido, teclado e rato.

Para as comunicações entre os processos Interface humana e Executor de sessões são sugeridos os mecanismos da Fig. 7.2: uma fila de mensagens [34] e dois "sockets" [35] (A e B). A fila, usada só para os sinais "session" da descrição SDL, implementa o conceito de

grupo de sessões. Todas as sessões que se encontrem nesta fila têm em comum o facto de exigirem a mesma configuração do SGDT. Estas mensagens, que são colocadas na fila pela Interface humana, são consumidas pelo executor de sessões, para as executar, ou pela Interface humana, no caso de o utilizador pretender retirar da fila uma sessão específica. Na implementação, a mensagem passada para a fila deve ser uma referência a um ficheiro, guardado em directório próprio, no qual se encontram as características de facto da sessão a executar, ao contrário do sinal "session" na descrição formal, que contem toda a informação da sessão.

Os restantes sinais de controlo ("gstart", "gstop", "end", "abort", "in_execution", "results_sig" e aborted_by_error") são transferidos através dos 2 "sockets" representados na figura, que deverão pertencer ao domínio Unix e serão usados para a implementação virtual do canal "cin" da descrição SDL. Da mesma forma que o sinal "session", também o sinal que retorna os resultados, o "results_sig", deve ser uma referência ao ficheiro no qual foram previamente armazenados os resultados pelo Executor de sessões e que deverá ficar também relacionado com o ficheiro de dados, por exemplo através do uso do mesmo nome mas com uma extensão diferente.

A comunicação entre o Executor de sessões e as UPs é feita também através de sockets, designados na Fig. 7.2 por C, D, E e F. Os três primeiros usados para a implementação virtual dos canais "c1", "c2" e "c3" são do tipo TCP, um protocolo usado normalmente para a implementação de software de rede do tipo cliente-servidor e que corre sobre a rede IP, que é um standarde de facto. O "socket" F usado para a implementação de um canal de difusão, não descrito na especificação formal, é do tipo UDP [33].

Pelo último "socket" passarão os sinais "order" no sentido WS - UP e "ok_order" em sentido contrário, que são usados para iniciar, terminar e sincronizar simultâneamente todas as UPs. Pelos outros "sockets" passarão os restantes sinais: "addresses", "intf_tasks", "new_data", "g_results", "end_group", "order", "the_end" e "fatal_error" no sentido descendente e "ok_addresses", "ok_intf_tasks", "ok_new_data", "ok_g_results", "ok_end_group" em sentido contrário.

7.3.2. Temporizador

As situações passíveis de contagem de tempo na WS são a temporização da duração da sessão em execução e a temporização das respostas de "ok_xxx" nas funções executadas nas UPs. Se por um lado as funções de manipulação dos "sockets" já oferecem facilidades de implementação de temporizações, já para a medida da duração da sessão isto não acontece. Pelo facto de no Executor de sessões poder ser recebido um sinal de "abort" enviado pela Interface humana, durante o estado de espera do sinal de fim de sessão, proveniente do "Timer", o processo não pode usar uma função de espera que o deixe bloqueado. Entenda-se por bloqueado ficar a executar esta função sem que outras actividades do mesmo processo possam ser executadas, como por exemplo a verificação de existência de uma mensagem de "abort" no "socket" correspondente. Por este motivo os temporizadores são implementados pelo uso de funcões que não adormeçam o processo. Basicamente trata-se de pedir ao sistema operativo que, terminada a temporização, envie ao processo um "Signal" Unix [36], que será tratado por este numa rotina assíncrona, ou de interrupção. Desta forma, o processo poderá ir executando outras acções durante a temporização.

7.3.3. Interface humana

A Interface humana deverá ser implementada sobre uma das interfaces gráficas suportadas pela WS. Do ponto de vista das comunicações, deverá monitorar a actividade do utilizador e escutar o "socket" A, por onde serão recebidos os sinais provenientes do Executor de sessões. A manipulação da fila será feita só por ordem explícita do utilizador ou seja sincronamente.

Do ponto de vista de funcionamento, deve ser capaz de virtualizar, ou seja dar ao utilizador a noção de objecto, as seguintes entidades: configuração, grupo com fila de espera associada, sessão, canal de pacotes e canal de sinalização. Relativamente à configuração, deve permitir ao utilizador atribuir tarefas às UPs. No que diz respeito ao grupo deve permitir inicializar, colocar uma sessão em fila, retirar uma sessão da fila e observar o estado de cada sessão do grupo (espera, execução ou completa). Sobre a sessão deve permitir a definição da origem e destino das suas ligações constituintes e a sua duração.

Deve ainda permitir, para as ligações, definir as características de geração, contagens e histogramas, visualizar os resultados associados, se já tiver sido executada a sessão que os contem, nomeadamente a média, o desvio padrão e correlações com curvas estatísticas conhecidas. De uma forma genérica, deverá ainda limitar todos os valores aceites e traduzir valores introduzidos pelo utilizador para valores aceites pelo sistema.

7.3.4. Executor de sessões

As comunicações do Executor de sessões com o seu exterior, tal como foi descrito em SDL, é feito apenas através de sinais. Isto significa que a passagem de informação no sentido Executor de sessões - exterior, ou seja, o envio de sinais por este processo, deverá ser sempre feito por um dos "sockets" pertencentes ao processo e descritos na Fig. 7.2, isto devido ao tipo de interface escolhida. A implementação da construção "OUTPUT" traduz-se, muito simplesmente, no envio de um sinal pelo "socket" adequado.

Pelos "sockets" TCP [33] (C, D e E), orientados às ligações, os sinais são enviados através de funções próprias. Atendendo ao modelo escolhido para as comunicações com as UPs, as funções de comunicação sobre os "sockets" são chamadas pelas funções remotas do tipo cliente-servidor já apresentadas no capítulo 5, que são "addr_attr()", "put_data()", "get_results()", "stop_group()" e "new_group()". Além destas funções, são ainda implementadas sobre estes sockets as funções "probe_pn()", usada para verificar se a UP está na rede, "read_data()" e "write_data()", usadas para ler ou escrever um grupo de "bytes" em qualquer posição de memória da placa mãe ou das placas dedicadas, já que estas têm a sua memória mapeada no espaço global de endereçamento VME, e "reset()", que faz renascer numa UP o Supervisor, limpo de erros e pronto a executar o seu programa do ínicio sem se fazer "reset" a todo o sistema. Pelo facto destas funções só comunicarem com uma UP de cada vez poderia ser usado, em vez dos 3 "sockets" TCP descritos, apenas um sendo feitas ligações às UPs com as quais o processo desejasse comunicar, nas alturas devidas. Sobre o "socket" F, do tipo UDP e não orientado às ligações, são enviados os sinais "order" também por funções próprias dos "sockets". Também aqui as comunicações são encapsuladas em RPCs ("send_active_pns()", do capítulo 5) que, dadas as características de difusão do canal, são no entanto implementadas de outra forma. A função envia um único sinal, que será difundido para todas as UPs. Estas, que deverão averiguar se estão activas,

executam as ordens apenas em caso afirmativo. Outras RPCs usando este "socket", ou seja esta característica de difusão, e não apresentadas na descrição formal são a "prepare_for_sync()", que informa as UPs da existência próxima de uma acção de sincronização, "sync()", que sincroniza os relógios de todas as placas e "continue()", usada para evitar os erros de temporização nas UPs, descritos atrás. Todas estas funções esperam mensagens de confirmação. O envio de sinais para a Interface humana faz-se pelo "socket" B também através de funções próprias dos "sockets". Para este "socket" não há do ponto de vista do Executor de sessões encapsulamento de comunicações.

Para a implementação da construção "INPUT", o SDL assume que cada instância de processo tem associada uma única fila de mensagens, na qual são depositados, por ordem de chegada, os sinais provenientes de todos as outras instâncias. Atendendo a que, na realidade, os processos têm diversos pontos de interacção com o exterior do ponto de vista da recepção de informação, a implementação de uma acção de "INPUT" deve ser feita, em cada estado, pela escuta das filas de mensagens associadas aos canais de onde são esperados os sinais que darão origem à transição. Assim, a implementação do "INPUT" neste processo resume-se, em cada estado, à escuta dos "sockets" adequados. Todas as funções de comunicação com as UPs obrigam a que a função RPC só retorne quando tiver sido recebida a correspondente resposta da UP ou, por ausência desta, por erro de temporização. Isto significa que apenas o "socket" pelo qual foi enviado o sinal de ordem deverá ser escutado para a recepção da resposta. Nos estados do processo, que não os pertencentes às RPCs, deverão ser inspecionados o "socket" B e, se necessário, a fila de mensagens.

Relativamente a outras questões não descritas na especificação SDL para o Executor de sessões, é de salientar a forma de armazenamento de informação consumida pelo processo e as facilidades que deverão ser oferecidas por este. No que diz respeito à primeira devem ser criados vários ficheiros contendo informação que varie de forma lenta no tempo e com tipos de informação bem identificada. Ficheiro de configuração física do sistema contendo por exemplo o número de UPs, o número de placas em cada UP e endereços, ficheiro de mensagens contendo o formato e os valores dos campos constantes das mensagens trocadas com a rede, e ficheiro de erros associando o número do erro ao seu significado são alguns exemplos de ficheiros que poderão existir. O programa deve ainda oferecer a possibilidade de funcionar sem Interface humana, devendo para tal o "socket" B e a fila de mensagens ser substituídos pelos mecanismos normais de entrada e saida da WS, normalmente associados

ao monitor e ao teclado. Todos os ficheiros, incluindo os de sessão e resultados de sessão, devem ser manipuláveis por editores de texto.

7.4. UNIDADE DE PROCESSAMENTO

Duas das placas da UP são a placa mãe, mestre VME, baseada num 68030 com 1Mbyte de RAM e a placa que implementa a interface com a Ethernet, baseada no controlador Am7990. A ligação do processador ao controlador é feita por uma RAM "dualport" existente na segunda placa, ligada por um dos portos ao barramento local da placa mãe e pelo outro ao controlador. Sobre este conjunto corre o OS9 [37], que é um sistema operativo parecido com o Unix, multitarefa e de características de tempo real. A sua arquitectura modular permite a adição de novos módulos de software como por exemplo o pacote de software de comunicações para Ethernet. Uma das UPs, equipada com um disco duro é usada como sistema de desenvolvimento de software, que deverá ser feito em C e "Assembly" 68000. Bibliotecas de funções, tradutores e compiladores e auxiliares de descoberta de erros estão disponíveis.

Interligadas com a placa mãe pelo VME encontram-se as duas placas dedicadas cujos requisitos foram descritos no capítulo 6. Devido a características próprias do OS9, estas placas são encaradas como dispositivos de entrada e saída de dados, significando isto que, por exemplo na comunicação entre as tarefas que correm no 68030 e as que correm no C25, não podem ser usados mecanismos de comunicação entre processos, mas apenas mecanismos para entrada e saída de dados. Este facto é relevante pelas suas consequências na definição da estratégia da implementação do software na UP . A primeira é a definição de um conjunto de funções que implementem o canal de comunicação do OS9 com as placas, normalmente conhecido por "device driver". A segunda é a necessidade de reestruturar a descrição SDL, definindo a placa dedicada C25 como um ambiente próprio, ou seja, um bloco que comunica com o OS9 através de um canal que é o "device driver". Apesar de esta última poder ser dispensável, a tradução da descrição SDL para a implementação torna-se mais sistemática se a alteração for feita.

Um "device driver" em OS9 [38] é, de uma forma simples, um conjunto de primitivas que permite o acesso a recursos exteriores ao computador, ou seja, que implementa um canal de comunicações próprio de um determinado recurso ou dispositivo. Estes dispositivos

actuam normalmente como servidores, atendendo pedidos de processos que correm no processador sobre o qual assenta o sistema. As primitivas, em número de 5, são abre_dispositivo() e fecha_dispositivo(), que inicializam e terminam o dispositivo, lê() e escreve(), que permitem a leitura e escrita de uma mensagem no dispositivo e lê_estado() e escreve_estado(), que permitem a leitura e a alteração do estado do dispositivo. Assumindo como exemplo um controlador de porta série como o dispositivo a manipular, este será inicializado através de um abre_dispositivo(), um caracter poderá ser lido da porta através da função lê() e enviado para a mesma porta pela função escreve(). Alterações ao registo de controlo do controlador podem ser feitas através de escreve_estado() sendo o seu uso terminado através da função fecha_dispositivo(). Associado a um "device_driver" existe ainda a rotina que faz o tratamento da interrupção gerada pelo dispositivo, e que é activada ou instalada com a função abre_dispositivo(). Imagine-se que, associada a uma porta série, existe existe uma fila de caracteres a enviar e que um determinado processo chama a função escreve(). Como o caracter não será enviado imediatamente e o retorno de escreve() garante o sucesso da operação, o processo adormece, passando a um estado de espera. A rotina de interrupção associada ao "driver", que é chamada sempre que um caracter é transmitido, deverá entre outras coisas acordar o processo, quando o seu caracter tiver sido transmitido, pelo envio de um sinal "acorda".

A alteração da estrutura da especificação SDL, feita no sentido de tornar a especificação formal mais próxima da implementação, consiste em criar dois novos blocos para as placas C25, dando-lhes ambiente próprio e definindo canais de interacção.

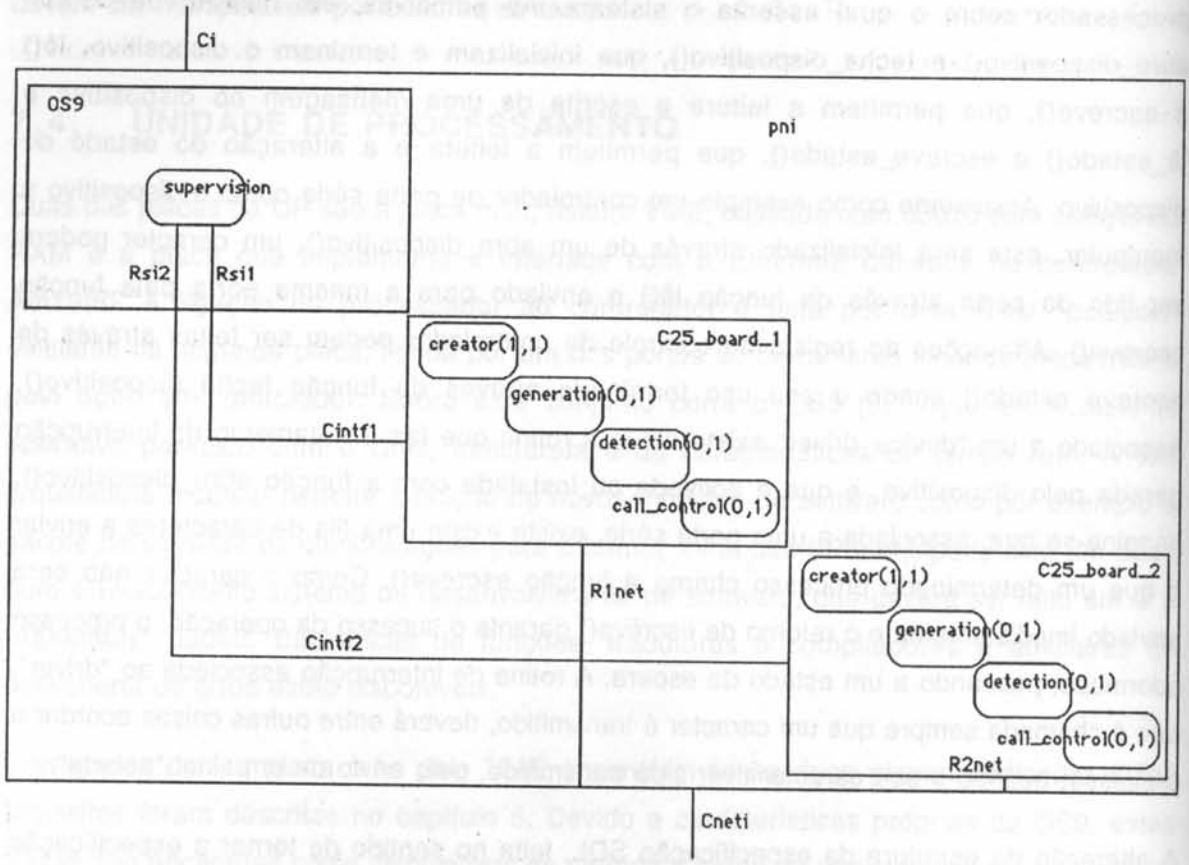


Figura 7.3 - Estrutura adaptada da UP

Como se pode observar na Fig. 7.3, o "pn" fica constituido pelos blocos "OS9", "C25_board_1" e "C25_board_2", interligados pelos canais "cintf1" e "cintf2". As alterações de funcionamento introduzidas são mínimas e bem localizadas. No bloco "OS9", constituido agora só pelo processo "supervision", as alterações surgem apenas nas rotinas de comunicação. A "generate_task()", em vez de criar as tarefas, envia um novo sinal com a descrição do processo a lançar para um processo "creator(1,1)", que criará o processo. Se a criação é bem sucedida, o processo criado envia ao "supervision" o sinal "ok_announc" com a sua identificação, se não, este sinal é enviado pelo "creator" com a identificação do processo criado igual a "NULL". Nas rotinas "send_no_reply()" e "send_w_reply()" haverá apenas que reformular os endereços e as vias escolhidas para a saída dos sinais, o que é muito simples. No bloco "C25_board_i" haverá que, além de introduzir o "creator" que só lança as tarefas, reformular os endereços dos sinais de

resposta. Apenas uma instância dos processos terminais ("generation", "detection" e "call_control") será permitida em cada bloco, sendo as construções "VIEW - REVEAL", usadas para as transferências de dados e resultados, substituídas pelas "IMPORT-EXPORT".

7.4.1. Interfaces

Pelo facto de no novo bloco "OS9" existir apenas um processo, o "supervision", são definidas apenas duas interfaces: "OS9" - "ws" e "OS9" - "C25_board", como se observa na Fig. 7.4.

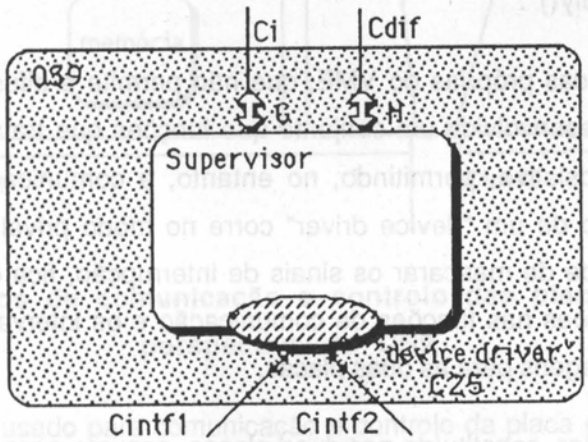


Figura 7.4 - Modos de comunicação no OS9

A interface do "supervision" com a "ws" é implementada por dois "sockets", G e H, o primeiro do tipo TCP e o segundo do tipo UDP, que são usados para a recepção dos sinais enviados, respectivamente, pelos "socket" TCP (C, D ou E) e UDP (F) da "ws" bem como para o envio dos sinais de confirmação. As mensagens tratadas em cada um dos "sockets" G e H encontram-se descritas em 7.3.1.

Para a implementação da interface com as placas dedicadas, o bloco "C25_board_i" da nova estrutura SDL, é necessário implementar um conjunto de funções que permitam a comunicação com a placa e que se enquadrem na filosofia de funcionamento do sistema operativo. As funções descritas na especificação SDL para comunicação com as placas, e

que se mantiveram praticamente inalteradas durante a sua evolução, "send_no_reply()" e "send_w_reply()", caracterizam-se por enviar mensagens para as duas placas, com um intervalo de tempo entre elas nulo do ponto de vista da especificação. Na prática isto significa que as mensagens devem ser enviadas ao mesmo tempo para as duas placas ou, na impossibilidade de difusão, devem ser enviadas com um intervalo de tempo determinístico e de valor desprezável do ponto de vista da aplicação. De facto, quando as mensagens em causa são as que ordenam às tarefas que iniciem e terminem a sua actividade, devem ser recebidas simultaneamente - exigência de atraso desprezável. Quando as mensagens são as usadas para o acerto dos relógios, devem chegar às placas com uma diferença de tempo constante - exigência de determinismo. Da descrição em apêndice, são os sinais "tstart", e "tkilltask", enviadas pela função "send_no_reply()", e os sinais "tinit" e "tstop" pela função "send_w_reply()".

Devido às características próprias do VME, que não permite a difusão de mensagens com destino múltiplo, foi desenvolvido um conjunto que funções que tratam as placas dedicadas como um único disposistivo, permitindo, no entanto, a comunicação com somente uma delas. Como o código de um "device driver" corre no modo priveligiado do 68030, o que significa a possibilidade de mascarar os sinais de interrupção, fica desta forma garantido o determinismo das acções nas funções de comunicação e os intervalos de tempo pequenos entre o envio de mensagens para as duas placas.

O dispositivo a controlar, constituído por duas placas, assume para o OS9 a forma descrita na Fig. 7.5. Cada placa é composta por um registo de estado, um registo de controlo, uma zona de memória para escrita de mensagens (A), uma zona de memória de leitura de mensagens (B), um relógio e memória. O dispositivo gera ainda duas interrupções, uma por placa, quando é recebida uma mensagem na zona de leitura. Na descrição SDL, o sinal escrito na zona A é "msg", que assume siginificados de "msg(tinit)", "msg(tstart)" e "msg(tstop)". Os sinais lidos da zona B são "ok_announc", "ttask_err" e "ok_msg".

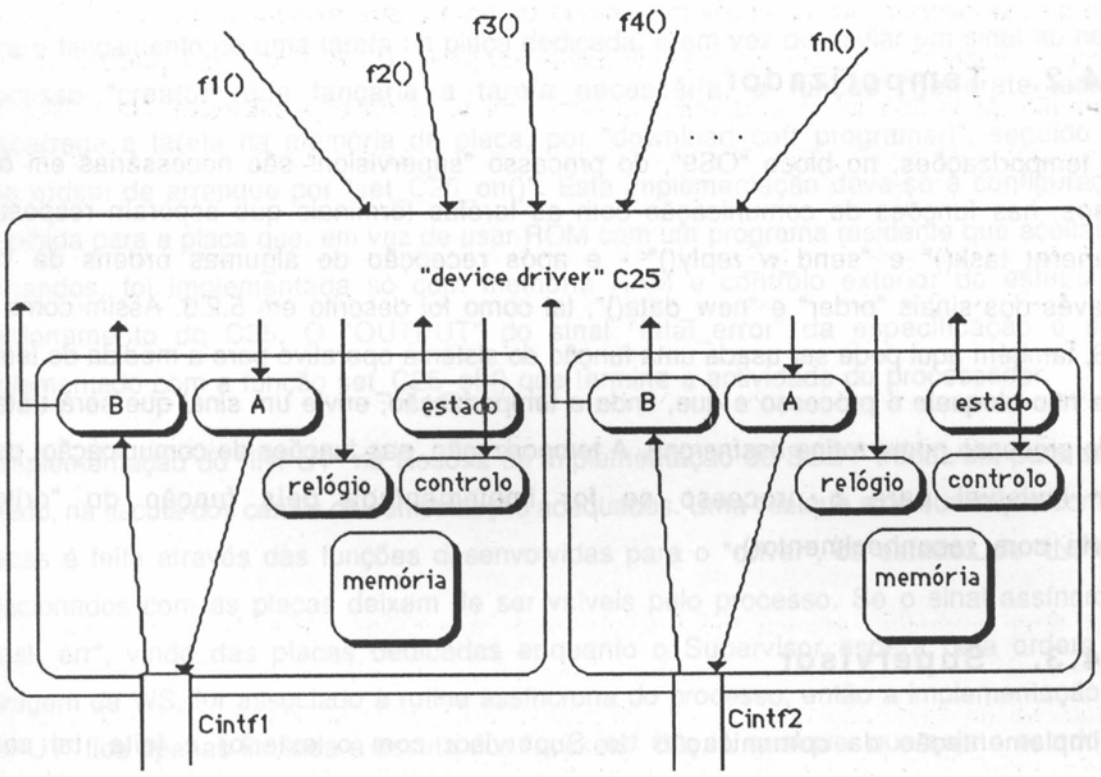


Figura 7.5 - Aspectos de comunicação e controlo das placas dedicadas, pela perspectiva do OS9

O conjunto de funções usado para comunicação e controlo da placa é composto por abre(), envia_com_reconhecimento(), envia(), sincroniza(), lê(), escreve_memoria(), lê_memoria(), lê_estado(), altera_estado() e fecha(). As funções abre() e fecha() são usadas para inicializar e terminar o dispositivo, envia_com_reconhecimento() envia uma mensagem para uma ou duas placas e espera pela(s) mensagem(s) de confirmação, envia() é usada para o envio de uma mensagem para uma ou duas placas sem esperar por confirmação, sincroniza() acerta os relógios escrevendo um valor no primeiro relógio e o mesmo valor adicionado do atraso no segundo relógio e as funções escreve_memoria() e lê_memoria() são usadas para ler e escrever grupos consecutivos de bytes na memória de uma das placas. As funções lê_estado() e altera_estado() servem para verificação e controlo do estado de cada uma das placas dedicadas.

7.4.2. Temporizador

As temporizações, no bloco "OS9", do processo "supervision" são necessárias em dois casos: nas funções de comunicação com as tarefas terminais que esperam resposta - "generat_task()" e "send_w_reply()" - e após recepção de algumas ordens da WS, através dos sinais "order" e "new_data()", tal como foi descrito em 5.2.3. Assim como na WS, também aqui pode ser usada uma função do sistema operativo para a medida de tempo que não bloqueie o processo e que, finda a temporização, envie um sinal que será tratado pelo processo numa rotina assíncrona. A temporização, nas funções de comunicação, pode ser invisível para o processo se for implementada pela função do "driver" envia_com_reconhecimento().

7.4.3. Supervisor

A implementação da comunicação do Supervisor com o exterior é feita, tal como apresentado na Fig. 7.4, através de 2 "sockets" e do "device driver" sendo, por este motivo, a implementação da construção "OUTPUT" feita de acordo com a especificidade de cada um destes canais de comunicação. Pelo "socket" G, do tipo TCP e orientado às ligações, e pelo socket H do tipo UDP são enviadas as mensagens de confirmação "ok_xxx", através de funções próprias dos sockets. A comunicação com as placas dedicadas é feita pelas funções de comunicação do "device_driver" descritas. Estas funções, além de usadas pelas funções especificadas "send_w_reply() e send_no_reply()", deverão também ser usadas por outras não especificadas, tais como "download_coff_programs()" que descarrega um programa em formato COFF na memória de uma placa dedicada, "set_C25_on()" e "set_C25_off()", que activam e desactivam o processador da placa, "synchronise()", que faz o acerto dos relógios, "new_data()", que descarrega os dados de uma nova sessão para a memória das placas dedicadas (implementação do "EXPORT" do SDL"), e "get_results()", usada para ler os resultados da sessão da memória da placa dedicada (implementação do "IMPORT"). Como se observa muitas destas funções têm tradução directa em funções do "driver" pelo facto deste ter sido construído com o objectivo de, em primeira análise, servir esta aplicação.

Para o lançamento de uma tarefa na placa dedicada, e em vez de enviar um sinal ao novo processo "creator" que lançaria a tarefa necessária, a função "generate_task()" descarrega a tarefa na memória da placa, por "download_coff_programs()", seguido de uma ordem de arranque por "set_C25_on()". Esta implementação deve-se à configuração escolhida para a placa que, em vez de usar ROM com um programa residente que aceitasse comandos, foi implementada só com memória RAM e controlo exterior do estado de funcionamento do C25. O "OUTPUT" do sinal "fatal_error" da especificação é aqui implementado com a função set_C25_off() que termina a actividade do processador.

A implementação do "INPUT" na filosofia de implementação do SGDT traduz-se, para cada estado, na escuta dos canais de comunicação adequados. Uma vez que a comunicação com as placas é feita através das funções desenvolvidas para o "driver", os estados de "INPUT" relacionados com as placas deixam de ser vsíveis pelo processo. Se o sinal assíncrono "ttask_err", vindo das placas dedicadas enquanto o Supervisor espera pela ordem de paragem da WS, for associado à rotina assíncrona do processo, então a implementação do "INPUT" fica apenas limitada à escuta dos "sockets" G e H, qualquer que seja o estado do processo. A associação de "ttask_err" à rotina assíncrona do processo é conseguida através do envio de um sinal particular pelo driver, quando se apercebe de "ttask_err" através da sua rotina de interrupção que está associada à detecção de mensagens.

Apesar de não especificado em SDL, o Supervisor deve ainda oferecer uma interface directa ao utilizador que será conseguida através de uma porta série da placa mãe. Do ponto de vista do utilizador isto significa que uma UP poderá ser usada isoladamente como geradora, detectora ou origem de chamadas, perdendo-se, no entanto, o conceito de canais de pacotes e sinalização. A UP passará a ser modelizada tal como é realmente: uma unidade com duas cartas capazes de correrem as tarefas descritas. Se a UP tiver disco deverá ainda permitir a definição das sessões e a gravação dos resultados em ficheiros. Do ponto de vista do programa, esta alteração traduz-se em usar a entrada e saída normal da UP (monitor e teclado) em vez dos "sockets".

7.4.4. Placa C25

O desenvolvimento de programas em "Assembly" ou C, bem como a sua simulação, é feito num PC, sendo estes programas posteriormente transferidos para as placas dedicadas onde são executados. Para que o desenvolvimento de programas para as placas se faça de uma forma metódica, serão criadas funções que permitam uma interacção simples com cada um dos seus dispositivos (interface_VME, interface_rede, relógio e temporizador). Estas funções, reunidas numa biblioteca, poderão ser posteriormente usadas por qualquer programa que, desta forma, passará a observar a placa tal como aparece modelizada na Fig. 6.1.

A interface_VME apresentada na Fig. 7.6 deverá, numa primeira fase, ser implementada sem filas de espera devido ao funcionamento dos processos que irá suportar. Estes processos, porque só atendem um pedido de cada vez, só recebem uma mensagem do OS9 depois de enviada a resposta ao pedido anterior.

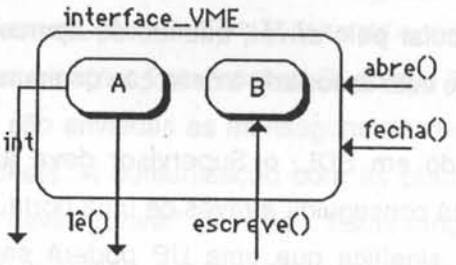


Figura 7.6 - Funções para manipulação da interface_VME

As funções a implementar para manipulação desta interface, ou canal de comunicação com o OS9, são: lê(), escreve(), abre() e fecha(). A função lê() deve ler a mensagem da zona A, com opção de espera, não podendo uma mensagem ser lida mais de que uma vez. Associada a esta função existe uma rotina de interrupção que é executada quando chega uma nova mensagem. A função escreve() deve escrever uma mensagem na zona B. As funções abre() e fecha() servem para, respectivamente, permitir e proibir o uso das funções anteriores.

A interface_rede, usada para transmitir pacotes para a rede, isto é, para implementar o canal de comunicação com a rede, será implementada com duas filas de espera, uma em cada sentido (Fig. 7.7). O dimensionamento destas filas bem como a sua natureza (hardware ou software) deverá ser motivo de estudo e, com certeza, variar de acordo com as exigências de cada uma das tarefas de geração, detecção e origem de chamadas.

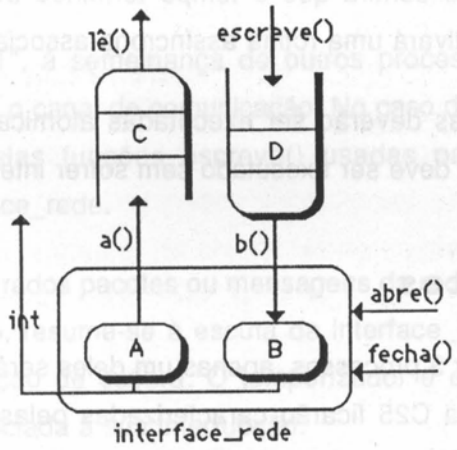


Figura 7.7 - Funções para manipulação da interface_rede

As funções a implementar são as mesmas que para a interface_VME. Desta vez, a função lê() deve ler a primeira mensagem, se existir, da fila C. Se não existir deve esperar pela mensagem, caso tenha sido chamada com opção de espera, ou retornar erro no caso contrário. A função escreve() deve escrever a mensagem na fila D ou, se tanto esta e como a zona B estiverem vazias, na zona B com a correspondente ordem de transmissão. Caso a fila se apresente cheia, deve retornar erro. Ambas as funções têm uma rotina de interrupção associada que é usada, no primeiro caso, quando um pacote é recebido e, no segundo caso, quando um pacote é transmitido. Quando um pacote é recebido esta rotina (a() na Fig 7.7) transfere o pacote da zona A para a fila C. Quando é transmitido um pacote, a rotina de interrupção correspondente (b() na Fig. 7.7) passa o primeiro pacote da fila D para a posição B e dá ordem de transmissão ou, caso a fila esteja vazia, armazena informação de zona B livre. As funções abre() e fecha() servem para iniciar e terminar o uso da interface.

Uma vez que é controlado pelo OS9 por razões de sincronismo, o relógio é acedido apenas através de uma única função, lê(), que deve ler o seu valor no momento da chamada.

O temporizador, à semelhança da construção "TIMER" do SDL, será acedido pelas funções activa() e desactiva(). A primeira introduz o intervalo de tempo de espera no temporizador e activa o seu funcionamento. A segunda desactiva o temporizador. O programa a correr no C25 sentirá que o tempo terminou através da interrupção gerada pelo temporizador, que activará uma rotina assíncrona associada ao programa.

Todas as funções descritas deverão ser executadas atomicamente, isto é, o conjunto de instruções de cada função deve ser executado sem sofrer interrupções.

7.4.4.1. Interfaces

Apesar de a placa suportar 3 processos, apenas um deles será executado de cada vez, pelo que as interfaces na placa C25 ficarão caracterizadas pelas interfaces destes processos com o exterior, isto é, o OS9 e a rede.

A interface entre a placa dedicada e o OS9 é implementada pelo conjunto de funções definidas para a interface_VME no ponto anterior. As mensagens transmitidas no sentido ascendente (placa dedicada - OS9) são "ok_announc", "ok_msg" e "ttask_err" e, no sentido contrário, "msg".

A outra interface, implementada pelas funções definidas para a interface_rede, suporta no sentido placa - rede as mensagens "tgds_announc" e "tgds_packet" e, no sentido contrário, "tgds_packet", não estando incluídas as mensagens da tarefa origem de chamadas.

7.4.4.2. Temporizador

Das tarefas de geração e detecção de pacotes, apenas a primeira usa temporizações e apenas para um caso: determinação do instante de lançamento do próximo pacote para a rede.

A implementação é conseguida pelo uso do temporizador da placa, que foi projectado especificamente para este fim. As funções usadas são as definidas em 7.4.4 para o temporizador - activa() e desactiva(). Quando a temporização expira, o programa a correr no C25 - a geração de pacotes - é avisado através de uma interrupção.

7.4.4.3. Gerador

A construção SDL "OUTPUT", à semelhança de outros processos nos outros blocos, é implementada de acordo com o canal de comunicação. No caso desta tarefa, a construção é directamente substituída pelas funções escreve() usadas para transmitir sinais pela interface_VME e pela interface_rede.

Uma vez que nunca são esperados pacotes ou mensagens da rede, a acção de "INPUT", em qualquer estado do processo, resume-se à escuta da interface_VME pela função lê(), que poderá ser usada com a opção de espera. O temporizador é escutado através da rotina assíncrona do programa associada à sua interrupção.

Esta rotina de interrupção, no caso da tarefa de geração, é usada como realimentador contínuo do temporizador. As acções associadas a esta rotina, cuja ordem pode ser alterada, são: calcular o comprimento do pacote a gerar, determinar o instante actual (que será enviado no pacote), actualizar contagens e histogramas, enviar o pacote para a rede, calcular o próximo intervalo entre pacotes e enviar este novo intervalo para o temporizador. As medidas efectuadas são a contagem do número de pacotes gerados durante a sessão e os histogramas de intervalo entre pacotes consecutivos e de comprimento entre pacotes. O tempo que é transmitido no pacote e usado para o cálculo do primeiro histograma é o instante em que o pacote é enviado para a fila da interface_rede e não o instante de transmissão. De facto, a fila de pacotes a transmitir, e representada na Fig. 7.7 como a fila D, implementa a fila de espera de acesso ao meio (rede), cujo comprimento em cada instante é função da relação tráfego gerado/carga na rede. A capacidade desta fila, que deve ser dimensionada de modo a não serem perdidos pacotes, pode ser consideravelmente aumentada se a interface_rede for alterada da seguinte forma: uma vez que os campos variáveis dos pacotes são o seu comprimento e o instante de geração, implementa-se uma fila em que cada pacote é representado apenas por estes dois campos, bastando para isso

que no inicio da sessão o formato do pacote seja dado a conhecer à interface. Deste modo consegue-se aumentar o número máximo de pacotes na fila, qualquer que seja o seu comprimento, e diminuir os tempos de transferência dos pacotes da fila D para a zona B (Fig. 7.7).

Também com o objectivo de diminuir os tempos de processamento, as funções de distribuição características de cada um dos seis parâmetros do modelo de geração de dois estados (intervalo entre pacotes no estado 1, comprimento de pacotes no estado 1, duração do estado 1, intervalo entre pacotes no estado 2, comprimento de pacotes no estado 2 e duração do estado 2) podem ser tabeladas em vez de calculadas. Desta forma a determinação do valor de um parâmetro, por exemplo da duração do próximo estado, é feito pelo cálculo de um número aleatório com distribuição uniforme, que será usado para endereçar a tabela desejada.

7.4.4.4. Detector

A construção "OUTPUT" é implementada, tal como na geração, pelas funções escreve(), das duas interfaces. As acções de "INPUT" da especificação são implementadas pela escuta das interface_VME e interface_rede através das funções lê(), sendo a escuta da interface_rede feita sem espera de mensagem e a da interface_VME feita com espera em estados onde não exista actividade de detecção.

As contagens efectuadas pelo detector durante uma sessão, são o número de pacotes detectados, o número de pacotes com erros e número de pacotes incompletos, sendo a informação relativa às duas últimas fornecida pela interface_rede. Os histogramas efectuados são o de atrasos extremo a extremo e o de intervalos entre pacotes consecutivos. Quando um pacote é detectado, deve ser-lhe anexado o tempo de chegada pela rotina de interrupção a() que o transfere da zona A para a fila C (Fig. 7.7). O atraso do pacote será a diferença entre o instante de chegada e o instante de geração, transmitido no pacote. Este atraso é soma do atraso de espera na fila da geração, com o atraso de acesso à rede e com o atraso de propagação. No fim da sessão este histograma contem informação sobre a distribuição dos atrasos. Os atrasos entre pacotes são calculados pela diferença dos tempos de chegada de dois pacotes consecutivos. A comparação deste histograma com o histograma equivalente obtido na geração permite caracterizar a variabilidade dos atrasos ou "jitter"

introduzido pela rede. O número de pacotes perdidos, a calcular na Interface humana, é determinado pela diferença entre os gerados e detectados.

Da mesma forma que na geração, também aqui o número de pacotes na fila de pacotes recebidos (fila C) pode ser aumentado se, em vez de armazenados os pacotes completos, forem apenas armazenados o campo de tempo de geração, juntamente com o tempo de chegada e a informação de erro ou pacote incompleto.

7.4.4.5. Origem de chamadas

Pelas razões descritas em 5.2.1 o projecto deste processo não foi completamente executado, tendo-se procurado sobretudo o seu enquadramento na filosofia de funcionamento do SGDT. Se a origem de chamadas for implementável apenas com um processo então a sua integração será trivial, à semelhança das outras duas tarefas. Se necessitar de dois processos, um para início e terminação de chamadas e outro para estabelecimento do protocolo de comunicações com o sistema operativo distribuído da rede, então a estratégia a usar deverá ser um pouco mais elaborada. Uma vez que a placa não tem uma plataforma de software multitarefa, em vez de dois processos implementam-se dois serviços (construção SDL "SERVICE") num único processo, que é a forma usada em SDL para a descrição do pseudo paralelismo assumido.

Em SDL, todos sinais que tenham como destino este processo, quer provenham do exterior quer dos próprios serviços, são encaminhados para a mesma fila. Como deve ser garantido que os sinais pertençam apenas a um serviço, em cada momento é executado o serviço ao qual a próxima mensagem da fila se destine.

Na estratégia de implementação adoptada, o processo não tem uma fila mas várias, tantas quantas os canais de entrada. Assim seriam também implementados os serviços: cada um deles, além dos canais próprios com o exterior, deveria ainda ter associada uma fila onde seriam depositados os sinais lhe fossem dirigidos pelo outro serviço. A decisão do próximo serviço a executar seria feito de acordo com um critério razoável para o bom funcionamento do programa, sendo o mais simples o de execução alternada dos dois serviços.

8. CONCLUSÕES

8.1. ACTIVIDADE

O trabalho apresentado nesta tese foi o resultado de 3 longos anos de actividade dedicada a estudos que directa ou indirectamente influenciaram o SGDT.

O primeiro trabalho consistiu na definição de cenários de utilização de redes de banda larga com integração de serviços, do ponto de vista do tráfego transportado. Para tal foram identificados os potenciais serviços de telecomunicações e caracterizados pelo tipo de informação transmitida e requisitos de comunicação. No âmbito do projecto RACE 1011 (BCPN) foi estudada a penetração destes serviços em ambientes empresariais, sendo cada um caracterizado sob o ponto de vista de duração média das chamadas e taxa média de chegada de chamadas na hora de ponta. Para obter os requisitos de comutação destes ambientes, o tráfego gerado por cada um dos serviços foi medolizado, tendo sido usado o modelo de dois estados. Na fase seguinte, para cada serviço identificaram-se os seus requisitos de transporte através da quantificação dos parâmetros de qualidade de serviço considerados mais importantes. Uma cópia do relatório final deste trabalho, realizado pelo autor em coautoria com 2 outros investigadores, é apresentada no apêndice A desta tese. Estes estudos infuenciaram fortemente o SGDT e justificam o modelo de geração adoptado e as medidas associadas ao canal de pacotes.

Um segundo trabalho no qual o autor participou foi a simulação do protocolo de acesso ao centro da estrela na rede UCOL. Este trabalho, também descrito num artigo cuja cópia é apresentada no apêndice B, permitiu uma melhor percepção do protocolo que o SGDT é suposto testar e validar, bem como a identificação do tipo de medidas a efectuar, perfis de tráfego e valores típicos de alguma grandezas, como por exemplo o atraso médio extremo a extremo sofrido pelas células (pacotes no SGDT).

O terceiro e último trabalho, relatado nesta tese, foi o desenvolvimento do SGDT, nas suas fases de especificação e projecto detalhado de software.

Os resultados vísiveis desta actividade são a publicação de 3 artigos apresentados em conferências patrocinadas pelo IEEE, 1 artigo na ENDIEL 91, 7 documentos apresentados no âmbito do projecto RACE - BCPN sobre características de tráfego, 20 documentos produzidos para o UCOL sobre a simulação de protocolo e o SGDT e 8 artigos para o DAMS sobre o SGDT. Foi também feito um pedido de registo de patente, o primeiro apresentado pelo INESC. Do ponto de vista do autor, os resultados foram o aumento dos seus conhecimentos em tráfego, redes com integração de serviços, simulação, métodos de desenvolvimento de sistemas, especificação e representação formal, SDL, sistemas operativos, software de comunicações, comunicações entre processos, sistemas de tempo real, sistemas distribuídos, barramentos normalizados e arquitecturas de computadores.

8.2. SGDT

O SGDT pode ser identificado como um sistema que gera e detecta tráfego, quer de pacotes de dados quer de mensagens de sinalização. Foi desenvolvido para ser usado em duas redes metropolitanas com integração de serviços (UCOL e DAMS), como gerador de cargas variáveis e avaliador de desempenho da rede através de medidas efectuadas sobre o tráfego. O seu principal atributo, que o distingue de outros sistemas do género, é a capacidade de geração e detecção de grandes quantidades de tráfego, controladas estatisticamente pelo utilizador. Isto significa a possibilidade de simular fontes de tráfego, individuais ou multiplexadas e avaliar o efeito da rede sobre as cargas geradas, através da medidas de parâmetros significativos. Pelo facto de poderem ser gerados pacotes de dados e mensagens de sinalização (usadas para o estabelecimento e terminação de chamadas) as várias fases de uma chamada podem ser avaliadas.

As medidas efectuadas são as possíveis de um sistema desta natureza: para os pacotes incluem atrasos na rede, variabilidade desses atrasos ("jitter"), pacotes perdidos e pacotes com erros. Para as mensagens de sinalização são os atrasos de estabelecimento e terminação de chamadas e as taxas de insucesso destas operações. Como valores característicos, o SGDT gera e detecta 50.000 pacotes por segundo por cada tarefa de geração ou detecção, estabelece 500 chamadas por tarefa de origem de chamadas, efectua medidas com uma resolução de 200ns e suporta 6 tarefas simultâneas.

O trabalho descrito nesta tese consistiu na especificação e definição da estratégia de implementação do SGDT. Nela sobressaem várias fases que contribuíram de forma decisiva para o resultado final: identificação dos requistos, especificação do sistema, projecto preliminar em SDL, definição da arquitectura do hardware e projecto detalhado do software.

Começou-se por justificar as razões da existência do Sistema como parte das estratégias de teste e validação adoptadas nas redes UCOL e DAMS. Atendendo à complexidade esperada para a execução do sistema e ao facto de serem esperados resultados numa data bem definida - Junho de 1992, foi adoptado um método para o seu desenvolvimento baseado na divisão do projecto em duas fases principais: planeamento e desenvolvimento.

A fase de planeamento consistiu na identificação dos requisitos do sistema e na elaboração das suas especificações. Os requisitos físicos para o sistema foram 3 unidades de processamento, cada uma destas contendo uma interface física de acesso à rede, interligadas entre si e a uma "workstation" através de uma Ethernet. Os pacotes, gerados por processos estatísticos, têm comprimento variável entre 53 e 2048 octetos e as mensagens de sinalização são as usadas no DAMS. As medidas efectuadas devem ser as possíveis e obtidas no sentido de avaliar o desempenho das redes. Na especificação foram introduzidos conceitos que permitem a simplificação da interacção do utilizador com o sistema: modos de funcionamento, configuração do sistema, canais de pacotes, canais de sinalização, sessão e grupo. Para cada um dos canais foram descritos os modelos de geração, dois estados para pacotes e um estado para sinalização, bem como as medidas efectuadas, descritas anteriormente. A arquitectura básica da UP apresentada consistiu numa placa mãe interligada por um barramento VME a 2 placas C25.

O projecto preliminar do sistema surgiu como o primeiro passo da fase de desenvolvimento. Com base nos requisitos, nos conceitos introduzidos e na arquitectura básica do sistema, o projecto foi apresentado na linguagem formal SDL. O sistema foi dividido em 4 blocos, a "ws" e 3 "pn", sendo o primeiro constituído pelo processo "session_execution" e os "pn" pelos processos "supervision", "generation", "detection" e "call_control". A estratégia adoptada para a detecção e identificação de erros foi também introduzida.

O requisitos da placa C25 foram definidos seguidamente: deve ser um periférico VME não inteligente, baseado no processador TMS320C25 e a cujo barramento local devem ser interligadas as interfaces de rede. Será ainda necessário que a placa suporte um temporizador e um relógio, e toda a sua memória deve ser mapeada no espaço de endereçamento global do VME.

Ao projecto preliminar seguiu-se o projecto detalhado, que consistiu na apresentação de uma estratégia para a implementação da descrição SDL e de assuntos não descritos formalmente, como por exemplo a sincronização dos relógios. Não podendo a tradução de SDL para uma linguagem de programação ser feita de uma forma universal, foram assumidas equivalências base para o sistema, descritas estratégias de implementação para os "Timers" e canais de cada bloco, e para os "Outputs" e "Inputs" de cada processo. Para a sincronização foram apresentadas duas alternativas: uma de hardware e outra de software.

A execução e teste do SGDT, não incluída nesta tese, tem vindo a ser desenvolvida por uma equipa de 6 pessoas. Nesta altura, Dezembro de 1991, encontram-se já executadas e testadas as placas C25 [39], a tarefa de geração [40], o "device driver" para as placas dedicadas no OS9 [41], o supervisor na versão local [41] e as RPCs [42]. Em adiantado estado de execução encontram-se a interface_rede, a tarefa de detecção e a interface humana [43]. Em ínicio de implementação está o executor de sessões e a tarefa origem de chamadas.

Resultados obtidos para a geração apontam para taxas de 59 kpac/s para um perfil genérico dois estados, 77 kpac/s para a função de geração Markoviana, 83 kpac/s para a função Determinística e 125 kpac/s para a AFAP [44].

8.3. FUTUROS DESENVOLVIMENTOS

Pela sua arquitectura modular, topologia distribuída e quantidades de tráfego geradas, o SGDT tem desfrutado de um bom acolhimento nos projectos europeus de investigação ESPRIT, para a geração e detecção de cargas controladas. De facto, estando já em uso no UCOL e DAMS, apresenta-se agora como parte da estratégia de validação de outras duas redes propostas a um novo concurso ESPRIT: o MODECS e o MANTIS. Por este facto existem já algumas ideias relativamente à evolução do SGDT.

É possível, no entanto, perspectivar outros desenvolvimentos que não a evolução natural deste sistema, e nos quais o autor espera vir a participar. De um modo genérico, esses desenvolvimentos podem ser decompostos em 3 vectores fundamentais: evolução do SGDT, validação de redes e SDL no SGDT.

Em termos evolutivos o SGDT deverá a curto prazo integrar o modo de funcionamento de teste, não implementado na versão actual, e assimilar os requisitos a identificar nos novos projectos. A mais longo prazo, impõe-se o desenvolvimento de um núcleo multitarefa para as placas C25, que deverá ser bastante eficiente para não prejudicar as taxas de geração. Desta forma poderão ser integradas novas tarefas orientadas à validação de protocolos de comunicações e outros procedimentos de rede, como por exemplo os relacionados com a gestão. O desenvolvimento de interfaces para outras redes será também uma prioridade.

Em termos de validação de redes também muito pode ainda ser feito no SGDT: trabalho teórico pode ser realizado nesta área de modo a serem definidas sessões ou grupos de sessões que serão vectores de teste, e que podem ser vistos como procedimentos integrados de validação.

O SDL pode ser usado para automatização do processo de desenvolvimento de software nos ambientes do próprio SGDT. A ferramenta de SDL existente no INESC, o Geode, além de edição, identificação de erros e simulação das especificações, traduz também SDL para C em ambiente UNIX. A tradução automática de SDL para C, em OS9 e sobre o núcleo básico multitarefa a desenvolver, não parece de difícil concepção. Se tal fosse conseguido a eficácia do SGDT aumentaria enormemente, pela possibilidade de particularização e personalização das suas tarefas de teste. Numa só linguagem, o SDL, especificando o funcionamento da entidade da rede a testar, poderiam ser simulados os procedimentos de teste, identificadas as potenciais situações de mau funcionamento, refinados os procedimentos de teste e gerado automaticamente o código de teste, que seria posteriormente corrido nas placas C25 para a validação do sistema sob teste.

9. REFERÊNCIAS

[1] - "Digital Telephony", John Bellamy, a Wiley Interscience Publication.

[2] - "PCM and Digital Transmission Systems", Frank F. E. Owen, Texas Instruments.

[3] - "Data and Computer Communications", William Stallings, Macmillan.

[4] - "Computer Networks", Andew S. Tanenbaum, Prentice Hall Editions.

[5] - "FDDI - A tutorial", Floyd E. Ross, IEEE Communications Magazine, May 1986, págs. 10 - 17.

[6] - "Optical Communications Systems", John Gowar, Prentice Hall.

[7] - "Distributed Systems and Computer Networks", Morris Sloman and Jeff kramer, Prentice Hall.

[8] - "Draft Recommendation I.150; B-ISDN ATM Functional Characteristics, TD 28 (ISDN); CCITT SGXVIII; Geneva, 8-19 January 1990.

[9] - "The impact of the ATM Concept on Video Coding", Luc Pinnoo and Bart Voeten, IEEE JSAC, December 1988, págs. 1623 - 1632.

[10] - "Broadband ISDN Switching Capabilities from a Services Perspective", Dan R. Spears, IEEE JSAC, October 1987, págs. 1222 - 1230.

[11] - "The Prelude ATD Experiment: Assessments and Future Prospects", Michel Devault, Jean Yves Cochennec and Michel Sevel, IEEE JSAC, December 1988, págs. 1528 - 1537.

[12] - "Esprit - Synopsis of Projects and Exploratory Actions", September 1990, Directorate General XIII Telecommunications, Information Industries and Innovation, CEE.

[13] - "Traffic Generator/Detector System Concept", E. Carrapatoso, C. Aguiar, M. Ricardo, EFOCLAN 90 - LAN Proceedings of The Seventh Annual European Fibre Optic Communication and Local Area Network Exposition, Munique, F.R.G., 1990, págs. 119-123.

[14] - "A Traffic Generator/Detector System for High Speed Networks", M. Ricardo, E. Carrapatoso, C. Aguiar, Communication Control and Signal Processing -

Proceedings of the 1990 Bilkent International Conference", Vol. 1, págs. 656-662.

[15] - "Sistema Gerador e Detector de Tráfego", E. Carrapatoso, M. Ricardo, P. Assis, A. Almeida, ENDIEL 91, ST3 - Comunicações, Redes, Equipamentos e Serviços", págs. 371-378.

[16] - "UCOL Project: Recent Advances", D. Capolupo e outros, Proceedings of the 7th Esprit Conference.

[17] - "Dynamically Adaptable Multiservice System - DAMS", D. Wybranietz e outros, Proceedings of the 7th Esprit Conference.

[18] - "High Speed Traffic Generator", Ph. Dermine, C. Constantinidis, P. Godelaine, Esprit 73 - Université de Liege, March 30, 1987.

[19] - "Real Time Computer Control: An Introduction", Stuart Bennet, Prentice Hall International.

[20] - "ICCC Interfaces Specification", Alain Dang, Jalal Samain, E. 2146: DAMS, JST 5/21.

[21] - "Recommendation Z.100", Volume X, Blue Book, CCITT, 1988.

[22] - "SDL`91 Evolving Methods", Ove Faergemand, Rick Reed, Proceedings of the 5th SDL Forum, Scotland, October 91.

[23] - "AGE/GEODE: Reference Manuals", Verilog, September 90.

[24] - "B asic SDL - A tutorial", Ove Faergemand, Proceedings of the 5th SDL Forum, Scotland, October 91.

[25] - "Telecommunications Systems Engineering using SDL", R. Saracco, J. Smith, R. Reed, North Holland.

[26] - "Evaluation of the TMS320C25 Performance", C. Aguiar, M. Ricardo, E. Carrapatoso, UCOL, WP10/INE/5.1.

[27] - "TMS320C25 Users`s Guide", Texas Instruments.

[28] - "The VMEbus Specification", Rev. C.1, Motorola.

[29] - "Report on DQAC Protocol Specification", UCOL, Deliverable 6.08, June 19, 1990.

[30] - "Clock Synchronization in Distributed Real Time Systems", Hermann Kopetz, Wilhelm Ochsenreiter, IEEE Transactions on Computers, Aug. 87, págs. 933 - 940.

[31] - "The TGDS Synchronization Problem", M. Ricardo, E. Carrapatoso, C. Aguiar", UCOL, WP10/INE/4.1.

[32] - "File Formats", SunOS Reference Manual, 1990.

[33] - "Internetworking with TCP/IP", Douglas Comer, Prentice Hall.

[34] - "System Calls", SunOS Reference Manual, 1990.

[35] - "A Socket-Based Interprocess Communications Tutorial", Network Programming Guide, Sun, 1990.

[36] - "System Services Overview", Programmer's Overview Utilities & Libraries, Sun 1990.

[37] - "OS9 Advanced System Software: Operating System Manuals", Microware.

[38] - "How to write a driver under OS9", C édric Rochat, GESPAC S. A.

[39] - "Notas de utilização sobre a placa de interface, TGD/LSI, utilizando o protocolo definido para o DAMS II, o Standard Internal Intrface (SII)", P. J. Assis, Doc. Interno INESC, Outubro 1991.

[40] - "Relat ório Final de Est ágio: Software para Geração e Detecção de Pacotes para uma Placa Baseado no DSP TMS320C25", M. E. Pinto Ferreira, FCUP, Julho 91.

[41] - "Funções para interface com duas placas dedicadas sobre OS9", P. Proença, Doc. Interno INESC, Novembro 1991.

[42] - "Software de Comunicações para o TGD: Relatório Final de Projecto", A. Almeida, N. Biltes, FEUP, Dezembro 1990.

[43] - "Interface Homem - M áquina para um Sistema de Geração e Análise de Tráfego em Redes de Comunicação", J. J. Pinto Ferreira, Tese de Mestrado, FEUP, 1990.

[44] - "Report on Experiments", UCOL, Deliverable 10.08, November 29, 1991.

APÊNDICES

APÊNDICE A - PARÂMETROS DE TRÁFEGO

Este apêndice é uma cópia de um relatório final sobre estudos de tráfego produzido pelo autor em coautoria com o Eng. Clemente Aguiar e o Prof. Dr. Eurico Carrapatoso para o projecto RACE 1011 (BCPN).

A.1 DEFINITION OF SERVICES

Broadband services have been classified, from the point of view of the network, by the CCITT [1] . Two main service categories have been identified: interactive services and distribution services. The interactive services are further subdivided in three classes of services: the conversational services, the messaging services and the retrieval services. The distribution services are represented by the classes of distribution services with and without user individual presentation control. A brief description of the definitions of service classes are presented:

Conversational services:

Bidirectional or unidirectional (e.g. video surveillance), real time, user to user or user to host (e.g. data processing) information data transfer. The transfer of information in each direction is controlled by the originator of that information.

e.g. Telephony, Interactive data, Videotelephony, etc.

Messaging services:

User to user communication via storage units with store and forward mechanism, mailbox or message handling systems.

e.g. Electronic mailbox.

Retrieval services:

Unidirectional information transfer from one or more users to one user. The transfer of information is controlled by the receiver.

e.g. Video retrieval, Image retrieval, Document Retrieval.

Distribution services:

Unidirectional information transfer from user to other users. The transfer of information is controlled by the originator.

e.g. Video distribution, Audio distribution.

For traffic evaluation and switching capacity dimensioning, it was assumed the more demanding ones in terms of required bandwidth, from the 26 possible services presented

in [2]: Telephony, Videotelephony Substandard Quality, Videotelephony Standard Quality, Interactive Data, CAD/CAM, Image Document Retrieval, Video Retrieval Standard Quality, Broadband Video Distribution Standard Quality, whose attributes are presented in the following section:

Telephony

- Service class : Conversational service.
- Type of Information : Sound.
- Applications : Communication for the transfer of voice (sound) between two locations (person-to-person).
- Establishment of commun. : On demand.
- Communication configuration : Point to point.
- Symmetry : Bidirectional symmetric.

Videotelephony

- Service class : Conversational service.
- Type of Information : Moving pictures and sound.
- Applications : Communication for the transfer of voice (sound), moving pictures, and video scanned still images and documents between two locations (person-to-person).
- Establishment of commun. : On demand.
- Communication configuration : Point to point.
- Symmetry : Bidirectional symmetric.

Interactive Data

- Service class : Conversational service.
- Type of Information : Data.
- Applications : .
- Establishment of commun. : On demand.
- Communication configuration : Point to point.
- Symmetry : Bidirectional symmetric.

CAD/CAM

- Service class : Conversational service.
- Type of Information : Data.
- Applications : .
- Establishment of commun. : On demand.
- Communication configuration : Point to point.
- Symmetry : Bidirectional asymmetric.

Video Retrieval

- Service class :	Retrieval service.
- Type of Information :	Moving pictures and sound.
- Applications :	.
- Establishment of commun. :	On demand.
- Communication configuration :	Point to point.
- Symmetry :	Bidirectional asymmetric.

Image Document Retrieval

- Service class :	Retrieval service.
- Type of Information :	Still images.
- Applications :	Professional image communications, medical image database.
- Establishment of commun. :	On demand.
- Communication configuration :	Point to point.
- Symmetry :	Bidirectional asymmetric.

Broadband Video Distribution

- Service class :	Distribution service without user individual presentation control.
- Type of Information :	Moving pictures and sound.
- Applications :	Distribution of video/audio signals.
- Establishment of commun. :	Permanent.
- Communication configuration :	Broadcast.
- Symmetry :	Unidirectional.

A.2 NUMBER OF USERS AND SERVICE PENETRATION

For traffic evaluation and switching capacity dimensioning for a large BCPN it was assumed 2000 of users.

For person to person communication services it was assumed that they would account for a total of 5 calls per user in the busy hour: 1 for Telephony, 2 for Videotelephony Sub-standard Quality and 2 for Videotelephony Standard Quality.

The values for the Call Holding Time, Busy Hour Call Rate and number of users (sources) for each service are those assumed in [3] and are presented in table 1.

	C.H.T	B.H.C.R.	Sources
Telephony		1.0	2000
Videotelephony Subst. Qual.	150	2.0	2000
Videotelephony St. Qual.	150	2.0	2000
Interactive Data	180	1.0	2000
CAD/CAM	300	0.05	2000
Video Retrieval St. Qual.	300	0.05	2000
Broadband Video Dist. St. Qual.	3600	1.0	2
Image Document Retrieval	90	0.1	2000

Table 1 - Service Penetration

A.3 TRAFFIC VALUES

The service parameters used for traffic modelling are, together with those of Table 1, the average and peak bandwidth for the D.B.O. services and the typical (average=peak) bandwidth for the C.B.O. services. For a better understanding of these parameters values presented in Table 2, we describe in the following section the adopted service model.

The exponential distribution is often used in many applications, but considered inaccurate for most types of variable bit rate traffic, which generate significant peaks sustained for long periods (in the order of seconds), as in video or data services. Therefore we have adopted a two-state model described in Fig. 1: Fig. 1.a is typical of a Variable Bit Rate service, such as video, while Fig. 1.b is a better description of a bursty source with alternating activity and inactivity periods, as in CAD/CAM, interactive data and voice.

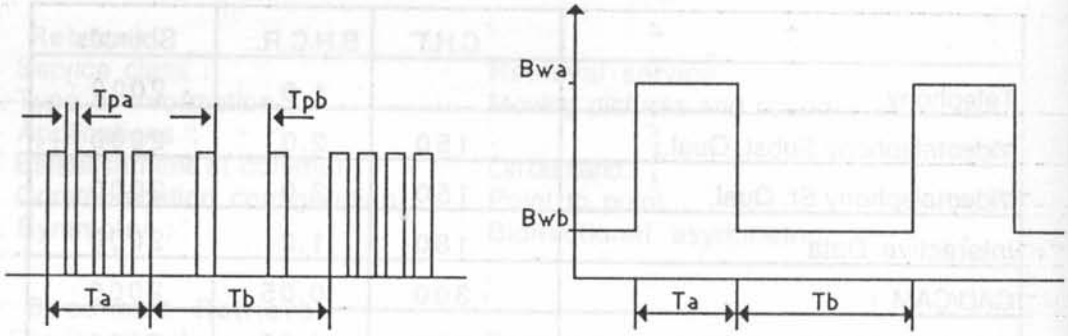


Figure 1.a - Two-State Model - Video D.B.O Services

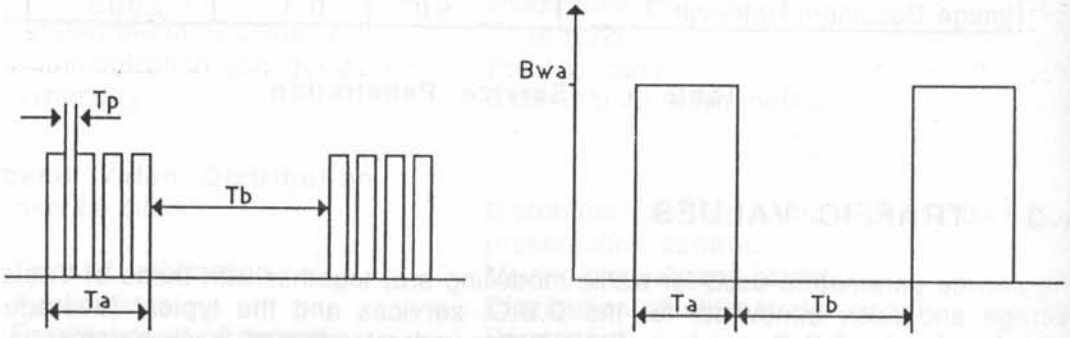


Figure 1.b - Two-State Model - Other D.B.O Services

Referring to Fig.1 the D.B.O. traffic is characterized as follows:

- the burst time, Ta, and the inter-burst time, Tb, are exponentially distributed;
- the inter-packet times, Tpa and Tpb in Fig. 1.a and Tp in Fig. 1.b, are constant;
- the peak bandwidth is BWa;
- the average bandwidth is Av = (Ta * BWa + Tb * BWb)/(Ta + Tb).

Another characteristic taken into account in the model was the service symmetry. For symmetric services the sources are grouped in pairs, which means that two (unidirectional) virtual channels per call are activated simultaneously. For asymmetric services, the bandwidth required in one of the directions was taken as negligible, so only one virtual channel was considered as for unidirectional services.

The adopted values are shown in Table 2.

	Type	Av (kbit/s)	BWa (kbit/s)	Ta (s)	BWb (kbit/s)	Tb (s)
Telephony	D	12	24	1.3	0	1.3
Videotel_Sub	C	2000	-	-	-	-
Videotel_St	D	10000	30000	1.0	9428	35.0
Interactive_Data	D	100	1000	1.0	0	9.0
Image_Doc_Retri	d	500	10000	0.3	0	5.7
CAD/CAM	d	4000	34000	1.0	0	7.5
Video_Retri_St	d	10000	30000	1.0	9428	35.0
Brd_Vid_Dis_St	u	10000	30000	1.0	9428	35.0

C - C.B.O. (bidirectional symmetric)
D - D.B.O. (bidirectional symmetric)
d - D.B.O. (bidirectional asymmetric)
u - D.B.O. (unidirectional)

Table 2 - Parameters for the Two State Model

A.4 QUALITY OF SERVICES

Referring to Quality of Service (QOS) and Network Performance clear distinctions are necessary.

1. User Oriented Quality of Service is the acceptable Quality of Service from a user's point of view. Attributtes such as intelligibility of voice, TV quality and other subjectives characteristics are meant by.

2. Network Oriented Quality of Service is the Quality of a Service necessary to fulfill, with a defined terminal, the acceptable user oriented QOS. Relevant attributes are Bit Error Ratio, Cell Error Ratio and others.

3. Network Performance of a connection type is the performance guaranted by the connection type (throughput, delay, cell loss probability, etc).

The Network Oriented Q.O.S. is the result of the effort of the network and the terminal. For the partitioning of the tasks between network and terminal there exists a certain degree of freedom, dependent on the Q.O.S. subattribute and the service. This is particularly true for new services and new networks such as ATM networks.

In the following are presented the Q.O.S. subattributes between two S/T interfaces of an ATM network.

The set of subattributes generally assumed for an ATM network from a network oriented point of view are:

1. End-to-End Transfer Delay (EED)
2. Delay Jitter (DJ)
3. Cell Loss Ratio (CLR)
4. Cell Insertion Ratio (CIR)
5. Bit Error Ratio (BER)

1. End-to-End Transfer Delay is due to 4 main factors:
    1. Propagation Delay
    2. Queueing Delay
    3. Packetization Delay
    4. Depacketization Delay

An ATM network must guarantee a delay not superior to that already avaiable in existent networks, for each service.

2. Delay Jitter.
This delay is due to the variability of the queueing delay, since the variability in propagation delay is certainly not relevant during a call.

The delay jitter is not critical if enough buffer space in the receiving terminal can be provided. This buffering adds to the overall delay time experienced by a cell and a buffer overflow/underflow will increase the cell slip. As long as these boundaries are not exceeded delay jitter is not critical.

3. Cell Loss Ratio
It is directily related to:
    1. Blocking of cells
    2. Errors in cell header

Blocking of cells is mainly caused by overflow in network buffers. Non detected errors in headers may cause misrouting of cells.

4. Cell Insertion Ratio
This parameter is related to misrouting of cells due to non detected errors in cell headers.

5. Bit Error Rate
It is due to errors in information field in the cell and depends mainly on the transmission links.

Cell Loss Ratio and Cell Insertion Ratio are ATM specific attributes.

The values generally assumed for these attributes for the relevant BCPN services discussed above are listed in the following table.

Services	EED	DJ	CLR	CIR	BER
Telephony	200ms	20ms	10^{-3}	10^{-3}	10^{-6}
Vid.Tel.Ss.Q.	200ms	20ms	10^{-8}	10^{-8}	10^{-6}
Vid.Tel.St.Q.	200ms	20ms	10^{-8}	10^{-8}	10^{-6}
Inter. Data			10^{-6}	10^{-6}	10^{-6}
Cad/Cam			10^{-7}	10^{-7}	10^{-8}
Vid.Retr.S.Q.			10^{-8}	10^{-8}	10^{-6}
Br.Vd.Dt.S.Q.	200ms	20ms	10^{-8}	10^{-8}	10^{-6}
Im.Doc.Retr.			10^{-6}	10^{-6}	10^{-6}

A more precise characterization seems still necessary and the values in the table should be taken as a preliminary guidance.

References

[1] - Recommendation I.121 on broadband aspects of ISDN.

[2] - "CPN Requirements Specification", Deliverable of A1.1, Project R1011 (BCPN), Sept. 1988.

[3] - C. Aguiar, M. Ricardo, E. Carrapatoso - "Traffic Mix Simulation - An Enhanced Model", R1011 A1.2/INE/009.2, 22 March 1989.

APÊNDICE B - ARTIGOS PUBLICADOS

Cópia do artigo apresentado na Bilcon 90, uma conferência patrocionada pelo IEEE, em Ankara - Turquia em 3 de Julho de 1990 e publicado em "Communication Control and Signal Processing - Proceedings of the 1990 Bilkent International Conference", Vol. 1, págs. 656-662.

A TRAFFIC GENERATOR/DETECTOR SYSTEM FOR HIGH SPEED NETWORKS

M. RICARDO[1], E. CARRAPATOSO[1,2] and C. AGUIAR[1]

[1]INESC, Largo de Mompilher 22, 4000 Porto, Portugal

[2]FEUP, Rua dos Bragas, 4099 Porto Codex, Portugal

ABSTRACT

The architecture of a flexible and universal tool, capable of generating and detecting traffic, to test and evaluate the performance of high speed networks, is presented.

INTRODUCTION

The growing tendency to internetwork computers and the fact that present and next generation computers need large transmission resources have lead to the development of high speed networks both for local and metropolitan areas (LANs and MANs). Also, due to, on one hand, the high bit rates offered by the optical fibres and, on the other hand, to recent developments in circuit integration and know-how in distributed systems, multiple telecommunications services may be integrated over the same network.

Those networks could be described as a set of concurrent processing units, distributed over the network access points; they are highly complex systems, whose development raises new problems which, for their solution, require new tools. The system presented in this paper, the TRAFFIC GENERATOR/DETECTOR SYSTEM (**TGDS**) is one of those tools, whose need was identified in two ESPRIT projects: UCOL and DAMS [1], [2].

The TGDS is capable of evaluating network performance and testing some specific control entities, through the generation, detection and analysis of traffic. The traffic used for performance evaluation may consist of packets, whose information field is irrelevant, or packets that simulate control messages, such as those found in ISDN, whereas, for testing, the packet contents may be dictated by the subsystem under test.

To characterise the network performance, the TGDS generates packet traffic, statistically defined by the probability distribution functions (PDF) of interpacket time and packet length and detects the generated traffic to evaluate delays and packet loss probability [3]. It may also be used as network terminal equipments (TE) to generate signalling messages emulating both the originator TE and destination TE [4]. Bursts of calls may be generated

and defined statistically by intercalls time and calls duration parameters. Set up delays and release delays can be evaluated.

As a testing device, the TGDS can be used to send and receive specific messages to network control entities, in order to, through this dialogue, test those entities.

The TGDS working principle and some basic concepts are presented in the **OVERALL DESCRIPTION** section, followed by the **SYSTEM ARCHITECTURE:** hardware and software.

OVERALL DESCRIPTION

The TGDS is a real time distributed system. Its components are a Workstation, where the user can define experiences to be carried out and analyse results, a variable number of Processing Nodes (PN), each capable of both generating and detecting traffic, and an Ethernet, to interconnect them. In Fig.1, a TGDS with three PNs can be observed.

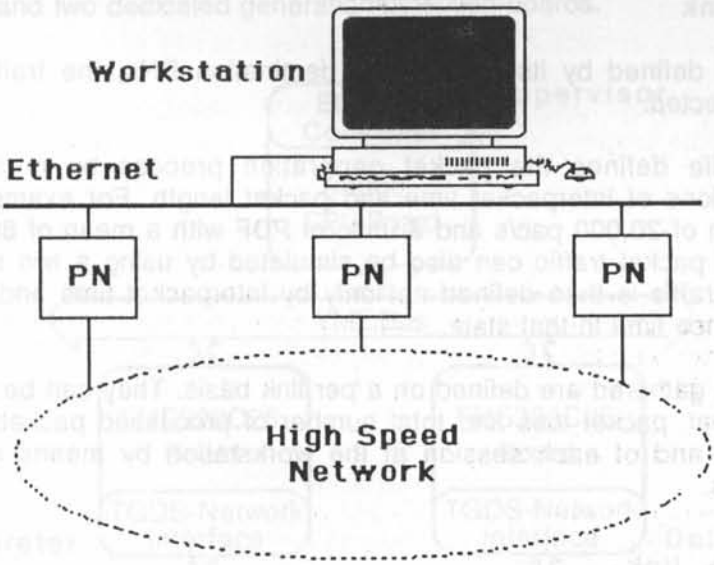


Fig.1. Traffic Generator Detector System

Each PN can generate and detect up to 80,000 pack/s and, when emulating terminal equipments for signalling messages, each PN can be the originator or the destination of 500 simultaneous calls. Measurements have a precision of, at least, 10µs.

Working principle

The TGDS has two operation modes: performance evaluation and test.

In the performance evaluation mode the TGDS is used as replacement for actual traffic sources; experiments, called sessions, are defined, executed and only after their execution may results be visualised in the workstation.

When operating in the test mode, and if traffic similar to the one used for performance evaluation is generated, parameters used to define the traffic may be changed during a session execution and results can be observed at the workstation in real time.

Basic concepts

In order to give some coherence to the various operating modes two concepts were introduced: **session** and **link.**

A **session** may be defined as the basic experiment unit and a **link** is a traffic flow defined, in one session, from one PN to the same or another PN. A link may carry either a packet traffic or a signalling traffic.

Each session supports as many links as the number of PNs.

Packet link

A packet link is defined by its source and destination PNs, the traffic profile and the results to be collected.

The traffic profile defines the packet generation process by means of probability distribution functions of interpacket time and packet length. For example an exponential PDF with a mean of 20,000 pac/s and a uniform PDF with a mean of 80 octets for packet length. Bursts of packet traffic can also be simulated by using a two state traffic model. For each state, traffic is then defined not only by interpacket time and packet length but also by permanence time in that state.

The results to be gathered are defined on a per link basis. They can be: end to end delay, interpacket interval, packet loss and total number of processed packets. Results may be displayed at the end of each session at the workstation by means of histograms and numerical values.

Signalling link

A signalling link is again defined by its source and destination PNs, traffic profile and results. The strategy for signalling traffic is that each PN may be the caller (originator TE) for simultaneous calls. The called PN (destination TE) may be any PN, that is, during a session, from one source PN of a link, any PN may be called.

The traffic profile defines the call arrival processes by means of the PDFs of intercall times, call duration and dial numbers. The TGDS is able to generate a subset of Q.931 CCITT recommendation [5] signalling messages, which are exchanged between the PNs and the network call control entities. Results obtained in a signalling link are variable and depend on the signalling protocol implemented, the objective being to statistically measure set up and release delays.

SYSTEM ARCHITECTURE

The general architecture of the TGDS was shown in Fig.1. It consists of one workstation, a variable number of processing nodes and a LAN- an Ethernet.

The reason for the system to be distributed is that one of the TGDS objectives is to generate and detect traffic simultaneously at various network access points, since to simulate real load conditions, the traffic must be injected in one network station and captured at another station. In order to have a large freedom in defining links several generating/detecting PNs must be allowed. The workstation will work as an access point, as the interface between the user and the distributed environment.

Hardware

The processing nodes, whose general architecture may be observed in Fig.2, consist of a supervisor board and two dedicated generation/detection boards.

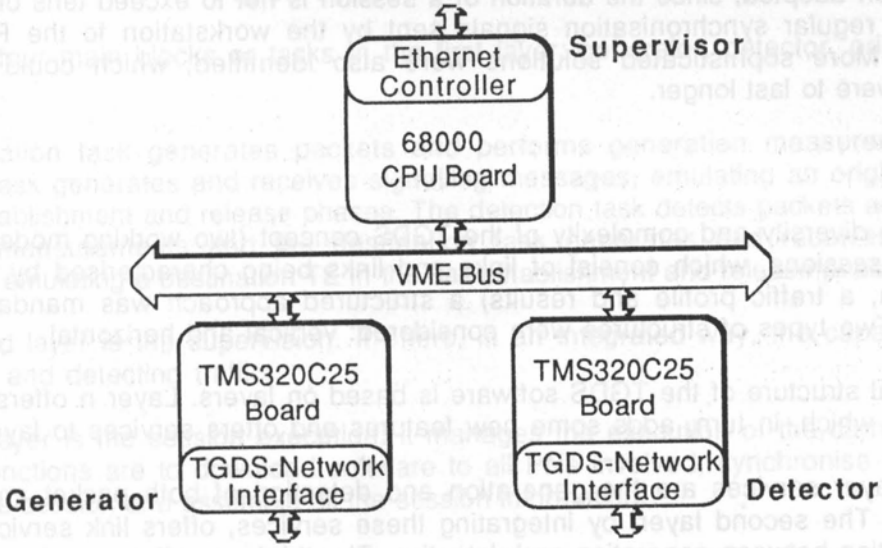


Fig.2. Processing node architecture

The supervisor board handles the communications across the Ethernet and manages the dedicated boards. The dedicated boards may have the functions of generating or detecting packet traffic or be a caller or a called TE in signalling traffic. The supervision board is based on the Motorola 68000 and supports an Ethernet controller as a piggyback, because one 68000 processor seems powerful enough to supervise the dedicated boards and control the Ethernet communications.

The generation/detection boards are based on the Texas Instruments DSP TMS320c25 and are connected to the supervision board via a VME bus.

The TGDS-NETWORK interface is defined as independent boards and plugged on the dedicated boards. Adapting the TGDS to a new network only requires the redefinition of part of this interface.

When generating packet traffic, the generator board will not only calculate the next instant to generate a packet and place in the packet some relevant pieces of information such as time stamps and packet sequence number, but also carry out some generation measurements. At the detector, when a packet arrives, the pre defined detection measurements are performed. The reason to have two dedicated boards, one for generation and another for detection, is that the packet rate required is to high to be handled by a single board.

The clock synchronisation of both dedicated boards in the PNs is an important issue. Since the traffic generated by one board is detected by a different board, either in the same or in another PN, to measure packet delays with the precision required it is necessary to guarantee the same time reference on the entire system.

The solution adopted, since the duration of a session is not to exceed tens of seconds, is based on regular synchronisation signals sent by the workstation to the PNs, via the Ethernet. More sophisticated solutions were also identified, which could be used if sessions were to last longer.

Software

Due to the diversity and complexity of the TGDS concept (two working modes, each one based on sessions, which consist of links and links being characterised by a source, a destination, a traffic profile and results) a structured approach was mandatory for the software. Two types of structures were considered: vertical and horizontal.

The vertical structure of the TGDS software is based on layers. Layer **n** offers services to layer **n+1**, which, in turn, adds some new features and offers services to layer **n+2**.

The first layer services are the generation and detection of both packet and signalling messages. The second layer, by integrating these services, offers link services: there is an association between generation and detection. The third one offers session services: an association is made by links. The fourth, by using sessions, executes sessions groups or uses sessions in a specific way. The fifth is a graphic interface between the user and the system.

The horizontal structure is used in a layer, to divide software into smaller blocks. Blocks are independents and grouped by the proximity of functions or services they offer or by physical association. The TGDS software structure, assuming three PNs, can be observed in fig.3.

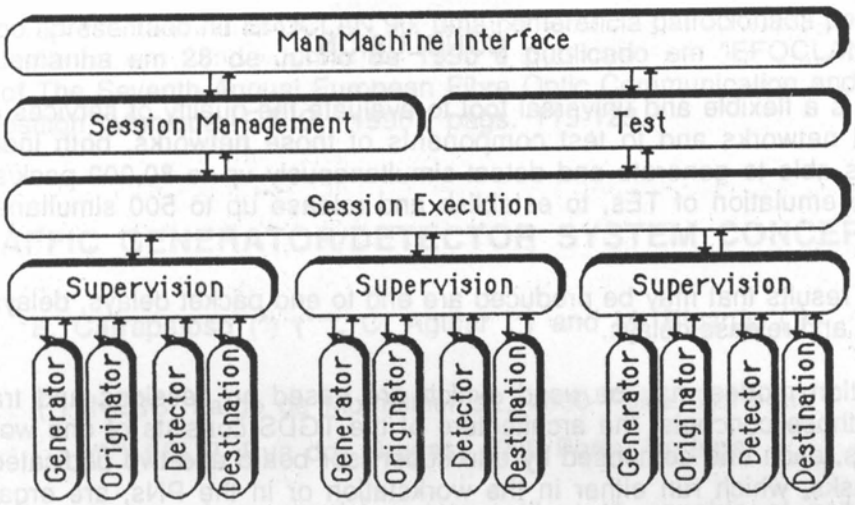


Fig.3. System software model

There are four main blocks or tasks in the first layer: generator, detector, originator and destination.

The generation task generates packets and performs generation measurements. The originator task generates and receives signalling messages, emulating an originator TE in the call establishment and release phases. The detection task detects packets and performs detection measurements and the destination task generates and receives signalling messages, emulating a destination TE in the call establishment and release phases.

The second layer is the supervision. It offers, in an integrated way, the capacity of both generating and detecting traffic.

The third layer is the session execution: it manages the execution of the current session. Its main functions are to download software to all PNs involved, synchronise them, begin and end the session and assemble all the session information.

In the fourth layer there are the session management and test tasks. The former handles the management of a session or groups of sessions for performance evaluation. The latter caters for the real time visualisation in the test mode and offers to the user, via the upper layer, some services to modify some of the session parameters.

The last layer is the man machine interface. It allows the user to access the system. Through it the user can define sessions, graphically analyse the results, order the execution of a session, define new probability distribution functions and define traffic profiles.

The layers one and two run in each PN. Layers three, four and five run in the workstation.

CONCLUSION

The TGDS is a flexible and universal tool to evaluate the quality of services offered by the high speed networks and to test components of those networks, both individually or in groups. It is able to generate and detect simultaneously up to 80,000 pack/s per PN and, through the emulation of TEs, to establish and release up to 500 simultaneous calls per PN.

Among the results that may be produced are end to end packet delays, delays jitter, packet loss, set up and release delays.

Two operation modes may be used, which are based on sessions and traffic links. To implement those concepts, the architecture of the TGDS consists of one workstation, and various PNs, each one composed by one supervisor board and two dedicated boards. The software tasks, which run either in the workstation or in the PNs, are organised both in vertical and horizontal structures.

ACKNOWLEDGMENT

The work presented here is being partly supported by two Esprit II projects: E.2054 UCOL and E.2146 DAMS.

REFERENCES

1 "UCOL: Ultra-Wideband Coherent Optical Lan", Fourth Call for Proposals, Technical Annex, 20th January 1989.

2 "DAMS: Dynamically Adaptable Multiservice System", Fourth Call for Proposals, Technical Annex, July 10th 1989.

3 "Draft Recommendation I.211", B-ISDN Service Aspects, TD 19 (ISDN), CCITT SGXVIII, Geneva, 8-19 January 1990.

4 "Telephone network and ISDN-Quality of service, network management and traffic engineering", Recommendations E.401-E.880 (Study Group II), Blue Book, Fascicle II.4, 1988.

5 "Integrated Services Digital Network (ISDN) - Overall network aspects and functions, ISDN user-network interfaces", Recommendations I.310-I.470, Fascicle III.7, Volume III, Blue Book, CCITT, 1988.

Cópia do artigo apresentado na EFOCLAN 90, uma conferência patrocionada pelo IEEE, em Munique - Alemanha em 28 de Junho de 1990 e publicado em "EFOCLAN 90 - LAN Proceedings of The Seventh Annual European Fibre Optic Communication and Local Area Network Exposition, Munique, F.R.G., 1990", págs. 119-123.

TRAFFIC GENERATOR/DETECTOR SYSTEM CONCEPT

E. Carrapatoso (*) (**), C. Aguiar (*) and M. Ricardo (*)

(*) INESC, Largo de Mompilher 22, 4000 Porto, Portugal

(**) FEUP, Rua dos Bragas, 4000 Porto, Portugal

Summary: The concept of a powerful and versatile tool, capable of generating and detecting traffic to test and assess the performance of high speed networks, is presented.

1. INTRODUCTION

From the user perspective, communication networks of the next generation will be different from their predecessors, such as Ethernet and narrowband ISDN (N-ISDN), in two main aspects: speed and service integration capabilities.

Commercially available LANs have, typically, data rates in the order of 10 Mbps and the services supported by N-ISDN are, usually, voice and data. But products belonging to the next generation will have speeds an order of magnitude higher [1] [2], and video services will be among those offered [3].

In this paper we present the concept of a test and performance measurement device, the traffic generator/detector system (TGDS), suitable to generate and detect high bit rate traffic, and which we believe will prove a valuable help in the development and testing of those networks.

2. THE NEED FOR THE TGDS

Recent developments in distributed systems, optical fibre technology, fast electronics and scale of integration have lead to the conception and building of prototypes for LANs and MANs with features which would have been unthinkable some years ago.

High speed communications among computers in an office or a campus environment have been one of the driving forces: they will soon require capacities that today's LANs/MANs will not be able to provide. These communications involve either machines connected to the same network or those which are attached to different networks and interconnected across a backbone.

Telecommunications have also played a key role: first, the narrowband ISDN offering digital communications with service integration (voice and data) up to the user's

premises, followed by the broadband ISDN (B-ISDN), which will add to those services others that require a much greater capacity (video services). Up to now separate networks provide the various services but the trend is for a single network with integrated services, which is currently being the object of much research and standardisation effort, providing the subscriber with a variety of services.

Nevertheless a long way has still to be covered since old topologies and protocols cannot cope with the new and stringent requirements. The old solutions were optimised for either circuit switched or packet services, but not for both. This fact has brought about a new transfer mode, ATM [4], which is, in principle, suitable for both, on account of its flexibility. But this solution requires new switching and management strategies, innovative and efficient.

It is important, in the process of defining those new network architectures, to well characterise their influence over the traffic that will be transported, so that its negative effects may be minimised. The quality of services (QOS) to be provided is also under standardisation efforts [3] in an attempt to, ultimately, guarantee the subscribers performance requirements.

Therefore, in the conception phase of the networks, the need arises to simulate possible alternative solutions in order to make the best choice in what concerns information transfer protocols, signalling and even management and in order to see how those decisions affect the QOS.

However, all simulations require models and models always imply simplification of the reality. Hence, it is not possible to guarantee that the system will perform as predicted, under real traffic conditions.

Simulation is therefore mandatory but may prove to be not enough. Prototypes have to be built and thoroughly analysed, under the most varied and stringent traffic conditions, in order to identify possible limitations and critical parameters which severely affect the overall behaviour. To achieve this purpose,

1. a tool, capable of assessing the performance under various load conditions, is required;

2. a device to define and modify the traffic load on the system over a large range of values is also required, for it is not practical to use actual traffic sources.

Since no tool is available for high speed networks, which may be used for those purposes, two ESPRIT II projects, in which we participate, decided to develop it: UCOL and DAMS. Both projects involve high speed optical networks: UCOL [6], whose objective is the provision of cost-effective local area wideband communication capacity to meet primarily industrial and commercial requirements, is a passive star network using coherent techniques and DAMS [7], aiming at the development of an advanced communication system for the private communications market, is based on FDDI II.

Other applications that may also be envisaged for such a tool include the testing of control and management functions, which will involve the exchange of well defined messages between the TGDS and the subsystem whose functionality is under test.

3. APPLICATIONS OF THE TGDS

In very general terms, the traffic generator/detector system may be described as a real time distributed system capable of generating and detecting traffic.

A system is said to be real time when it reacts to random solicitations from its environment (control or information) in a time which is upper bounded. It is said to be distributed when it is subdivided into autonomous subsystems which execute functions concurrently, but that cooperate to realise a common objective.

Primarily, the traffic generated by the TGDS will be used to assess the network or some of its components, either by measuring packet delay or by simulating ISDN calls and measuring signalling delay. That traffic will enter the system at a certain point and then emerge at some other location.

Unlike computer simulation, in which traffic sources are modelled by probability distribution functions (PDF) but traffic is not actually produced, the TGDS will generate, in most cases, packet traffic, also according to distributions, but which will be transported by the system under test.

The traffic generated will usually be received by some component of the TGDS so that some measurements may be carried out to assess the system performance. If the traffic is not detected, that will prove that something is wrong in the chain of components being tested.

These measurements will show how well the various components found in the path from the generation to the detection cooperate to deliver information units and also to characterise the overall delay introduced by the system.

In the case of ATM networks, packets, produced/generated by sources, may be either ATM cells (48+5 octets) or other types of packets: in the first case, cells may be transmitted immediately whereas, in the second, packets will have to go through the ATM adaptation sublayer [5] which may, therefore, be also tested.

3.1 Packet Delay

To assess the system performance through packet traffic, the TGDS will have to generate packets according to statistical laws which modulate not only the packet arrival processes but also their lengths and burstiness.

The generation processes will be accompanied by detection processes whose responsibility will be to receive outgoing packets. Packets will have a time stamp field which contains the information of the instant they left the generator. At the receiving end, the detector may then compare that information with the current time and determine the delay experienced by packets. For this value to be accurate, particular attention has to be given to the synchronisation of the distributed clocks.

The parameters which may be measured are:

- end-to-end delay

- interpacket intervals
- packet loss.

The first two are given by their histogram, mean, variance and maximum value and the third by its probability.

The first and last are self explanatory but a few words about the second are in order. Interpacket intervals are measured both at the generator and at the detector; at the first since, although a given distribution was selected, it may not be possible to strictly comply with that requirement and this parameter allows the checking of the differences; at the detector to be able to ascertain the differences between the originator and the destination, or, in other words, in order to evaluate the delay jitter introduced by the system. All these parameters are mandatory if QOS is to be assessed.

3.2 Signalling Delay

Another possibility, to assess system performance, is the simulation of ISDN calls to measure signalling delays. It must be noted that the objective is not to validate call control and, therefore, not all the signalling protocol for call establishment and release need to be emulated. The aim is, solely, to measure delays incurred by such requests, and this implies that only a basic subset of those procedures should be implemented.

To achieve this objective, the TGDS will have to generate signalling messages requesting the establishment and release of calls, according to statistical laws which modulate not only the call arrival processes but also their duration.

This feature should allow the test of the components involved in these processes and also ensure that setup and release delays are within limits recommended by standardisation bodies [8].

4. WORKING PRINCIPLE

The TGDS working principle is illustrated in Fig.1, for three attachment points. This number is given as an example only; it does not correspond in any way to the maximum that may be supported.

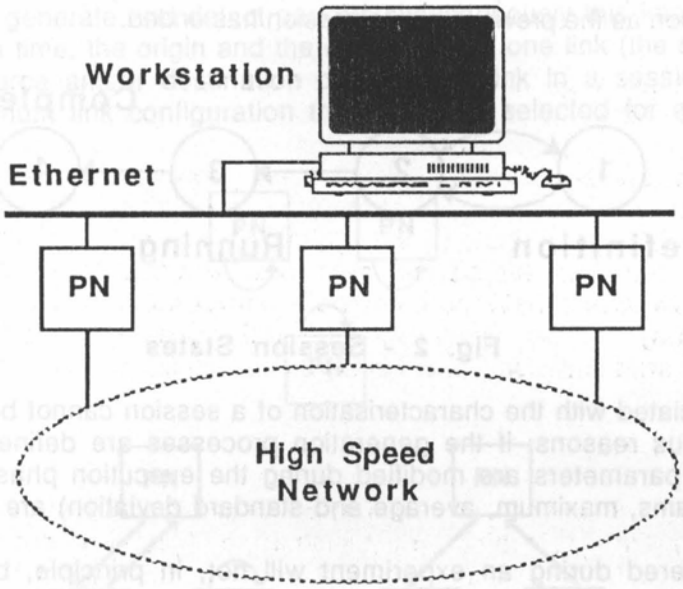


Fig. 1 - TGDS Model

A processing node (PN) is an entity capable of:

1. generating packets according to the selected statistical laws
2. detecting packets sent to it
3. performing, over the traffic generated and/or detected, the measurements selected.

Those PNs are interconnected among themselves and to a Workstation (WS) by means of an Ethernet. The WS is the TGDS component where the man machine interface resides: it is used to offer a good definition and analysis environment, to inform the user about what is happening in the entire system and to manipulate conveniently the data structures.

In order to characterise the network, both from the performance and QOS point of view, and to test its components, experiments (sessions) have to be executed.

4.1 Session

A session may be defined as the time interval during which a set of PNs will generate and detect traffic. Their activity will start and end at the same time. In a session four states may be clearly identified: definition, queued, running and completed.

Before a session enters its execution phase (running) it must be defined and, to achieve that, the user (operator) will characterise the traffic to be generated, its destination and the results to be collected both at the generation and at the detection.

After the definition, a session will join the queue of sessions waiting for execution. During the execution phase, all selected PNs will generate and detect traffic as specified in the

definition phase. This phase may be entered in one of two ways: either at a predefined time of the day or as soon as the previous active session has ended.

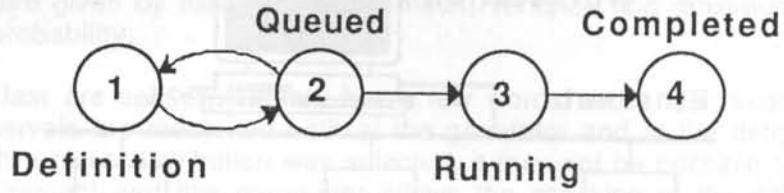


Fig. 2 - Session States

Parameters associated with the characterisation of a session cannot be modified while it is running for obvious reasons: if the generation processes are defined through statistical laws and if their parameters are modified during the execution phase, the results being collected (histograms, maximum, average and standard deviation) are meaningless.

The results gathered during an experiment will not, in principle, be displayed in real time but will be made available at the WS when the session reaches its end (completed state). The reasons for this are:

1. each session is not likely to last more than a few seconds, since that, in 10 seconds, and using 20% of the PN full capacity only, 80,000*0.2*10 = 160,000 packets may be generated, which seems enough to perform measurements with a fairly good degree of certainty (80,000 packets per second is the minimum generation/detection rate that each PN should be capable of supporting). And with such short duration, an update of results obtained 'so far' does not seem to make sense.

2. if the TGDS resources are not required to exchange information during a session then they can devote all their capacities to the activities of generating and detecting traffic and that will be reflected in the maximum number of packets per second that may be processed: 80,000 packets per second is a value that may be reached, in each PN, if no other tasks interfere with the generation and detection.

3. if other tasks interfere with the generation and/or detection processes, the precision of the measurements being made may be impaired.

It will be possible to define sessions with much longer duration or even session whose duration is not known in advance if the TGDS is to be used to load the network, i.e., to create some kind of background load, with no measurements being carried out, on top of which some particular service or signalling procedure is to be tested.

4.2 Link

To render the definition of the generation/detection environment of a session more straightforward and less error prone, the concept of link has been introduced. A link is a packet flow from one PN to another PN.

Since each PN may generate and detect packets simultaneously this implies that each PN may be, at the same time, the origin and the destination of one link (the same or not). Any PN may be the source and/or destination of only one link in a session. In Fig. 3 are illustrated the maximum link configuration that could be selected for a TGDS having 3 PNs.

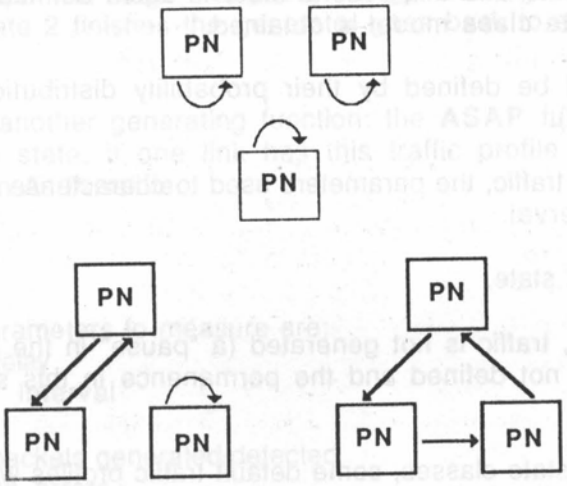


Fig. 3 - Possible Link Configurations involving 3 PNs

Each link requires the following attributes for its definition:

- Source
- Destination
- Traffic profile
- Results.

Source

To characterise a link, the first attribute to choose is its source, i.e. the PN which will generate packets. Any PN may be the source of only one link in a session.

Destination

Destination is the second parameter to define, although it is not mandatory: links may have no destination, if their only function is to load the system. Each PN may be the destination of only one link.

Traffic Profile

The user may either select one of the pre-defined profiles or define one of his own.

Traffic is defined by means of two parameters, packet length and inter-packet intervals, and may be of two types: **single state** or **two state.**

The **single state class** traffic is characterised by the length of packets and the inter-packet interval. If we also define the time that the system remains in this state before switching to another state and this second state is again defined with the same type of parameters, a **two state class** model is obtained.

These parameters will be defined by their probability distribution function, mean and variance (when needed).

For the two state class traffic, the parameters used to characterise each state are:
- inter-packet interval
- packet length
- time spent in that state.

If, in one of the states, traffic is not generated (a "pause" in the generation occurs), the inter-packet interval is not defined and the permanence in this state is defined in time units.

For both, one and two state classes, some default traffic profiles will be defined, providing the user with the capacity of generating traffic according to models found in the literature for certain services. PDFs will also be available which may be used as the building blocks to define other profiles.

One State: In this class, three default profiles were assumed: deterministic, Markovian and Gaussian.

The difference between profiles is the distribution of the intervals between consecutive packets. For all of them, the packet length was assumed to be constant.

Deterministic: the inter-packet interval is constant.

Markovian: the inter-packet interval has an exponential distribution.

Gaussian: the inter-packet interval has a Gaussian distribution.

Two State: In this class, two default functions were assumed, again with constant length packets: deterministic burst and Poisson burst.

Deterministic Burst: In both states
- the inter-packet interval is constant
- packet length is constant
- the time spent in each state is constant.

Note that the generator alternates between state 1 and state 2: the generator begins, for example, in state 1 and, after the generation of N_1 packets, goes to state 2 remaining there for a predefined time. After that, the generator goes back to state 1 and the cycle goes on until the simulation time ends.

Poisson Burst: In both states
- the inter-packet interval has an exponential distribution
- packet length is constant
- the time spent in each state has an exponential distribution.

Again, when the time spent in state 1 elapses, the generator goes to state 2. When the permanence time in state 2 finishes the generator goes back to state 1 and the cycle is repeated.

ASAP: There will be another generating function: the **ASAP** function. This function is neither single nor two state: if one link has this traffic profile the generator PN will generate traffic **A**s **S**oon **A**s **P**ossible.

Results

The most significant parameters to measure are:
- end to end delay
- interpacket interval
- packet loss
- number of packets generated/detected
- packets with errors.

End to end delay: It is the delay experienced by a packet since it is injected in the network (since it leaves the generator) until it arrives at the detector.

End to end delay will be shown by means of:
- a histogram
- the mean and the variance
- the maximum delay.

This value will be measured by the detection PN.

Interpacket interval: This time is measured between consecutive packets. This kind of information is useful to compare the traffic actually generated with the traffic requested by the user. It may also be used to compare interpacket gaps in the generator with those in the detector in order to observe, statistically, the jitter introduced by the network.

Interpacket interval will be shown by means of:
- a histogram
- the mean and the variance
- the maximum interpacket interval.

This value can be measured by the generation and detection PNs.

Packet loss: Packet loss informs about the number of packets generated per link that do not arrive at the detectors.

This value will be measured by the detectors, based on missing sequence numbers.

Poisson Burst: In both states
- the inter-packet interval has an exponential distribution
- packet length is constant
- the time spent in each state has an exponential distribution.

Again, when the time spent in state 1 elapses, the generator goes to state 2. When the permanence time in state 2 finishes the generator goes back to state 1 and the cycle is repeated.

ASAP: There will be another generating function: the **ASAP** function. This function is neither single nor two state: if one link has this traffic profile the generator PN will generate traffic **A**s **S**oon **A**s **P**ossible.

Results

The most significant parameters to measure are:
- end to end delay
- interpacket interval
- packet loss
- number of packets generated/detected
- packets with errors.

End to end delay: It is the delay experienced by a packet since it is injected in the network (since it leaves the generator) until it arrives at the detector.

End to end delay will be shown by means of:
- a histogram
- the mean and the variance
- the maximum delay.

This value will be measured by the detection PN.

Interpacket interval: This time is measured between consecutive packets. This kind of information is useful to compare the traffic actually generated with the traffic requested by the user. It may also be used to compare interpacket gaps in the generator with those in the detector in order to observe, statistically, the jitter introduced by the network.

Interpacket interval will be shown by means of:
- a histogram
- the mean and the variance
- the maximum interpacket interval.

This value can be measured by the generation and detection PNs.

Packet loss: Packet loss informs about the number of packets generated per link that do not arrive at the detectors.

This value will be measured by the detectors, based on missing sequence numbers.

Number of packets: It refers to the number of packets generated and detected in a link, measured by generators and detectors.

Packets with errors: This parameter informs about the number of packets generated that arrive at the detectors with an error indication. Of course this information may be shown only if the lower layers are able to provide the TGDS with it.

This value will be measured by detectors.

5. CONCLUSIONS

In this paper the TGDS concept was presented, together with those of session and link. A session is the basic simulation unit and may stay in one of the following states: definition, queued, running and completed.

A session consists of links, which are characterised by: source, destination, traffic profile and results. Each processing node will generate at most one link per session.

The TGDS consists of a workstation, several PNs and an Ethernet. The WS will be in charge of the dialogue with the user, the PNs will generate and detect traffic and the Ethernet will be used to convey information between the PNs and the WS.

ACKNOWLEDGEMENT

The work presented here is being partly supported by two ESPRIT II projects: E.2054 UCOL and E.2146 DAMS.

The authors also wish to thank Mr. V. Nicodemo and Dr. D. Wybranietz for helpful discussions and suggestions, and Dr. J. Ruela for his review work.

REFERENCES

[1] - Proposed Standard Distributed Queue Dual Bus (DQDB) Metropolitan Network (MAN), IEEE 802.6, Unapproved Draft, November 15, 1988.

[2] - FDDI token ring media access protocol (MAC), Amer. Nat. Stand., draft proposal X3T9/84-100, Feb. 28, 1986.

[3] - Draft Recommendation I.211; B-ISDN Service Aspects, TD 19 (ISDN); CCITT SGXVIII; Geneva, 8-19 January 1990.

[4] - Draft Recommendation I.150; B-ISDN ATM Functional Characteristics, TD 28 (ISDN); CCITT SGXVIII; Geneva, 8-19 January 1990.

[5] - Draft Recommendation I.362; B-ISDN ATM Adaptation Layer Functional Description, TD 28 (ISDN); CCITT SGXVIII; Geneva, 8-19 January 1990.

[6] - UCOL: Ultra-Wideband Coherent Optical Lan, Forth Call for Proposals, Technical Annex, 20 January 1989.

[7] - DAMS: Dynamically Adaptable Multiservice System, Forth call for Proposals, Technical Annex, July 10th 1989.

[8] - Telephone network and ISDN - Quality of service, network management and traffic engineering, Recommendations E.401-E.880 (Study Group II), Blue Book, Fascicle II.4, 1988.

Cópia do artigo apresentado na Bilcon 90, uma conferência patrocionada pelo IEEE, em Ankara - Turquia em 3 de Julho de 1990 e publicado em "Communication Control and Signal Processing - Proceedings of the 1990 Bilkent International Conference", Vol. 1, págs. 76-82.

The usage of prediction in demand assignment with distributed control protocols

C. AGUIAR[1], E. CARRAPATOSO[1,2], M. RICARDO[1]
F. D'IGNAZIO[3], and A. FANTINI[3]

[1]INESC, Largo de Mompilher 22, 4000 Porto, Portugal
[2]FEUP, Rua dos Bragas, 4099 Porto Codex, Portugal
[3]Alcatel Face Research Centre, Via Nicaragua 10, 00040 Pomezia, Italy

ABSTRACT

In this paper, a prediction mechanism, capable of considerably improving the delay characteristics of MAC protocols based on demand assignment with distributed control, is presented and its effects analysed through simulation.

INTRODUCTION

The emergence of new services in the telecommunications scene has brought about the need for new networks with different topologies and configurations but, although the technological evolution has significantly increased the transmission and switching capacity of telecom systems, the efficient use of resources is still a main concern for designers.

One important factor which may severely affect overall network performance is the medium access control protocol, particularly its delay characteristics: additional delay, introduced by the MAC layer, may not only degrade service quality but will almost inevitably demand extra buffer capacity.

Protocols based on demand assignment with distributed control usually rely on a capacity reservation scheme that, although showing some relevant features, implies an additional delay for the requests to be processed. However, as it will be shown in the following sections for one of those protocols, that situation may be altered by the introduction of a prediction mechanism, which significantly reduces that delay.

A BRIEF INTRODUCTION TO THE UCOL SYSTEM

The UCOL project [1] aims at the realization of a large communication system whose main objective is to fully exploit the huge bandwidth of fibre optics to simultaneously transmit on a large number of wideband channels (100 at approximately 160 Mbps) using optical frequency division multiplexing.

The information units in the UCOL system are ATM cells: on each optical carrier a time division multiplexed channel is implemented, where ATM cells can be allocated to more than one session, thus enabling different users to share the same channel. Since ATM can be used both for synchronous and asynchronous traffic, these classes of traffic can be transported on the same channel.

UCOL relies on a passive star topology where the star centre couples the signals coming from the in-bound fibres into a composite signal which is propagated along the out-bound fibres and distributed (broadcast) to all stations in the network. The UCOL network can have as many as 64 stations, which may be equipped with up to 15 Network Interfaces (NI); each NI has a transmitter and a receiver part. These two parts are connected to the optical star by means of optical fibres, up to 50 Km long: one fibre carries the (single-channel) signal to the star centre, the other carries the (multi-channel) signal from the star centre.

ACCESS PROTOCOL BASIC CONCEPTS

The Distributed Queue Access Control (DQAC) protocol used in UCOL is based on the following principle: requests for access are forward to the network according to the information held in the local queues, the access control being intrinsically a distributed function since every station reads the requests from all other stations and processes them before physical access is actually performed.

Even though UCOL is a multi-channel network, the access protocol procedures are applied to a SINGLE CHANNEL and each entity handling a channel operates INDEPENDENTLY from all the others.

The basic features of DQAC are explained hereafter:

1 - **optimised signal multiplexing at the optical star.** An optimised signal multiplexing scheme at the star centre requires that data bursts coming from different stations must be separated by a time gap negligible when compared with the duration of the burst: by making the gap as short as possible, a higher efficiency, in terms of bandwidth exploitation, can be achieved. The condition that has to be fulfilled in order to achieve this is that every station must know the exact time, referred to the optical star, at which its data bursts have to reach the star centre.

This, in turn, translates into the following conditions:

- a time reference has to be cyclically distributed to all the stations

- every station has to know in advance where to put its data bursts within the cycle
- every station has to anticipate the transmission of its data bursts by an amount of time equal to the propagation delay along the fibre.

The first condition is satisfied by the adoption of a frame based transmission scheme since the cyclically transmitted frame synchronism can also be used as the star centre reference time signal.

The second condition can be satisfied by providing all stations with the necessary information; to understand how this is done, some additional concepts about the frame structure will be introduced in the next section.

In order to satisfy the third condition, each station, during Station Initialisation, measures its own distance from the star centre.

2 - adoption of a frame based transmission scheme. The frame is 1 ms long and has the following fields: Frame Header, Queue Status (QS) and Information.

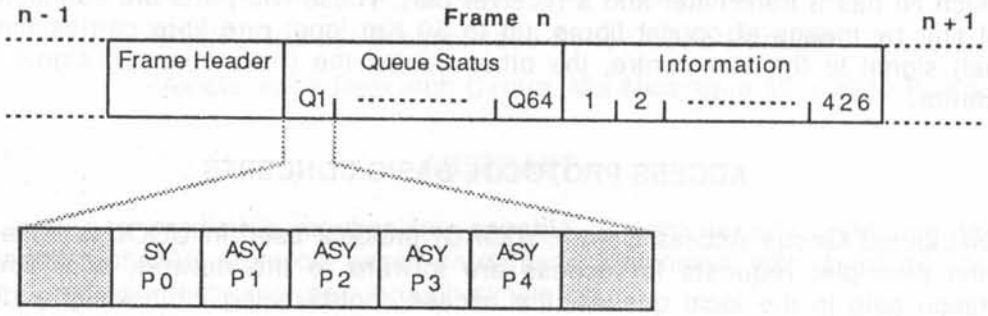


Figure 1 - Frame Structure

The Information field is divided into 426 subfields (data slots), each able to contain one ATM cell. The QS field contains 64 QS subfields, one for each station, according to a fixed transmission order: the first subfield is reserved to station 1, the second to station 2 and so forth. This implies that, for the QS field, the second condition is satisfied.

The QS field provides all stations with the information they need to decide which data slots they will use. The information carried by the QS field of frame **n** is used to arbitrate the access to the Information field of frame **n+1**: so, when a new frame starts, every station already knows (according to a predefined algorithm executed by all stations) which slots of the Information field it will use in that frame (thus satisfying the second condition also for the Information field).

3 - the QS based access mechanism. Each QS subfield indicates, for each station, how many cells are held in each of the five priority queues (P0 to P4).

The slot allocation algorithm is quite simple: after all the stations have received the QS they begin to elaborate it concurrently. Starting from the highest priority (P0),

slots are assigned, one to each station (if, of course, that station has a cell to transmit), until every station has obtained a number of slots equal to the number of cells queued. Only after all slots of a given priority have been assigned, will slots of the next priority undergo the allocation procedure.

Synchronous traffic is assigned priority P0, whilst priorities P1 to P4 are used for asynchronous traffic.

PREDICTION CONCEPT

A network, such as UCOL, must handle traffic generated by bursty sources. Although, due to statistical multiplexing, load fluctuations may be smoothed, each individual source may generate peak loads during relatively short periods, thus producing a high peak to average load ratio.

The basic DQAC protocol does not react quickly to sudden changes of bursty traffic and that may lead to bandwidth waste; in fact, all cells must wait at least for the transmission of the QS to request slots for them in a future frame, even if the current and the next frame are not fully occupied. This fact, not only causes unnecessary delay but, may also lead to congestion situations if, meanwhile, too many cells have arrived.

However, it is possible to try to predict the load that will be generated in the near future and request bandwidth in advance for that load. This scheme enables the reservation of slots, that would otherwise be wasted, for the transmission of recently generated cells. If these reserved slots are in fact used, there is a gain; if they are not used, there is no loss relatively to the basic algorithm. Another important characteristic of this scheme is that, if a frame is already full, the prediction requests are ignored and not even processed.

In order to request those slots separately from the normal QS requests, the need to include a new field in the QS of the UCOL frame arises: the Prediction (PRD) field. The two separate requests (QS and PRD) make it possible for the Slot Allocation algorithm, running in each station, to allocate, in the first place, slots for the QS requests and then, if slots are still available, to allocate them for PRD requests.

The frame should support separate predictions fields, one for each priority, so that the slot assignment is also prioritised for the Prediction requests (Fig. 2).

	Prt 0		Prt 1		Prt 2		Prt 3		Prt 4		
	QS	**PRD**	QS	**PRD**	QS	**PRD**	QS	**PRD**	QS	**PRD**	

Figure 2 - A Possible Station QS Field

Prediction Methods

Several prediction algorithms could be envisaged, but although accuracy is highly desirable, there are two other characteristics that should also have a great weight in

selecting the most suitable: simplicity and speed. In this section, we present three methods that comply with these requirements.

In order to predict future load, one technique is to measure and save past and present load history. For that purpose cell counters are used: counters hold the number of cells that arrive to a queue during each frame interval. The following prediction methods use at most two counters: **Nc** - number of cells generated during the current frame interval and **Np** - number of cells generated during the previous frame interval.

Method A	PRD := Nc;
Method B	PRD := 2*Nc - Np; if (PRD < 0) then PRD := 0;
Method C	if (Nc > Np) then PRD := 2*Nc - Np; else PRD := Nc;

Method A assumes that, in the near future, load will be the same as in the recent past. Its major drawback is that it does not take into account the evolution of the load and, for that reason, it does not react quickly to load changes.

Method B is a linear extrapolation of the two previous load values. A simple analysis will show that it reacts quickly to load fluctuations. Although this is a great advantage when the load is increasing, it is also a drawback when the load is decreasing, in case of a transient situation.

Method C is a mixture of both methods, retaining their best attributes: fast response to increasing loads and slower response to decreasing loads.

Since it is not possible to achieve 100% accuracy, the best principle to adopt is that it is preferable to over-estimate the number of required slots rather than under-estimate it because, in the normal operation of a network, the average offered load will be, most of the time, well below 100% of the total capacity.

SIMULATION ANALISYS

The performance evaluation of the DQAC basic protocol and its enhanced versions with the three prediction methods was carried out through simulation. The model was developed to study the error free operation of the protocol and, for the sake of analysis simplicity, the results obtained with only one priority will be presented. However, more complete studies [2-4], both for isochronous and non isochronous sources, also consubstantiate the results presented hereafter.

Simulation Model

Cells are generated according to Poisson processes, identical for all 64 stations. This seems a reasonable approximation of the traffic that may be expected in a real situation,

considering that each station will act as a concentrator of multiple, statistically independent and bursty traffic sources.

Each frame has 426 slots and each cell 48+5 octets. Stations are equidistant from the centre of the star and their MAC buffers infinitely long. It is also assumed that the QS and PRD fields of the frame can hold requests for at least the entire capacity of the frame (426 slots).

Results

Fig. 3 shows the average and maximum waiting time experienced by cells since they arrive at the MAC buffer until they are, eventually, transmitted. Offered load is the average number of cells generated, as a percentage of the 426 slots in a frame.

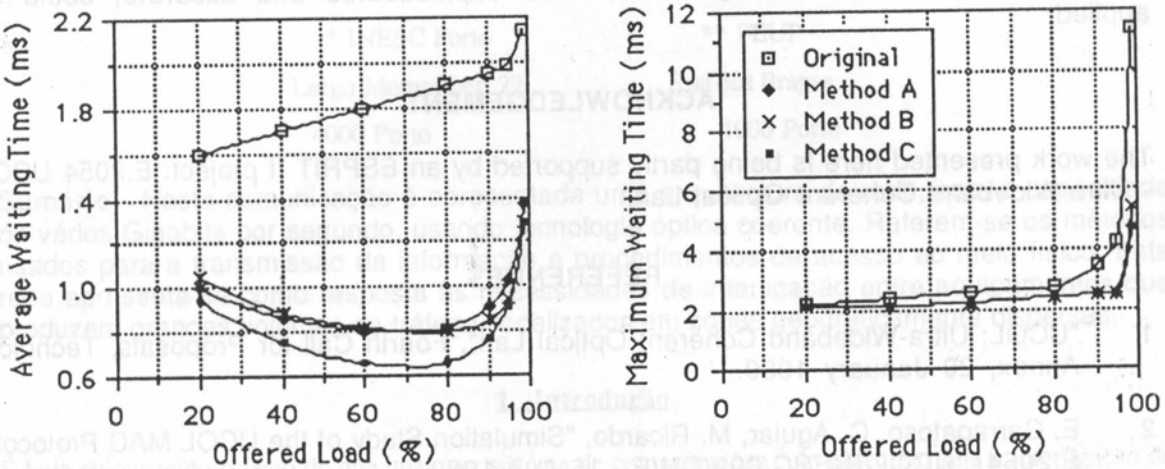


Figure 3 - Waiting Time vs. Offered Load

In the original version of the protocol, cells experience, from the outset, and even for very low loads, an average delay of at least 1.5 ms. This is due to the fact that a cell has to wait, since its arrival, for the correspondent request to be transmitted in the next QS and, only in the following frame, is the cell in condition to be sent. With the new mechanism this is no longer the case and a cell may well be transmitted during the frame interval of its arrival.

Observing Fig. 3 it is evident that any of the protocols using prediction shows better performance than the original protocol, with gains varying from 30% to more than 60%. The three methods perform similarly, being method C slightly better than the others.

A greater difference between methods A and B could be expected, but, as previously mentioned, the quick response to decreasing load compensates the gain obtained with increasing load. Method C, which has not this disadvantage, reveals itself better over the entire load scale.

CONCLUSIONS

Errors are inherent to any process that tries to predict the future. Two types of prediction errors can occur: under or over estimation. The first type of errors explains the negative slope of the curves for the prediction methods (Fig. 3) with the load increase, because the ratio error to load decreases with increasing load. The fact that curves start rising earlier than for the original protocol is explained mostly by the second type of errors: requests exceed the number of slots available per frame.

Nevertheless, the prediction mechanism represents, unquestionably, a considerable performance improvement for the DQAC. And UCOL is not the only application: it could also produce significant gains for other reservation protocols, with minor costs. And if cost is irrelevant, other prediction algorithms, more sophisticated and accurate, could be applied.

ACKNOWLEDGEMENT

The work presented here is being partly supported by an ESPRIT II project: E.2054 UCOL - Ultra-Wideband Coherent Optical Lan.

REFERENCES

1 "UCOL: Ultra-Wideband Coherent Optical Lan", Fourth Call for Proposals, Technical Annex, 20 January 1989.

2 E. Carrapatoso, C. Aguiar, M. Ricardo, "Simulation Study of the UCOL MAC Protocol", E.2054: UCOL/INESC/009/DN/F, 16-Aug-1989.

3 C. Aguiar, E. Carrapatoso, M. Ricardo, "An Enhanced Version of the UCOL MAC Protocol", E.2054: UCOL/INESC/011/DN/F, 29-Sep-1989.

4 C. Aguiar, E. Carrapatoso, M. Ricardo, "Comparison of Prediction Methods", E.2054: UCOL/INESC/016/DN/F, 01-Mar-1989.

Cópia do artigo apresentado na ENDIEL 91, uma conferência patrocionada pela ANIMEE, em Lisboa em 1 de Junho de 1990 e publicado em "ENDIEL 91, ST3 - Comunicações, Redes, Equipamentos e Serviços", págs. 371-378.

REDE INTEGRADA DE MULTI-SERVIÇOS USANDO TÉCNICAS ATM E TECNOLOGIA ÓPTICA COERENTE

José Nuno Almeida*; José Manuel Cabral*; Manuel Alberto Ricardo*;
Eurico Carrapatoso* ,**; A. Pimenta Alves*, **.

* INESC Porto
Largo Mompilher, 22
4000 Porto

** FEUP
Rua dos Bragas
4000 Porto

Sumário - Nesta comunicação é apresentada uma arquitectura de rede, com a capacidade de vários Gigabits por segundo, usando tecnologia óptica coerente. Referem-se os métodos usados para a transmissão da informação e procedimentos de acesso ao meio físico. Esta rede apresenta-se como resposta às necessidades de interligação entre equipamentos que produzem grandes volumes de tráfego, localizados em zonas geográficamente dispersas.

1. Introdução

É hoje reconhecido o facto de que qualquer sistema de comunicações cujo objectivo seja a interligação de equipamentos produtores de elevado débito de tráfego, como estações de trabalho, computadores e terminais de vídeo, deverá ter uma largura de banda substancialmente superior à oferecida pelas actuais redes locais (LANs).

Efectivamente, é por demais notório o facto de que a quantidade de informação que necessita de ser transferida continua a aumentar rapidamente. Em particular irá notar-se um crescente aumento de pedidos da capacidade, mesmo em ambientes empresariais, para transferência de sinais de vídeo em tempo real, a acrescentar às largas quantidades de dados e informações sobre imagens de alta definição. Para transportar todo este tráfego, desta ou doutra natureza, em grandes edifícios ou entre lugares separados, o sistema de comunicações precisará duma capacidade de vários Gigabits por segundo.

Na verdade, nos últimos anos o tráfego trocado dentro de áreas confinadas tem aumentado rápidamente; hoje redes com capacidades de 100 Mbit/s já não são caso raro. A norma FDDI (Fiber Distributed Data Information), que já tem produtos no mercado, é um exemplo de uma dessas redes.

O mercado Europeu de LANs cresce a um ritmo aproximado de 40 % ao ano. Perspectivas para o futuro parecem confirmar esta tendência esperando-se que nos próximos anos o mercado esteja já pronto para aceitar redes com capacidades na ordem dos Gigabits/s.

Esta necessidade duma maior capacidade aparecerá, por exemplo, devido aos seguintes factores:
- distribuição de recursos (ao invés da actual distribuição de serviços a partir de um recurso único);
- transmissão de vídeo em tempo real;
- troca de imagens móveis, de alta definição, entre "supercomputadores" e estações de trabalho.

Torna-se assim aparente que o crescimento explosivo no uso e inter-ligação de equipamentos para processamento de dados, previsto para os próximos anos, requererá redes com grande capacidade, preparadas para interligar um grande número de utilizadores com diferentes necessidades de ritmos de transmissão, e facilmente expansíveis de maneira a cobrir o imprevisível crescimento do uso de "supercomputadores".

É neste contexto que o projecto ESPRIT nº2054/UCOL (Ultra-wideband Coherent Optical LAN) se propôs demonstrar o uso efectivo de técnicas ópticas coerentes em redes locais. Estas técnicas permitirão capacidades de sistema da ordem das dezenas de Gbit/s, resultando simultâneamente em custos relativamente baixos, visto que a capacidade total pode ser partilhada por um grande número de utilizadores de uma maneira bastante flexível.

O projecto UCOL baseia-se nos resultados de actividades iniciadas em Fevereiro de 1985, através do programa ESPRIT e com o apoio financeiro da CEE. Essas actividades determinaram que as técnicas de Multiplexagem por Divisão de Frequência (FDM) óptica, numa topologia de rede em estrela passiva, se apresentam como uma alternativa válida às técnicas actuais.

2. Generalidades e Objectivos

O objectivo deste projecto é o de realizar, no espaço de 5 anos, um protótipo laboratorial de um sistema baseado na topologia de "estrela passiva", e demonstrar a viabilidade desta LAN capaz de satisfazer requisitos de serviços futuros numa pluralidade de ambientes de utilizadores (empresarial, cientifico, médico, educacional, residencial).

O requisito chave deste projecto é a flexibilidade. Para esse efeito o conceito de sistema é baseado numa transmissão óptica multicanal, tendo sido adoptadas as seguintes soluções [1]:

- esquema híbrido de multiplexagem TDM e FDM;
- gestão distribuída;
- modo de transferência ATM (Asynchronous Transfer Mode);
- protocolo de acesso adequado e eficiente para qualquer tipo de tráfego.

O conceito de sistema desenvolvido associa a multiplexagem óptica por divisão de frequência com uma arquitectura de rede e com técnicas de gestão de maneira a permitir uma grande flexibilidade e eficiência no tratamento simultâneo dos múltiplos canais virtuais (em cada frequência óptica). Cada um destes canais virtuais pode apresentar ritmos de transmissão desde uma pequena fracção de Mbit/s até, aproximadamente, 150 Mbit/s. Aplicações específicas requerendo, individualmente, mais do que a capacidade de um único canal óptico podem ser satisfeitas através de transmissões paralelas em vários daqueles canais.

Uma grande área geográfica pode ser abrangida: por exemplo, a estrela pode oferecer até 1024 x 1024 ligações com um diâmetro inferior a 20 Km, ou até 256 x 256 com um diâmetro máximo de 50 Km.

A configuração escolhida permite ir até 64 estações e até 16 Interfaces de Rede (ópticas) por estação.

Cada Interface pode ser sintonizada para uma de entre 100 frequências diferentes, podendo operar até à capacidade oferecida por cada canal óptico (150 Mbit/s). Assim a capacidade total do sistema ultrapassa os 15 Gbit/s.

Devido à capacidade oferecida, a rede UCOL surge como a "espinha dorsal" de interligação entre diversos tipos de LANs, mesmo as de alto débito tais como FDDI ou DQDB. Acessos às redes públicas estarão também presentes (figura 1) [2].

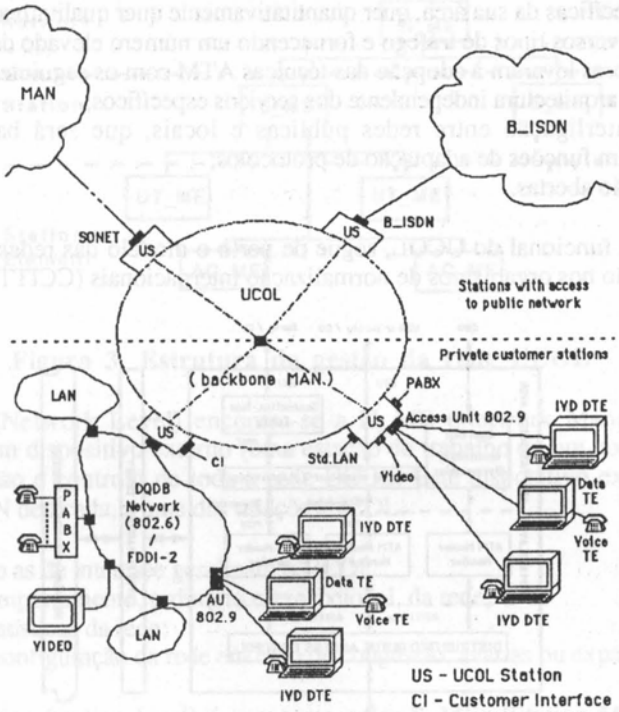


Figura 1. Possíveis aplicações da rede UCOL

Um aspecto essencial é a adesão da arquitectura da rede ao modelo de referência OSI (Open System Interconnection), garantindo assim a conectividade com as redes existentes e futuras através do uso de serviços de gestão e protocolos compatíveis com o modelo OSI.

Outro objectivo importante do conceito de sistema é manter a rede aberta a qualquer futura evolução, através da sua capacidade de auto-reconfiguração sem mudanças físicas, demonstrando assim a principal vantagem do sistema multicanal.

Esta rede deverá ser altamente fiável e capaz duma eficiente interconexão com as LANs existentes e com as redes públicas, logo deverá estar de acordo com a arquitectura geral das redes integradas de serviços.

A exploração dos resultados terá lugar em duas fases e escalas diferentes. Numa primeira fase, os componentes e subsistemas desenvolvidos no projecto darão lugar a protótipos que poderão ser usados num grande número de potenciais produtos, diferentes nas caracteristicas e aplicações. Numa fase posterior, todo o conceito de sistema desenvolvido poderá encontrar uma aceitação comercial conjuntamente com a tecnologia associada.

3. Arquitectura da Rede UCOL

As linhas principais na definição da arquitectura foram:

- exploração eficiente da capacidade de cada canal óptico através da partilha da largura de banda disponível entre vários utilizadores por meio da técnica TDMA;
- gestão das frequências disponíveis de modo a garantir conectividade total, optimizar o uso da capacidade global do sistema e da razão desempenho de rede / custo;

- definição de uma política de "mundo exterior" de modo a permitir, a cada estação UCOL, satisfazer necessidades específicas da sua área, quer quantitativamente quer qualitativamente, suportando conexões simultâneas de diversos tipos de tráfego e fornecendo um número elevado de ligações físicas.

As considerações anteriores levaram à adopção das técnicas ATM com os seguintes propósitos:

- definição de uma arquitectura independente dos serviços específicos;
- permitir fácil interligação entre redes públicas e locais, que será baseada em técnicas ATM, minimizando assim funções de adaptação de protocolos;
- evitar soluções não abertas.

O modelo de referência funcional do UCOL, segue de perto o modelo das redes de banda larga (B_ISDN), actualmente em discussão nos organismos de normalização internacionais (CCITT) [3].

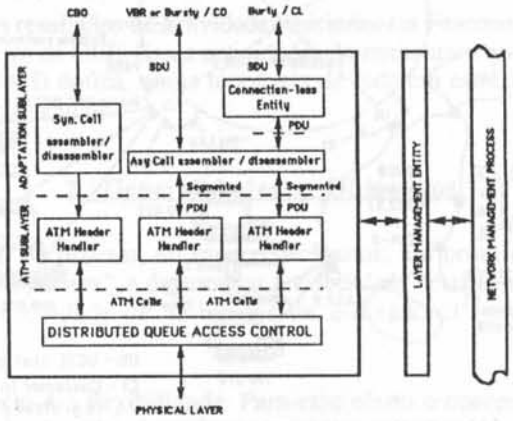


Figura 2. Modelo de referência funcional

A localização das camadas de adaptação (AAL) e de ATM, na estação UCOL, está representada na figura 2.

Três tipos de serviços são oferecidos:

- isócronos, com conexão estabelecida ;
- não isócronos, com conexão estabelecida ;
- não isócronos, sem conexão estabelecida .

Esta estrutura apresenta uma flexibilidade que permitirá satisfazer as necessidades dos serviços actuais e futuros.

4. Gestão da rede UCOL

A gestão da rede UCOL está funcionalmente distribuída ao longo dos três níveis hierárquicos, como é ilustrado na figura 3.

Para garantir a abertura do UCOL para o mundo exterior, a gestão da rede UCOL deverá ser compatível com o modelo OSI, em termos de gestão.

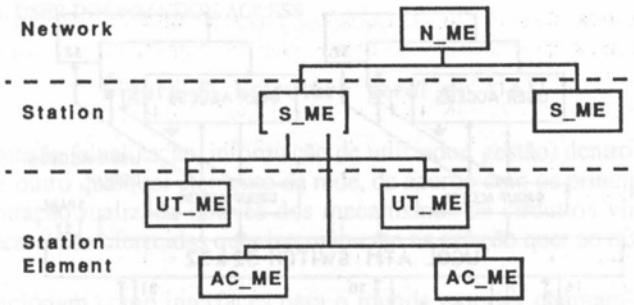


Figura 3. Estrutura de gestão da rede UCOL

No nível mais elevado (Network Level) encontra-se a N_ME (Network Management Entity). A sua funcionalidade residirá num dispositivo externo (uma estação de trabalho ou um computador pessoal) e será responsável pela supervisão e controlo de toda a rede UCOL. Este dispositivo externo estará fisicamente ligado, através de uma LAN dedicada, a uma das estações UCOL.

As funções da N_ME serão as de interesse geral para o UCOL:

- gestão do comportamento, ordinário e excepcional, da rede;
- colher as estatísticas da rede;
- gestão da reconfiguração da rede em caso de congestão, avarias ou expansão "on line".

O segundo nível hierárquico (station level) é composto pelas S_MEs (Station Management Entities). As S_MEs são responsáveis, através da N_ME, pela gestão de todos os recursos da estação, pela optimização do seu comportamento, assim como pelo encaminhamento do tráfego através da rede óptica (escolha do canal no qual a conexão deve ser estabelecida ou onde o tráfego assíncrono deve ser colocado). Tal como para o seu "pai" comum N_ME, também as S_MEs realizarão as mesmas funções. Comportamento semelhante existirá ainda no nível seguinte. A diferença básica entre as entidades está no contexto de operação onde elas são chamados a supervisionar e controlar.

O requisito de compatibilidade com o modelo OSI de gestão afecta apenas o comportamento externo do UCOL, o que não acontece com a arquitectura interna de gestão. No entanto, serviços comuns de informação de gestão (CMIS) e protocolos comuns de informação de gestão (CMIP) serão os instrumentos para a troca de informação de gestão mesmo no interior do UCOL.

5. Arquitectura das Estações

A arquitectura da estação UCOL é ilustrada na figura 4. Nela podem ser identificadas três níveis diferentes. A parte superior representa a interface UCOL - "Mundo Exterior". Cada equipamento terminal de utilizador acede à rede através de uma interface dedicada chamada "User Access Unit" (UAU).

Na parte inferior está representada a interface da estação com a estrela óptica. Existe uma Unidade de Controle de Acesso (ACU), que implementa o protocolo DQAC (Distributed Queue Access Control), para cada interface de rede. O protocolo DQAC será descrito na secção 6.

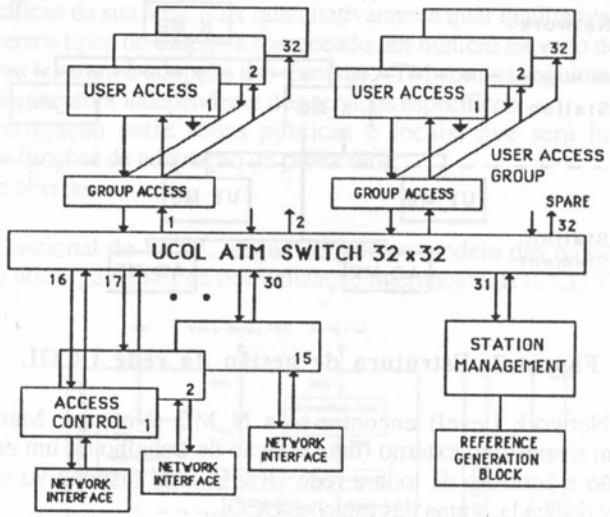
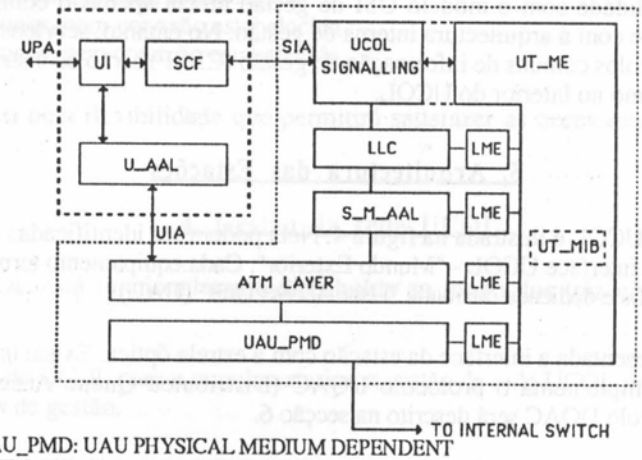


Figura 4. Arquitectura da estação Ucol

A parte intermédia representa o sistema interno de comutação que é baseado num comutador ATM. Os aspectos chave examinados durante a definição preliminar da arquitectura foram:

- capacidade de comutação em termos da probabilidade de bloqueio, atrasos introduzidos, capacidade de impor prioridades, número de conexões simultâneas, suporte de "Broadcast" e "Multicast";
- modularidade;
- fácil adaptatibilidade aos cenários emergentes para redes públicas e privadas.

A arquitectura proposta assenta no conceito básico de que as funções de comutação são realizadas com base no encaminhamento das células ATM.



UAU_PMD: UAU PHYSICAL MEDIUM DEPENDENT
LME: LAYER MANAGEMENT ENTITY
UT_MIB: UT MANAGEMENT INFORMATION BASE
UT_ME: UT MANAGEMENT ENTITY
S_M_AAL: SIGNALLING & MANAGEMENT AAL
UI: USER INTERFACE
U_AAL: USER ATM ADAPTATION LAYER
SIA: SIGNALLING INFORMATION ACCESS

SCF: SIGNALLING CONVERGENCE FUNCTIONS
UIA: USER INFORMATION ACCESS

Figura 5. Estrutura geral do UAU

Isto significa que a informação (sinalização, informação de utilizador, gestão) dentro do UCOL é trocada, entre uma unidade da estação e outro qualquer elemento da rede, de acordo com os principios do ATM: células com comprimento fixo, comutação realizada através dos mecanismos de circuitos virtuais. As capacidades de "Multicast" ou de "Broadcast" são oferecidas quer internamente na estação quer ao nível da rede.

Os sub-sistemas que funcionam como interfaces para o mundo exterior chamam-se UAUs (UCOL Access Units). Cada UAU (figura 5) é funcionalmente dividido em dois grandes blocos: o UT (UCOL Termination) e o TA (Terminal Adapter). A razão desta divisão é manter uma separação lógica das funções integradas no ambiente UCOL das relativas ao mundo exterior.

O TA é composto pelos blocos UI (User Interface), U_AAL (User_ATM Adaptation Layer) e SCF (Signalling Convergence Functions), suportando três funções básicas:

- terminação dos protocolos do equipamento terminal com a separação da informação de controle da informação útil de utilizador;
- adaptação da informação de controle para a forma requerida pela entidade de sinalização do UCOL;
- adaptação da informação útil de utilizador ao formato requerido pelo nível ATM.

A função da UI é, essencialmente, separar o tráfego em dados e sinalização. Os dados seguirão depois através da camada de U_AAL, chegando ao comutador ATM, enquanto que a sinalização cruzará o SCF, onde se tornará "compatível" para os formatos de sinalização do UCOL.

A informação de sinalização passará do TA para o UT através da interface SIA (Signalling Information Access), enquanto os dados chegarão ao UT através da interface UIA (User Information Access).

Uma entidade importante dentro do UT é o UCOL Signalling. Esta entidade recebe do TA a sinalização em formato UCOL, vinda do mundo exterior, usando-a de modo a implementar internamente, através de mensagens e primitivas, os vários processos de controle (e.g. estabelecimento, renegociação de parâmetros e fim de chamada).

Os processos de sinalização presentes no UCOL Signalling comunicam com o bloco de gestão local (UT_ME), interactuando este de seguida com a entidade de gestão da estação (S_ME). De facto, no sentido de estabelecer uma ligação entre dois pontos, o processo de sinalização necessita de informações respeitantes a:

- canais comuns entre pontos chamadores e chamados;
- canal a ser usado pela sinalização.

Estas informações serão facultadas pelo sistema de gestão e despoletarão as seguintes acções:

- atribuição dos VCI / VPI para as células que transportam a informação de sinalização;
- atribuição da largura de banda requerida;
- atribuição dos VCI / VPI para transportar a informação de data do utilizador.

6. Protocolo de Acesso à Rede

O protocolo que regula o acesso ao centro da estrela na rede UCOL ("Distributed Queue Control Protocol - DQAC") baseia-se no princípio de que as estações efectuam periodicamente pedidos de acesso à rede de acordo com o comprimento das suas filas de espera.

SCF: SIGNALLING CONVERGENCE FUNCTIONS
UIA: USER INFORMATION ACCESS

Figura 5. Estrutura geral do UAU

Isto significa que a informação (sinalização, informação de utilizador, gestão) dentro do UCOL é trocada, entre uma unidade da estação e outro qualquer elemento da rede, de acordo com os principios do ATM: células com comprimento fixo, comutação realizada através dos mecanismos de circuitos virtuais. As capacidades de "Multicast" ou de "Broadcast" são oferecidas quer internamente na estação quer ao nível da rede.

Os sub-sistemas que funcionam como interfaces para o mundo exterior chamam-se UAUs (UCOL Access Units). Cada UAU (figura 5) é funcionalmente dividido em dois grandes blocos: o UT (UCOL Termination) e o TA (Terminal Adapter). A razão desta divisão é manter uma separação lógica das funções integradas no ambiente UCOL das relativas ao mundo exterior.

O TA é composto pelos blocos UI (User Interface), U_AAL (User_ATM Adaptation Layer) e SCF (Signalling Convergence Functions), suportando três funções básicas:

- terminação dos protocolos do equipamento terminal com a separação da informação de controle da informação útil de utilizador;
- adaptação da informação de controle para a forma requerida pela entidade de sinalização do UCOL;
- adaptação da informação útil de utilizador ao formato requerido pelo nível ATM.

A função da UI é, essencialmente, separar o tráfego em dados e sinalização. Os dados seguirão depois através da camada de U_AAL, chegando ao comutador ATM, enquanto que a sinalização cruzará o SCF, onde se tornará "compatível" para os formatos de sinalização do UCOL.

A informação de sinalização passará do TA para o UT através da interface SIA (Signalling Information Access), enquanto os dados chegarão ao UT através da interface UIA (User Information Access).

Uma entidade importante dentro do UT é o UCOL Signalling. Esta entidade recebe do TA a sinalização em formato UCOL, vinda do mundo exterior, usando-a de modo a implementar internamente, através de mensagens e primitivas, os vários processos de controle (e.g. estabelecimento, renegociação de parâmetros e fim de chamada).

Os processos de sinalização presentes no UCOL Signalling comunicam com o bloco de gestão local (UT_ME), interactuando este de seguida com a entidade de gestão da estação (S_ME). De facto, no sentido de estabelecer uma ligação entre dois pontos, o processo de sinalização necessita de informações respeitantes a:

- canais comuns entre pontos chamadores e chamados;
- canal a ser usado pela sinalização.

Estas informações serão facultadas pelo sistema de gestão e despoletarão as seguintes acções:

- atribuição dos VCI / VPI para as células que transportam a informação de sinalização;
- atribuição da largura de banda requerida;
- atribuição dos VCI / VPI para transportar a informação de data do utilizador.

6. Protocolo de Acesso à Rede

O protocolo que regula o acesso ao centro da estrela na rede UCOL ("Distributed Queue Control Protocol - DQAC") baseia-se no princípio de que as estações efectuam periodicamente pedidos de acesso à rede de acordo com o comprimento das suas filas de espera.

Os pedidos, que são recebidos por todas as estações, são processados de uma forma distribuída por todas elas antes de efectuarem o acesso físico.

Este protocolo, que regula o acesso à estrela num só canal, baseia-se nos seguintes princípios:

a) Eficiente multiplexagem no centro da estrela.

Para que a multiplexagem da informação no centro da estrela seja óptima, é necessário que os pacotes de dados provenientes de várias estações sejam separados por intervalos de tempo considerados desprezáveis em relação ao comprimento desses pacotes. Reduzindo o tamanho destes intervalos, a eficiência em termos da exploração da banda existente é aumentada. Para que esta condição seja satisfeita é necessário que cada estação conheça o instante em que os seus dados devem atingir o centro da estrela.

Esta condição obriga a que, para todas as estações, sejam satisfeitos os seguintes requisitos:

- distribuição periódica de uma referência temporal;
- conhecimento prévio do instante de transmissão num ciclo;
- antecipação da transmissão dos dados por um período de tempo igual ao atraso de propagação na fibra.

O primeiro requisito é satisfeito pela utilização de um esquema de transmissão baseado em tramas: o sinal de sincronismo de trama é usado como a referência temporal esperada vinda do centro da estrela.

O segundo requisito é satisfeito pela estrutura usada para a trama, que fornece às estações a informação de transmissão necessária.

Para satisfazer o terceiro requisito, todas as estações deverão medir a sua distância ao centro da estrela, durante a fase de inicialização da rede.

b) Transmissão baseada em tramas

A trama definida para o DQAC (figura 6), com uma duração de 1ms, é constituida por três campos: "Frame Header", "Queue Status" e "Information".

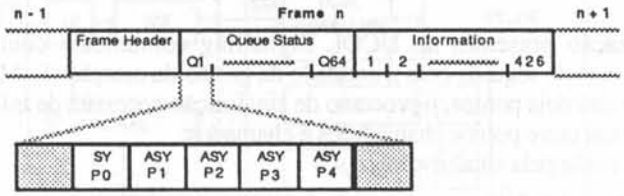


Figura 6. Estrutura da Trama

O campo "Information", usado para a transmissão de informação, é constituido por 426 subcampos destinados ao transporte de células ATM.

O campo "QS" é composto por 64 subcampos, um para cada estação de acordo com a ordem de transmissão estabelecida: o primeiro subcampo é para a estação 1, o segundo para a estação 2, e etc. Esta ordem implícita satisfaz o primeiro requisito imposto no ponto anterior.

O "QS" fornece a informação necessária para a atribuição dos slots de informação que as estações deverão utilizar para a transmissão de informação.

A informação veículada no "QS" de uma trama **n** é usada para a decisão da ordem de transmissão no campo "Information" na trama **n+1**. Desta forma, quando uma nova trama começa, já todas as estações sabem quais os slots de informacão que lhes estão reservados, podendo assim ser satisfeito o terceiro requisito para o campo de informação.

c) Mecanismo de acesso baseado no "QS"

Cada subcampo de "QS" contém o número de células que cada estação possui em cada fila de prioridades (P0 a P4), prontas para serem transmitidas.

O algoritmo para a reserva de slots implica que cada estação, depois de ter recebido todos os QS da trama **n**, faça a sua própria reserva de slots de informação para a trama **n+1**. Esta reserva é feita tomando em consideração as outras estações.

A prioridade P0 é utilizada para tráfego síncrono, e as restantes para tráfego assíncrono.

7. Conclusões

Redes com débitos de vários Gigabits/s serão, provavelmente, um lugar comum na próxima década.

O uso de óptica coerente, explorando dispositivos optoeléctronicos integrados de ponta, em combinação com uma nova arquitectura de sistema poderá ultrapassar as limitações técnicas das actuais redes ópticas.

A rede UCOL representa a primeira tentativa para definir um sistema de comunicações capaz de oferecer essas capacidades e utilizando óptica coerente, possibilitando a realização de um sistema multicanal em que cada canal tem débitos de transmissão próximos dos de algumas das melhores redes actuais.

Mas não são só a tecnologia e a capacidade de transmissão que fazem do UCOL um sistema promissor. A preocupação de utilizar técnicas ATM e a filosofia dos sistemas abertos, colocam-no em boa posição para ser uma das futuras normas neste campo.

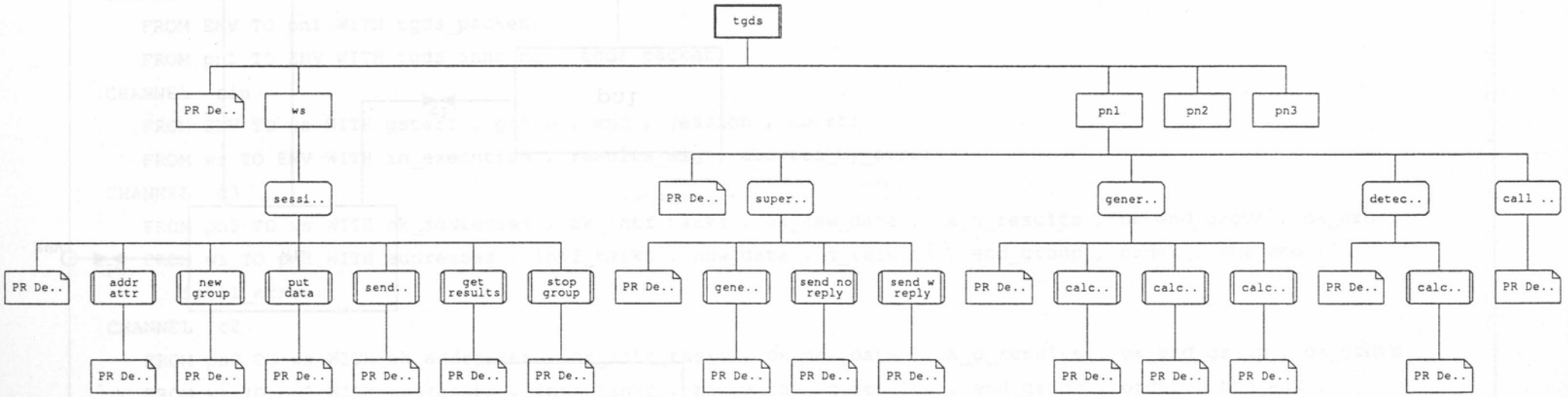
8. Referências

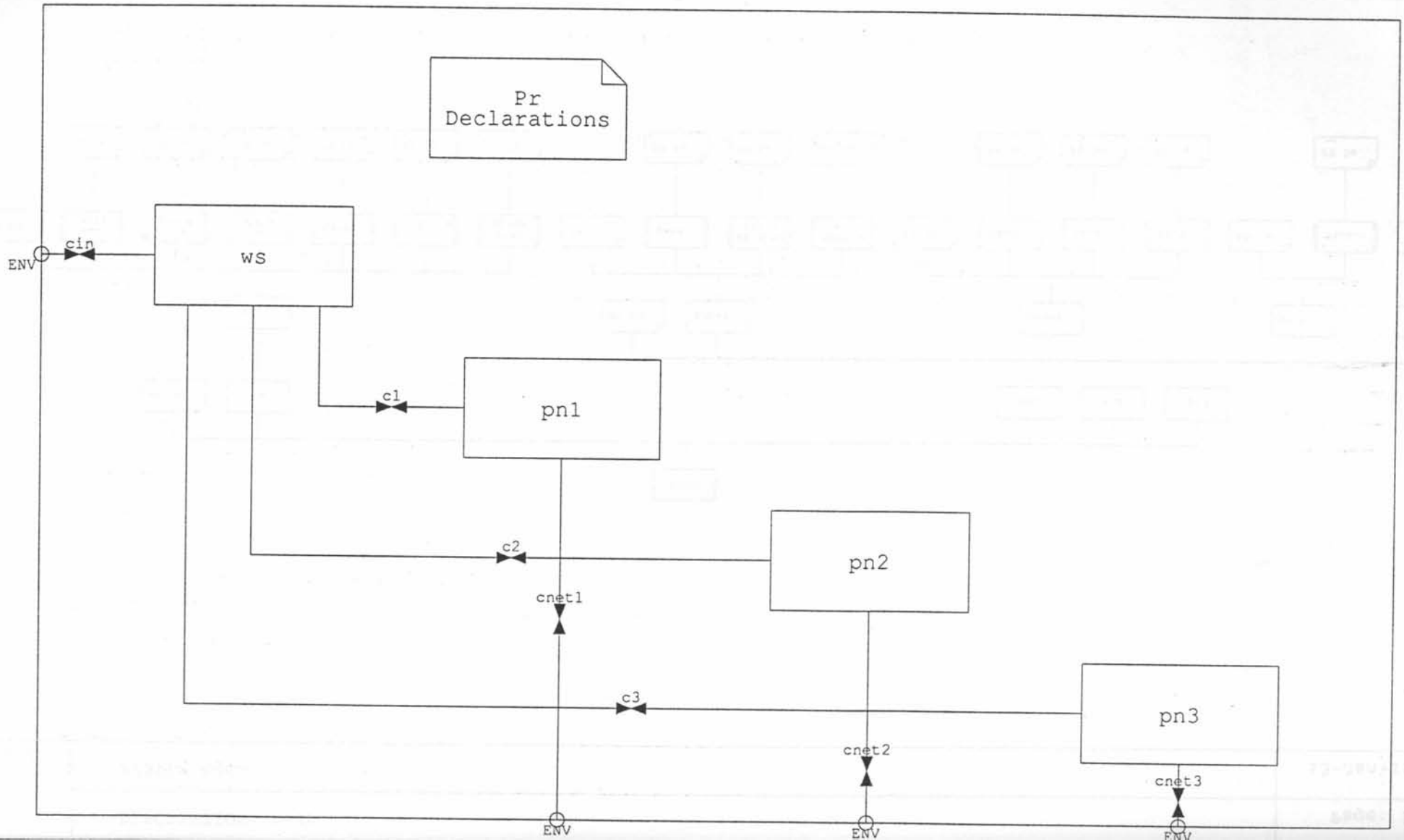
[1] Proposta ESPRIT 5107, "Anexo técnico", 1990.

[2] ESPRIT Project 2054, "Deliverable 905: Report on video interfaces", UCOL/INESC/019/DR/F, 1990-04-26.

[3] Draft Recommendation I.321, "B_ISDN Protocol Reference Model and its applications", CCITT, SGXVIII, Geneva, Maio 1990.

APÊNDICE C - ESPECIFICAÇÃO FORMAL


tgds
PR De..
ws
pn1
pn2
pn3
sessi..
PR De..
super..
gener..
detec..
call ..
PR De..
addr attr
new group
put data
send..
get results
stop group
PR De..
gene..
send no reply
send w reply
PR De..
calc..
calc..
calc..
PR De..
calc..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..
PR De..


Pr
Declarations
ENV
cin
ws
c1
pn1
c2
pn2
cnet1
c3
pn3
cnet2
cnet3
ENV
ENV
ENV

```
CHANNEL  cnet3
   FROM ENV TO pn3 WITH tgds_packet;
   FROM pn3 TO ENV WITH tgds_announc , tgds_packet;
CHANNEL  cnet2
   FROM ENV TO pn2 WITH tgds_packet;
   FROM pn2 TO ENV WITH tgds_announc , tgds_packet;
CHANNEL  cnet1
   FROM ENV TO pn1 WITH tgds_packet;
   FROM pn1 TO ENV WITH tgds_announc , tgds_packet;
CHANNEL  cin
   FROM ENV TO ws WITH gstart , gstop , end , session , abort;
   FROM ws TO ENV WITH in_execution , results_sig , aborted_by_error;
CHANNEL  c3
   FROM pn3 TO ws WITH ok_addresses , ok_intf_tasks , ok_new_data , ok_g_results , ok_end_group , ok_order;
   FROM ws TO pn3 WITH addresses , intf_tasks , new_data , g_results , end_group , order , the_end ,
      fatal_error;
CHANNEL  c2
   FROM pn2 TO ws WITH ok_addresses , ok_intf_tasks , ok_new_data , ok_g_results , ok_end_group , ok_order;
   FROM ws TO pn2 WITH addresses , intf_tasks , new_data , g_results , end_group , order , the_end ,
      fatal_error;
CHANNEL  c1
   FROM pn1 TO ws WITH ok_addresses , ok_intf_tasks , ok_new_data , ok_g_results , ok_end_group , ok_order;
   FROM ws TO pn1 WITH addresses , intf_tasks , new_data , g_results , end_group , order , the_end ,
      fatal_error;
```

```
ENDSYNTYPE STRG_BROAD;
NEWTYPE PN_ADDRESS
    STRUCT
        intf1 NATURAL;
        intf2 NATURAL;
ENDNEWTYPE PN_ADDRESS;
NEWTYPE PN_TASKS
    STRUCT
        intf1 CHAR_TASK;
        intf2 CHAR_TASK;
ENDNEWTYPE PN_TASKS;
/* ^ configuration types ^ */
SYNTYPE NAT_PDF = NATURAL
    CONSTANTS 1: 4
ENDSYNTYPE NAT_PDF;
SYNTYPE NAT_FUNC = NATURAL
    CONSTANTS 1: 9
ENDSYNTYPE NAT_FUNC;
SYNTYPE NAT_INTF = NATURAL
    CONSTANTS 1: 2
ENDSYNTYPE NAT_INTF;
SYNTYPE NAT_PN = NATURAL
    CONSTANTS 1: 3
ENDSYNTYPE NAT_PN;
NEWTYPE PARAMETER
    STRUCT
```

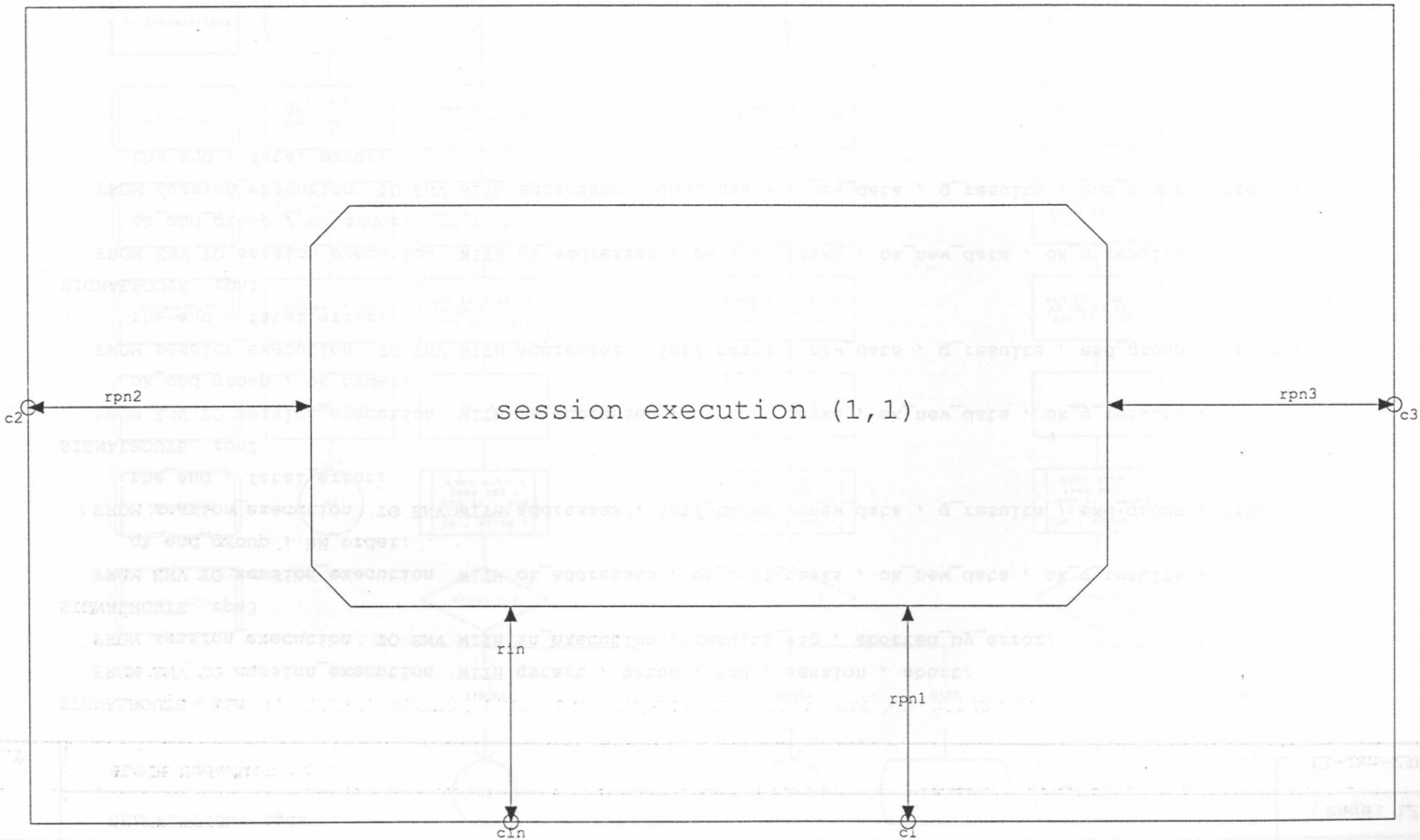
```
        htg3 HISTOGRAM;
        htg4 HISTOGRAM;
ENDNEWTYPE MEAS_DATA;
NEWTYPE INTERFACE_SESSION_DATA
    STRUCT
        taskid CHAR_TASK;
        gen_par GEN_DATA;
        measurements MEAS_DATA;
        src_dst_ieee NATURAL;
ENDNEWTYPE INTERFACE_SESSION_DATA;
NEWTYPE PN_SESSION_DATA
    STRUCT
        intf1 INTERFACE_SESSION_DATA;
        intf2 INTERFACE_SESSION_DATA;
ENDNEWTYPE PN_SESSION_DATA;
/* ^ data types for a new session ^ */
SYNTYPE NAT_HIST = NATURAL
    CONSTANTS 1: 512
ENDSYNTYPE NAT_HIST;
NEWTYPE HIST_VAL
    ARRAY (NAT_HIST, NATURAL);
ENDNEWTYPE HIST_VAL;
NEWTYPE HIST_RES
    STRUCT
        min_val NATURAL;
        int_width NATURAL;
```

```
        n_interval NATURAL;
        max_val NATURAL;
        values HIST_VAL;
ENDNEWTYPE HIST_RES;
NEWTYPE INTERFACE_RESULTS_DATA
    STRUCT
        taskid CHAR_TASK;
        measures NATURAL;
        meas1 NATURAL;
        meas2 NATURAL;
        meas3 NATURAL;
        meas4 NATURAL;
        histograms NATURAL;
        htg1 HIST_RES;
        htg2 HIST_RES;
        htg3 HIST_RES;
        htg4 HIST_RES;
ENDNEWTYPE INTERFACE_RESULTS_DATA;
NEWTYPE PN_RESULTS_DATA
    STRUCT
        intf1 INTERFACE_RESULTS_DATA;
        intf2 INTERFACE_RESULTS_DATA;
ENDNEWTYPE PN_RESULTS_DATA;
/* ^ data types for results of a session ^ */
NEWTYPE ACTIVE_SYS
    ARRAY (NAT_PN, BOOLEAN);
```

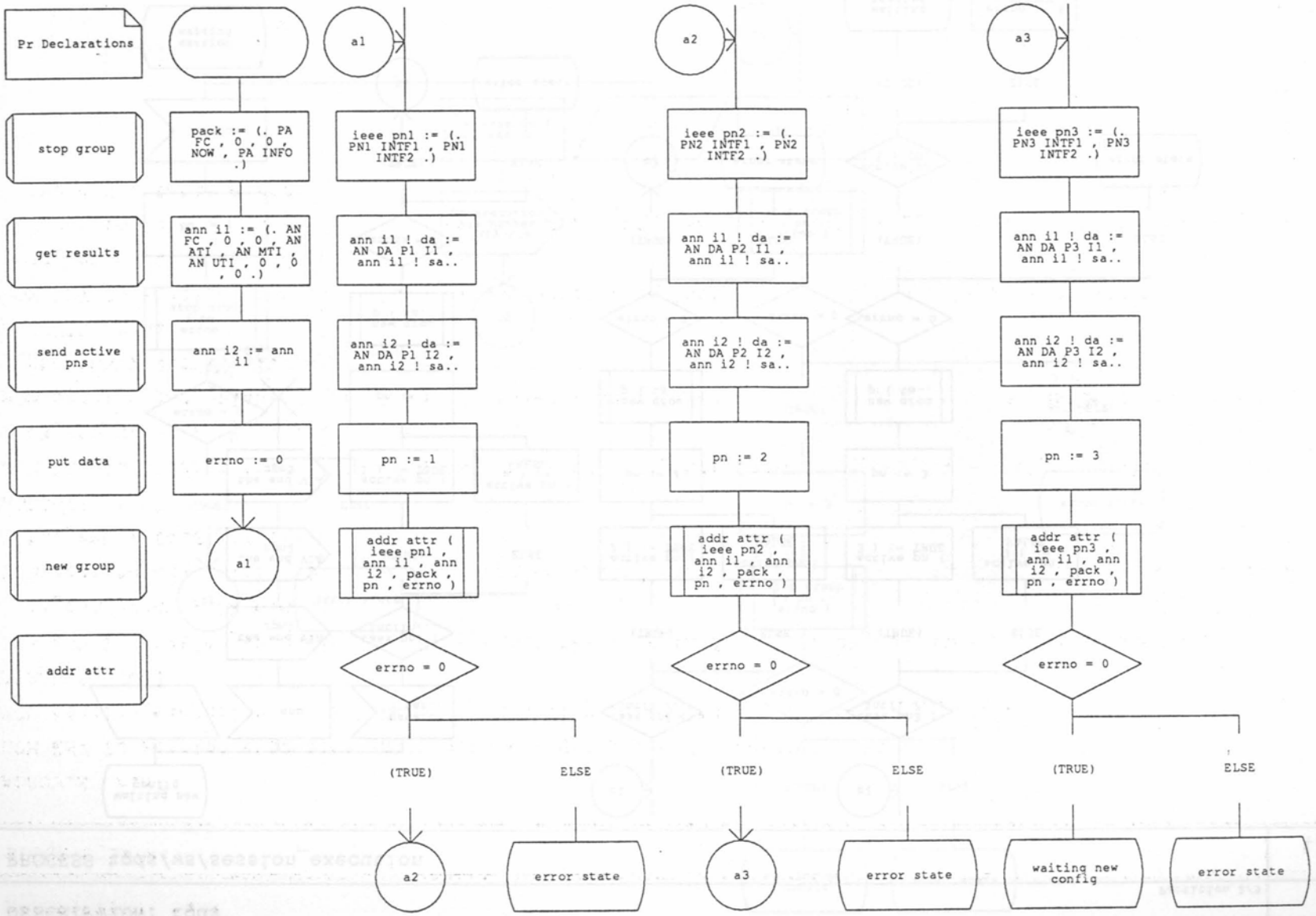
```
ENDNEWTYPE ACTIVE_SYS;
/* ^ system data types ^ */
SIGNAL
    gstart(PN_TASKS, PN_TASKS, PN_TASKS),
    gstop,
    end,
    session(NATURAL, DURATION, PN_SESSION_DATA, PN_SESSION_DATA,
        PN_SESSION_DATA),
    abort,
    in_execution(NATURAL),
    results_sig(NATURAL, PN_RESULTS_DATA, PN_RESULTS_DATA, PN_RESULTS_DATA),
    aborted_by_error(NATURAL);
/* ^ signals exchanged between the WS and the ENV ^ */
SIGNAL
    addresses(NATURAL, NATURAL, ANNOUNC, ANNOUNC, PACKET),
    intf_tasks(CHAR_TASK, CHAR_TASK),
    end_group,
    the_end,
    new_data(INTERFACE_SESSION_DATA, INTERFACE_SESSION_DATA),
    order(STRG_BROAD),
    g_results,
    ok_addresses(NAT_PN, NATURAL),
    ok_intf_tasks(NAT_PN, NATURAL),
    ok_end_group(NAT_PN, NATURAL),
    ok_new_data(NAT_PN, NATURAL),
    ok_order(NAT_PN, NATURAL),
```

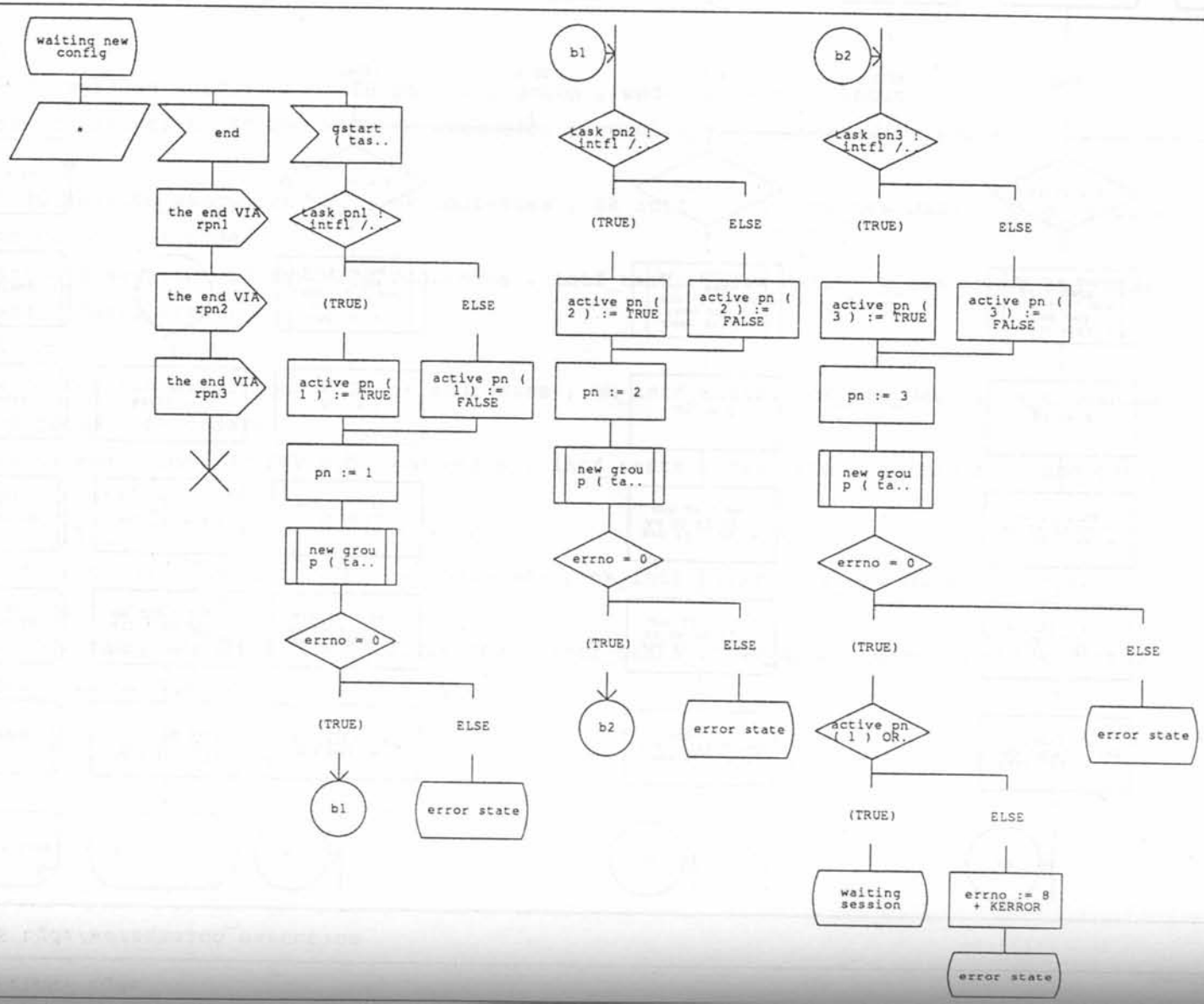
```
    ok_g_results(INTERFACE_RESULTS_DATA, INTERFACE_RESULTS_DATA, NAT_PN, NATURAL),
    fatal_error;
/* ^ signals exchanged between the WS and the PNs ^ */
SIGNAL
    tgds_announc(ANNOUNC),
    tgds_packet(PACKET, NATURAL);
/* ^ signals exchanged between the PNs and the ENV ^ */
```

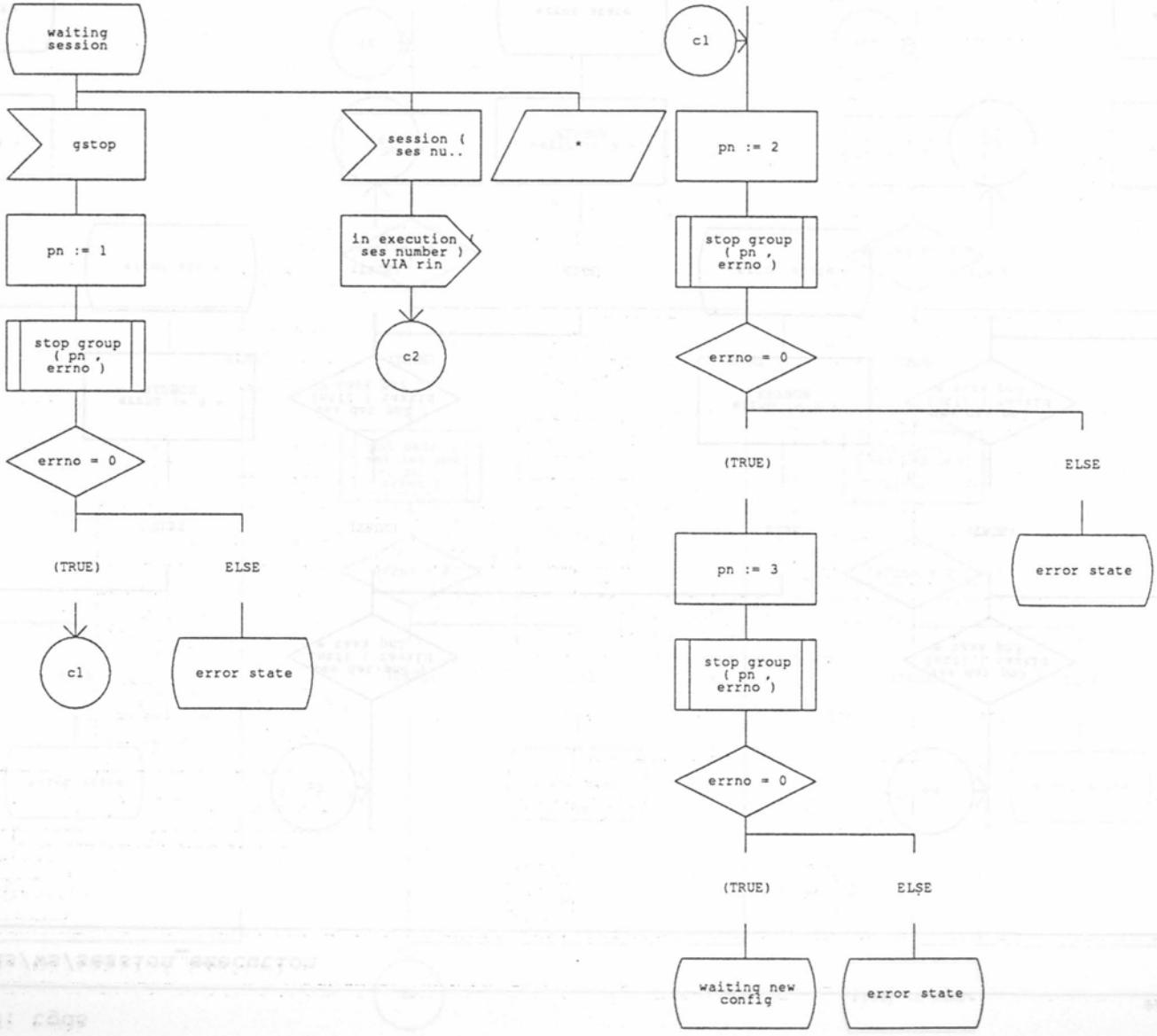
	DESCRIPTION: tgds	Page: 11
1.3	BLOCK tgds/ws	13-Jan-1992

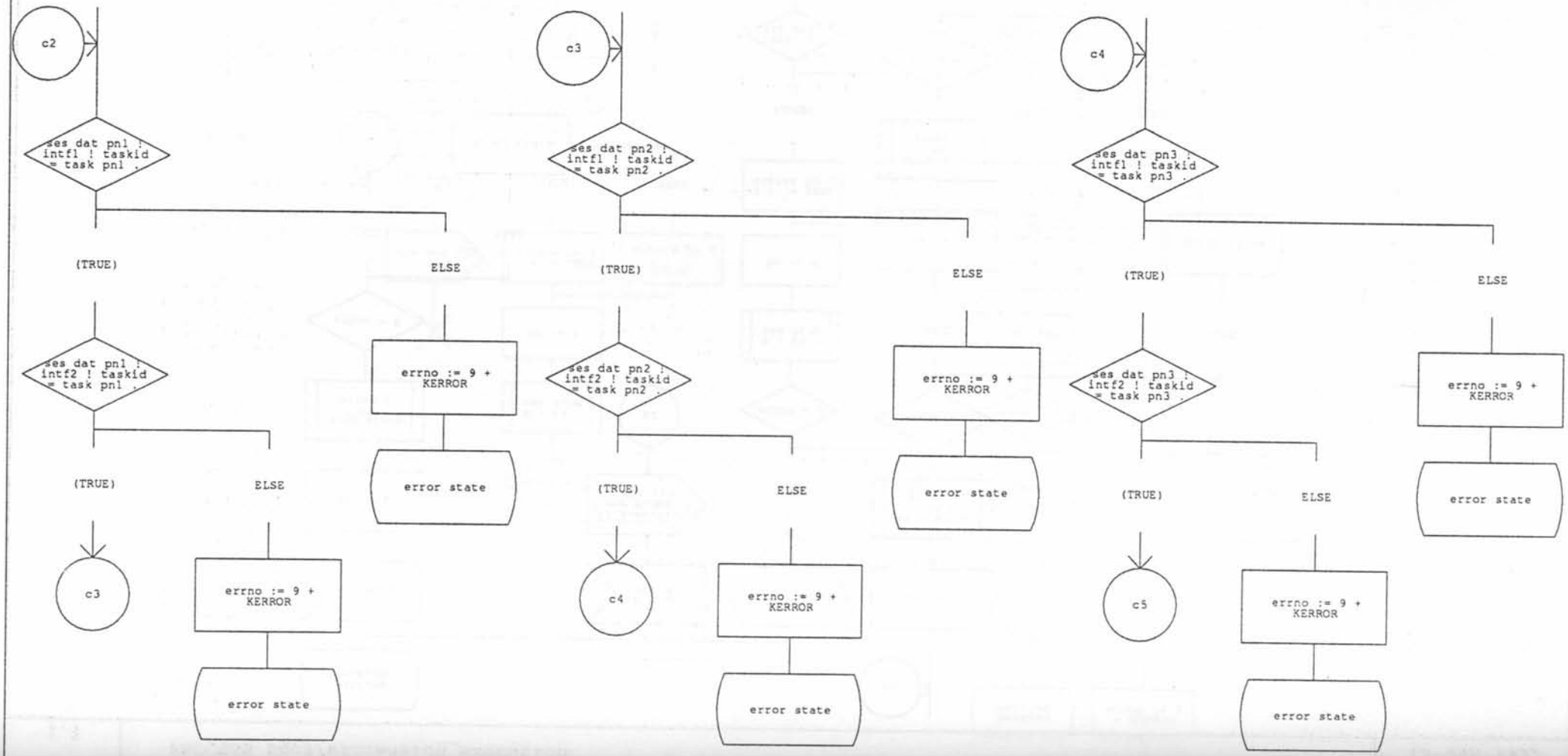


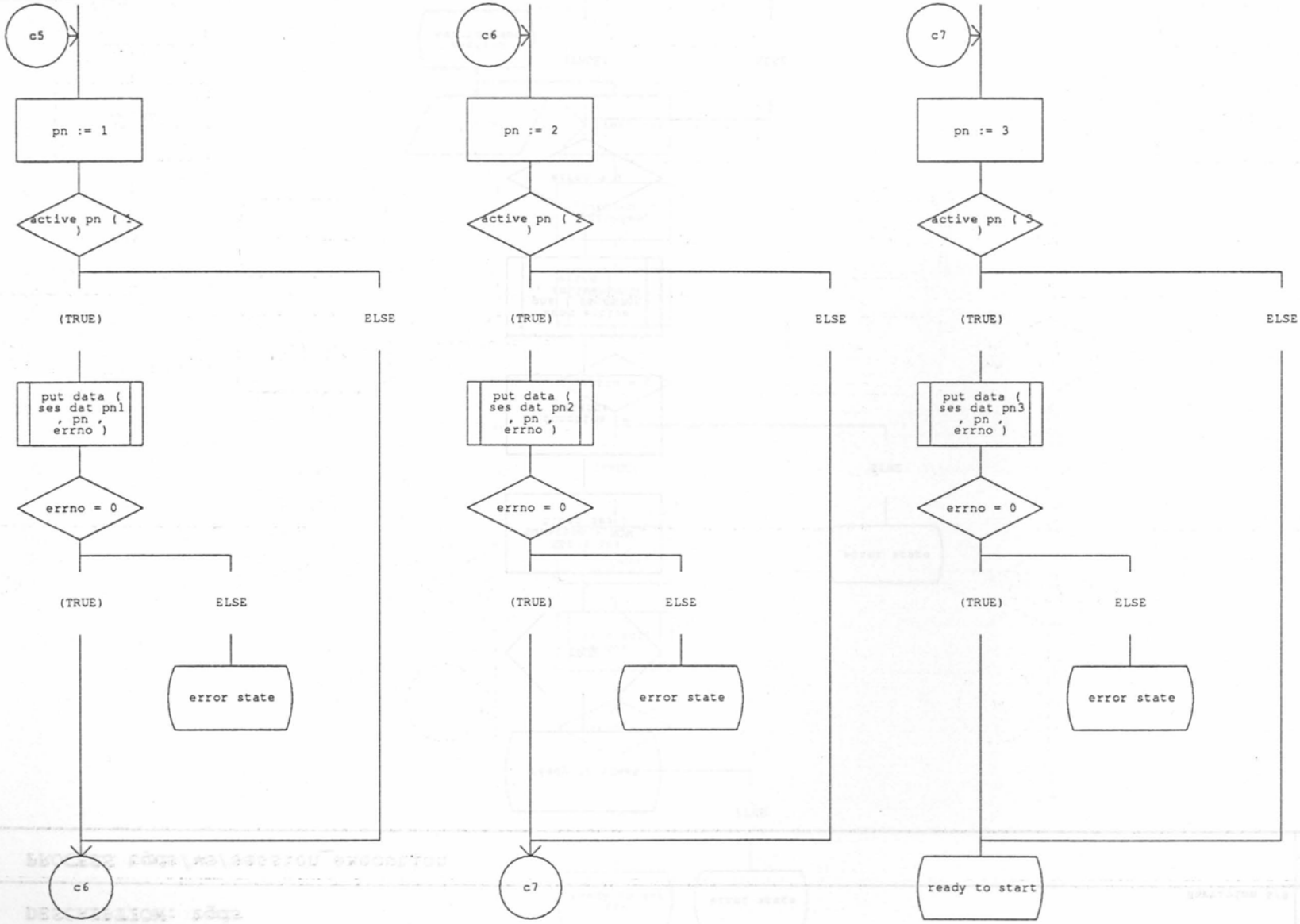
```
SIGNALROUTE  rin
   FROM ENV TO session_execution  WITH gstart , gstop , end , session , abort;
   FROM session_execution  TO ENV WITH in_execution , results_sig , aborted_by_error;
SIGNALROUTE  rpn3
   FROM ENV TO session_execution  WITH ok_addresses , ok_intf_tasks , ok_new_data , ok_g_results ,
      ok_end_group , ok_order;
   FROM session_execution  TO ENV WITH addresses , intf_tasks , new_data , g_results , end_group , order ,
      the_end , fatal_error;
SIGNALROUTE  rpn2
   FROM ENV TO session_execution  WITH ok_addresses , ok_intf_tasks , ok_new_data , ok_g_results ,
      ok_end_group , ok_order;
   FROM session_execution  TO ENV WITH addresses , intf_tasks , new_data , g_results , end_group , order ,
      the_end , fatal_error;
SIGNALROUTE  rpn1
   FROM ENV TO session_execution  WITH ok_addresses , ok_intf_tasks , ok_new_data , ok_g_results ,
      ok_end_group , ok_order;
   FROM session_execution  TO ENV WITH addresses , intf_tasks , new_data , g_results , end_group , order ,
      the_end , fatal_error;
```

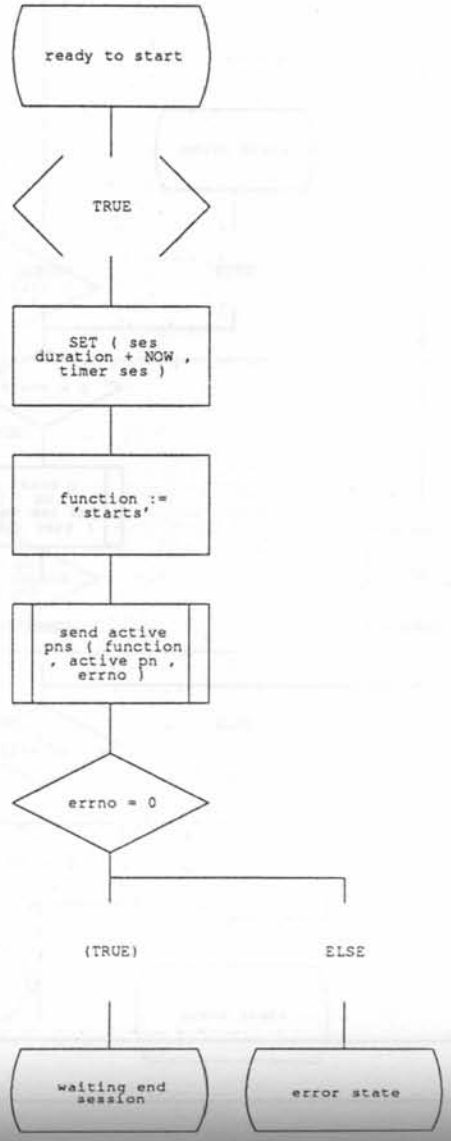

Pr Declarations
stop group
get results
send active pns
put data
new group
addr attr
pack := (. PA FC , 0 , 0 , NOW , PA INFO .)
ann i1 := (. AN FC , 0 , 0 , AN ATI , AN MTI , AN UTI , 0 , 0 , 0 .)
ann i2 := ann i1
errno := 0
a1
a1
ieee pn1 := (. PN1 INTF1 , PN1 INTF2 .)
ann i1 ! da := AN DA P1 I1 , ann i1 ! sa..
ann i2 ! da := AN DA P1 I2 , ann i2 ! sa..
pn := 1
addr attr (ieee pn1 , ann i1 , ann i2 , pack , pn , errno)
errno = 0
(TRUE)
ELSE
a2
error state
a2
ieee pn2 := (. PN2 INTF1 , PN2 INTF2 .)
ann i1 ! da := AN DA P2 I1 , ann i1 ! sa..
ann i2 ! da := AN DA P2 I2 , ann i2 ! sa..
pn := 2
addr attr (ieee pn2 , ann i1 , ann i2 , pack , pn , errno)
errno = 0
(TRUE)
ELSE
a3
error state
a3
ieee pn3 := (. PN3 INTF1 , PN3 INTF2 .)
ann i1 ! da := AN DA P3 I1 , ann i1 ! sa..
ann i2 ! da := AN DA P3 I2 , ann i2 ! sa..
pn := 3
addr attr (ieee pn3 , ann i1 , ann i2 , pack , pn , errno)
errno = 0
(TRUE)
ELSE
waiting new config
error state

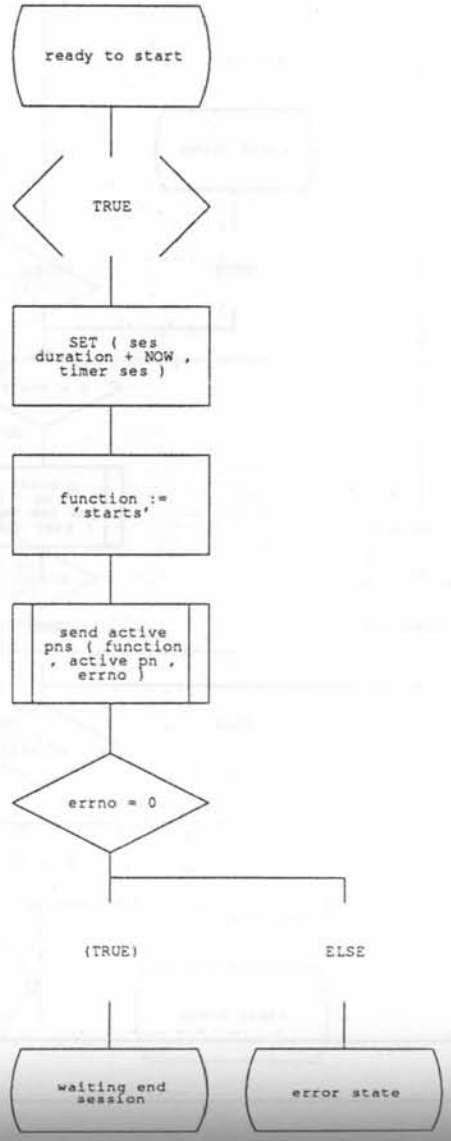

waiting new config
*
end
gstart (tas..
the end VIA rpn1
the end VIA rpn2
the end VIA rpn3
task pn1 ! intfl /..
(TRUE)
ELSE
active pn (1) := TRUE
active pn (1) := FALSE
pn := 1
new grou p (ta..
errno = 0
(TRUE)
ELSE
b1
error state
b1
task pn2 ! intfl /..
(TRUE)
ELSE
active pn (2) := TRUE
active pn (2) := FALSE
pn := 2
new grou p (ta..
errno = 0
(TRUE)
ELSE
b2
error state
b2
task pn3 ! intfl /..
(TRUE)
ELSE
active pn (3) := TRUE
active pn (3) := FALSE
pn := 3
new grou p (ta..
errno = 0
(TRUE)
ELSE
active pn (1) OR..
error state
(TRUE)
ELSE
waiting session
errno := 8 + KERROR
error state

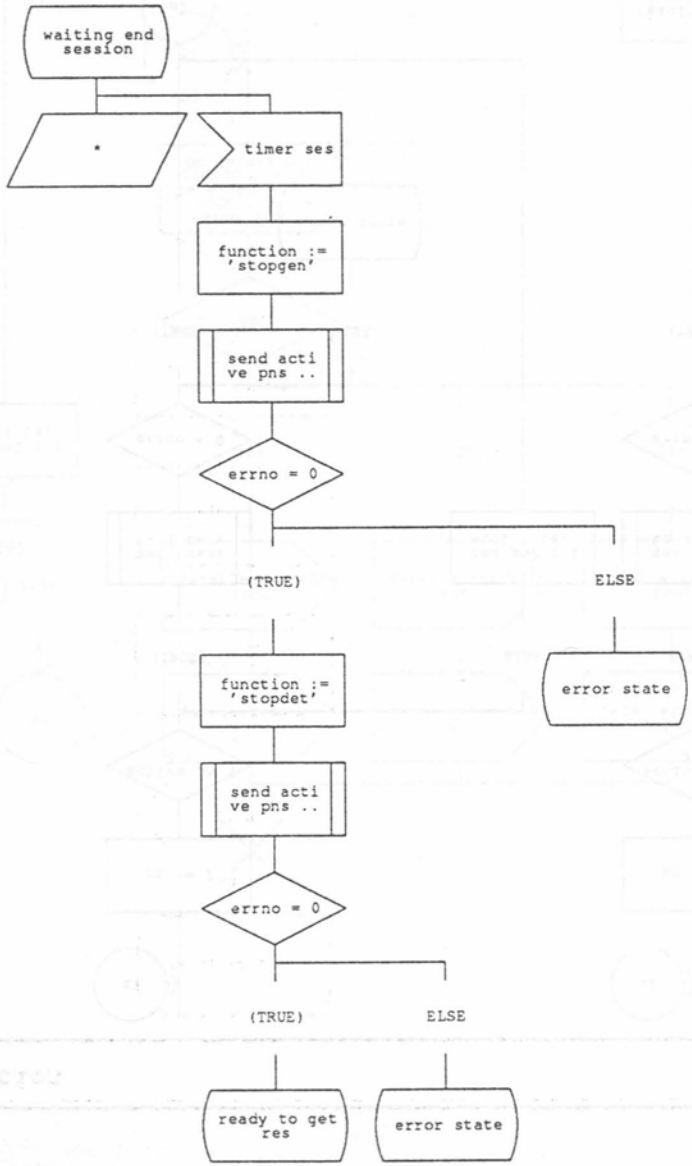

waiting session
gstop
pn := 1
stop group (pn , errno)
errno = 0
(TRUE)
ELSE
c1
error state
session (ses nu..
*
in execution (ses number) VIA rin
c2
c1
pn := 2
stop group (pn , errno)
errno = 0
(TRUE)
ELSE
error state
pn := 3
stop group (pn , errno)
errno = 0
(TRUE)
ELSE
waiting new config
error state

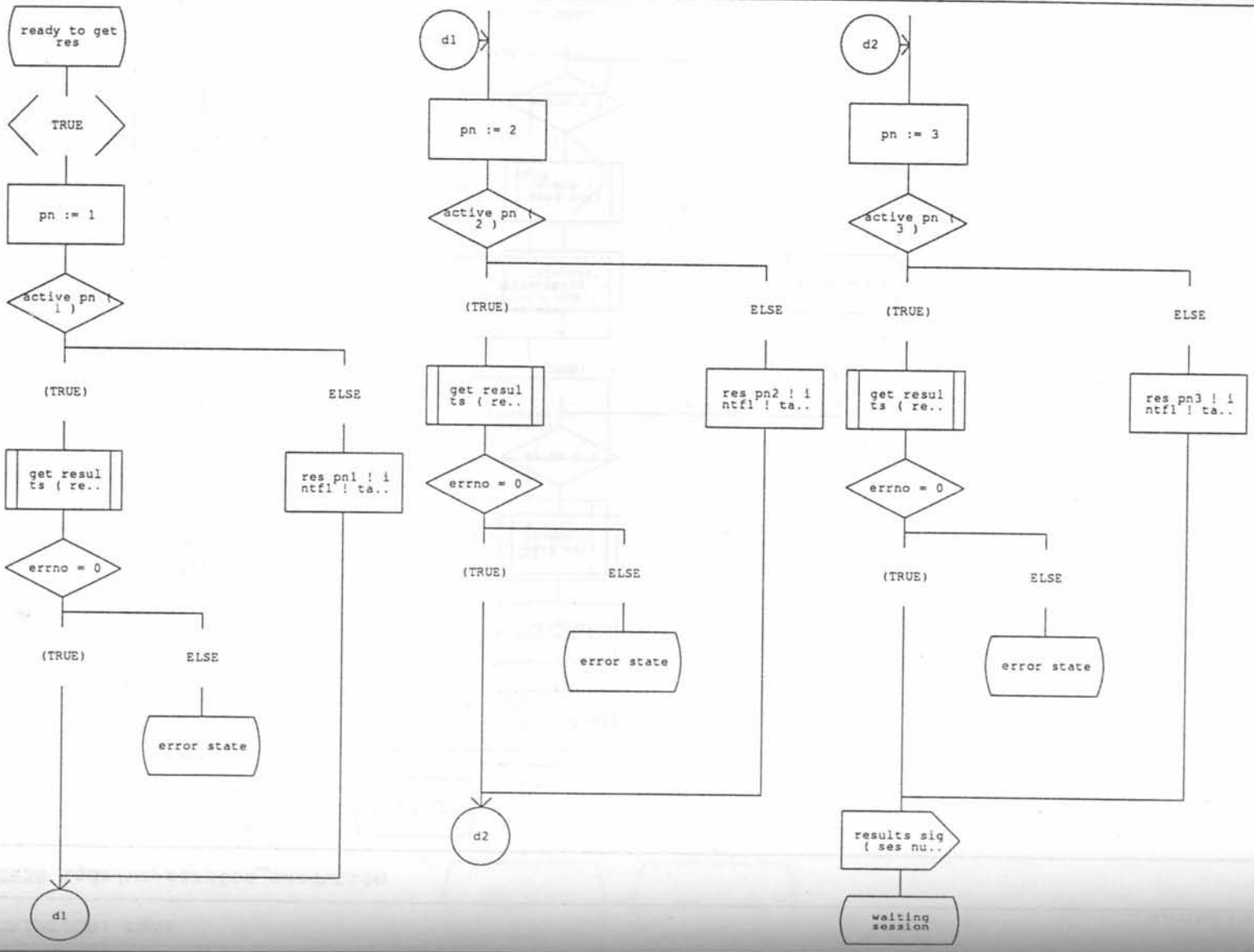

c2
ses dat pn1 ! intf1 ! taskid = task pn1 .
(TRUE)
ELSE
ses dat pn1 ! intf2 ! taskid = task pn1 .
errno := 9 + KERROR
(TRUE)
ELSE
error state
c3
errno := 9 + KERROR
error state
c3
ses dat pn2 ! intf1 ! taskid = task pn2 .
(TRUE)
ELSE
ses dat pn2 ! intf2 ! taskid = task pn2 .
errno := 9 + KERROR
(TRUE)
ELSE
error state
c4
errno := 9 + KERROR
error state
c4
ses dat pn3 ! intf1 ! taskid = task pn3 .
(TRUE)
ELSE
ses dat pn3 ! intf2 ! taskid = task pn3 .
errno := 9 + KERROR
(TRUE)
ELSE
error state
c5
errno := 9 + KERROR
error state

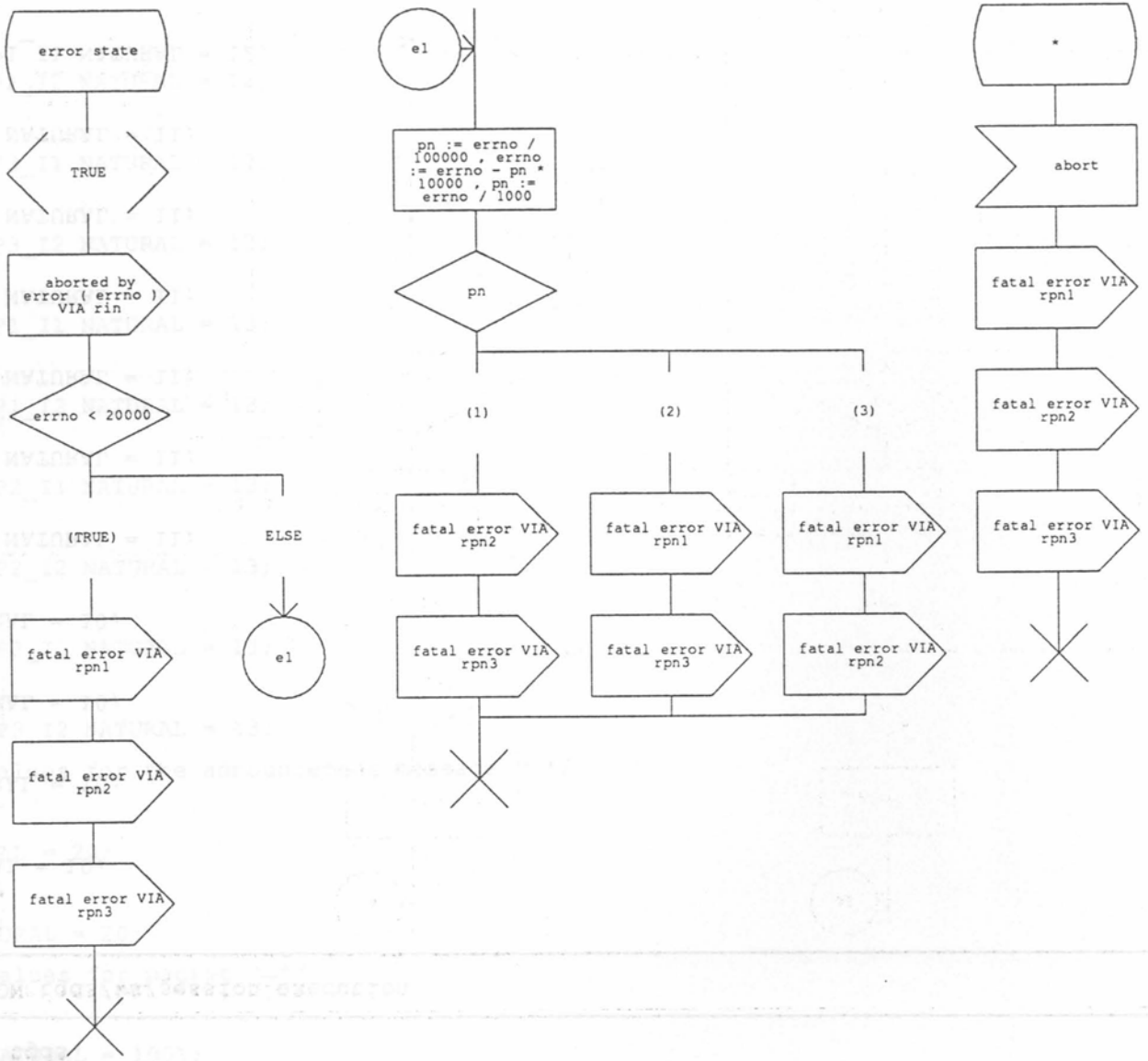

c5
pn := 1
active pn (1)
(TRUE)
ELSE
put data (ses dat pn1 , pn , errno)
errno = 0
(TRUE)
ELSE
error state
c6
c6
pn := 2
active pn (2)
(TRUE)
ELSE
put data (ses dat pn2 , pn , errno)
errno = 0
(TRUE)
ELSE
error state
c7
c7
pn := 3
active pn (3)
(TRUE)
ELSE
put data (ses dat pn3 , pn , errno)
errno = 0
(TRUE)
ELSE
error state
ready to start


ready to start
TRUE
SET (ses
duration + NOW ,
timer ses)
function :=
'starts'
send active
pns (function
, active pn ,
errno)
errno = 0
(TRUE)
ELSE
waiting end
session
error state


ready to start
TRUE
SET (ses
duration + NOW ,
timer ses)
function :=
'starts'
send active
pns (function
, active pn ,
errno)
errno = 0
(TRUE)
ELSE
waiting end
session
error state


waiting end session
*
timer ses
function := 'stopgen'
send acti ve pns ..
errno = 0
(TRUE)
ELSE
function := 'stopdet'
error state
send acti ve pns ..
errno = 0
(TRUE)
ELSE
ready to get res
error state


ready to get res
TRUE
pn := 1
active pn (1)
(TRUE)
ELSE
get resul ts (re..
errno = 0
(TRUE)
ELSE
error state
res pn1 ! i ntf1 ! ta..
d1
d1
pn := 2
active pn (2)
(TRUE)
ELSE
get resul ts (re..
errno = 0
(TRUE)
ELSE
error state
res pn2 ! i ntf1 ! ta..
d2
d2
pn := 3
active pn (3)
(TRUE)
ELSE
get resul ts (re..
errno = 0
(TRUE)
ELSE
error state
res pn3 ! i ntf1 ! ta..
results sig (ses nu..
waiting session

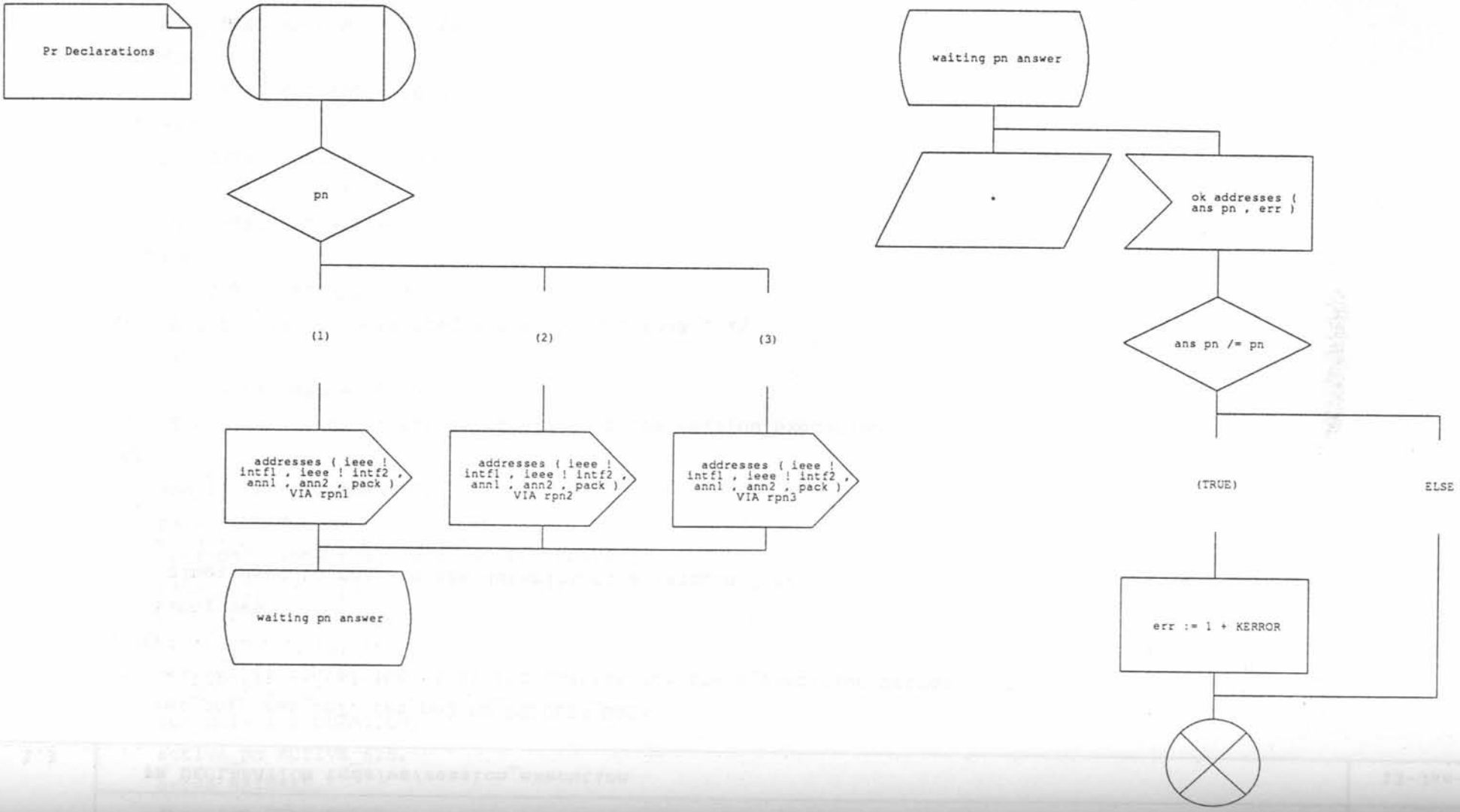

error state
TRUE
aborted by error (errno) VIA rin
errno < 20000
(TRUE)
ELSE
fatal error VIA rpn1
fatal error VIA rpn2
fatal error VIA rpn3
e1
e1
pn := errno / 100000 , errno := errno - pn * 10000 , pn := errno / 1000
pn
(1)
(2)
(3)
fatal error VIA rpn2
fatal error VIA rpn3
fatal error VIA rpn1
fatal error VIA rpn3
fatal error VIA rpn1
fatal error VIA rpn2
*
abort
fatal error VIA rpn1
fatal error VIA rpn2
fatal error VIA rpn3

```
SYNONYM
    AN_FC NATURAL = 10;
SYNONYM
    AN_ATI NATURAL = 10;
SYNONYM
    AN_MTI NATURAL = 10;
SYNONYM
    AN_UTI NATURAL = 10;
SYNONYM
    AN_DA_P1_I1 NATURAL = 11;
SYNONYM
    AN_DA_P1_I2 NATURAL = 11;
SYNONYM
    AN_DA_P2_I1 NATURAL = 11;
SYNONYM
    AN_DA_P2_I2 NATURAL = 11;
SYNONYM
    AN_DA_P3_I1 NATURAL = 11;
SYNONYM
    AN_DA_P3_I2 NATURAL = 11;
SYNONYM
    AN_UMGT_PI_P1_I1 NATURAL = 12;
SYNONYM
    AN_UMGT_PI_P1_I2 NATURAL = 12;
SYNONYM
    AN_UMGT_PI_P2_I1 NATURAL = 12;
```

```
SYNONYM
    AN_UMGT_PI_P2_I2 NATURAL = 12;
SYNONYM
    AN_UMGT_PI_P3_I1 NATURAL = 12;
SYNONYM
    AN_UMGT_PI_P3_I2 NATURAL = 12;
SYNONYM
    AN_NODE_ID_P1_I1 NATURAL = 13;
SYNONYM
    AN_NODE_ID_P1_I2 NATURAL = 13;
SYNONYM
    AN_NODE_ID_P2_I1 NATURAL = 13;
SYNONYM
    AN_NODE_ID_P2_I2 NATURAL = 13;
SYNONYM
    AN_NODE_ID_P3_I1 NATURAL = 13;
SYNONYM
    AN_NODE_ID_P3_I2 NATURAL = 13;
/* ^ constant values for the announcement message ^ */
SYNONYM
    PA_FC NATURAL = 20;
SYNONYM
    PA_INFO NATURAL = 20;
/* ^ constant values for packet ^ */
SYNONYM
    PN1_INTF1 NATURAL = 1001;
```

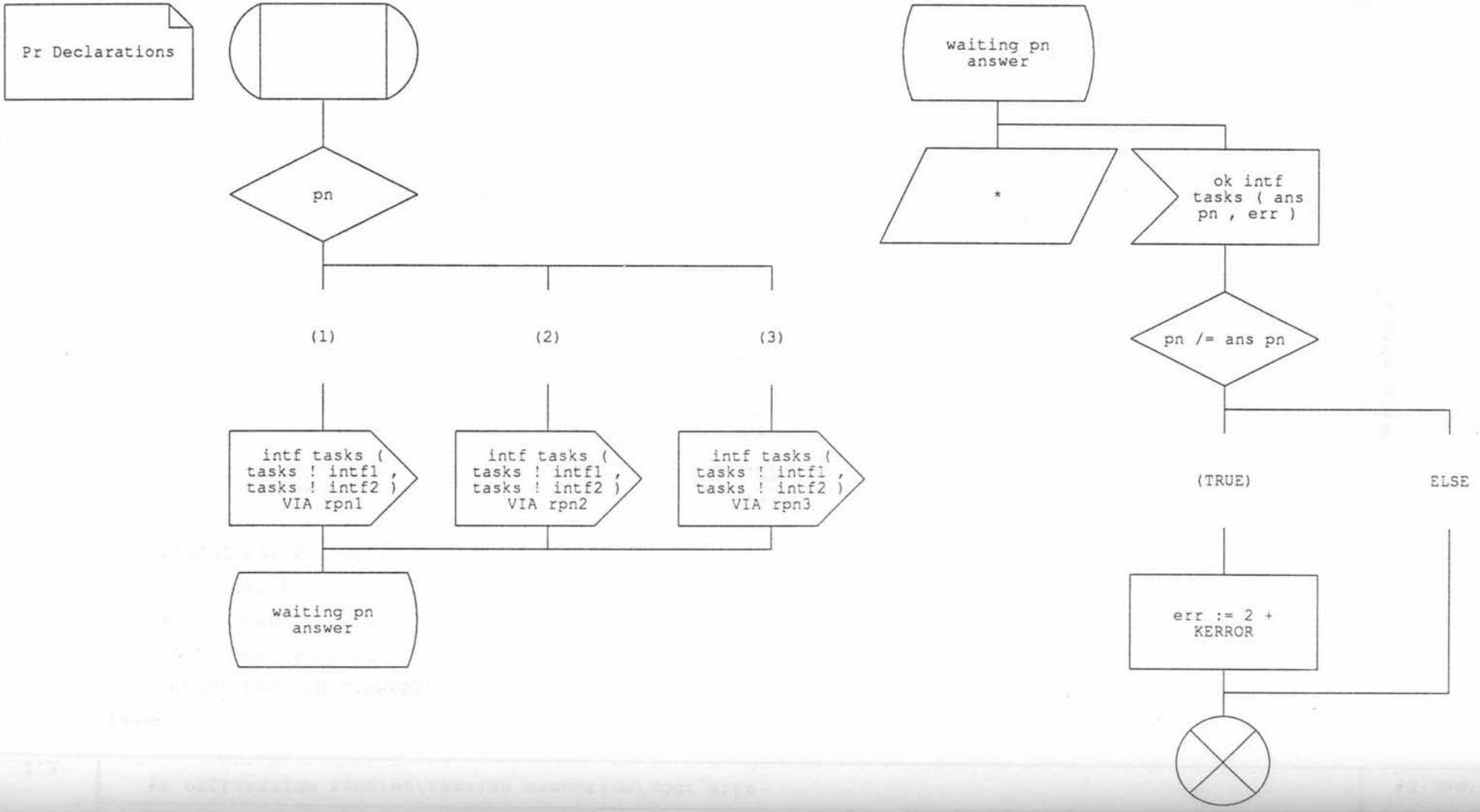
```
SYNONYM
    PN1_INTF2 NATURAL = 1002;
SYNONYM
    PN2_INTF1 NATURAL = 2001;
SYNONYM
    PN2_INTF2 NATURAL = 2002;
SYNONYM
    PN3_INTF1 NATURAL = 3001;
SYNONYM
    PN3_INTF2 NATURAL = 3002;
/* ^ All the values described above are fictious ^ */
SYNONYM
    KERROR NATURAL = 10000;
/* ^ The bias value for errors ocurring at the session_execution ^ */
DCL
    ann_i1, ann_i2 ANNOUNC,
    pack PACKET,
    ieee_pn1, ieee_pn2, ieee_pn3 PN_ADDRESS,
    task_pn1, task_pn2, task_pn3 PN_TASKS,
    ses_dat_pn1, ses_dat_pn2, ses_dat_pn3 PN_SESSION_DATA,
    ses_number NATURAL,
    pn NAT_PN,
    ses_duration DURATION,
    active_pn ACTIVE_SYS,
    errno NATURAL,
    function STRG_BROAD,
```

```
    res_pn1, res_pn2, res_pn3 PN_RESULTS_DATA;
/* ^ variables containing vital information for the s_execution process ^ */
TIMER
    timer_ses;
/* ^ timer used to measure the duration of a session ^ */
```


Pr Declarations
pn
(1)
(2)
(3)
addresses (ieee ! intf1 , ieee ! intf2 , ann1 , ann2 , pack) VIA rpn1
addresses (ieee ! intf1 , ieee ! intf2 , ann1 , ann2 , pack) VIA rpn2
addresses (ieee ! intf1 , ieee ! intf2 , ann1 , ann2 , pack) VIA rpn3
waiting pn answer
waiting pn answer
*
ok addresses (ans pn , err)
ans pn /= pn
(TRUE)
ELSE
err := 1 + KERROR

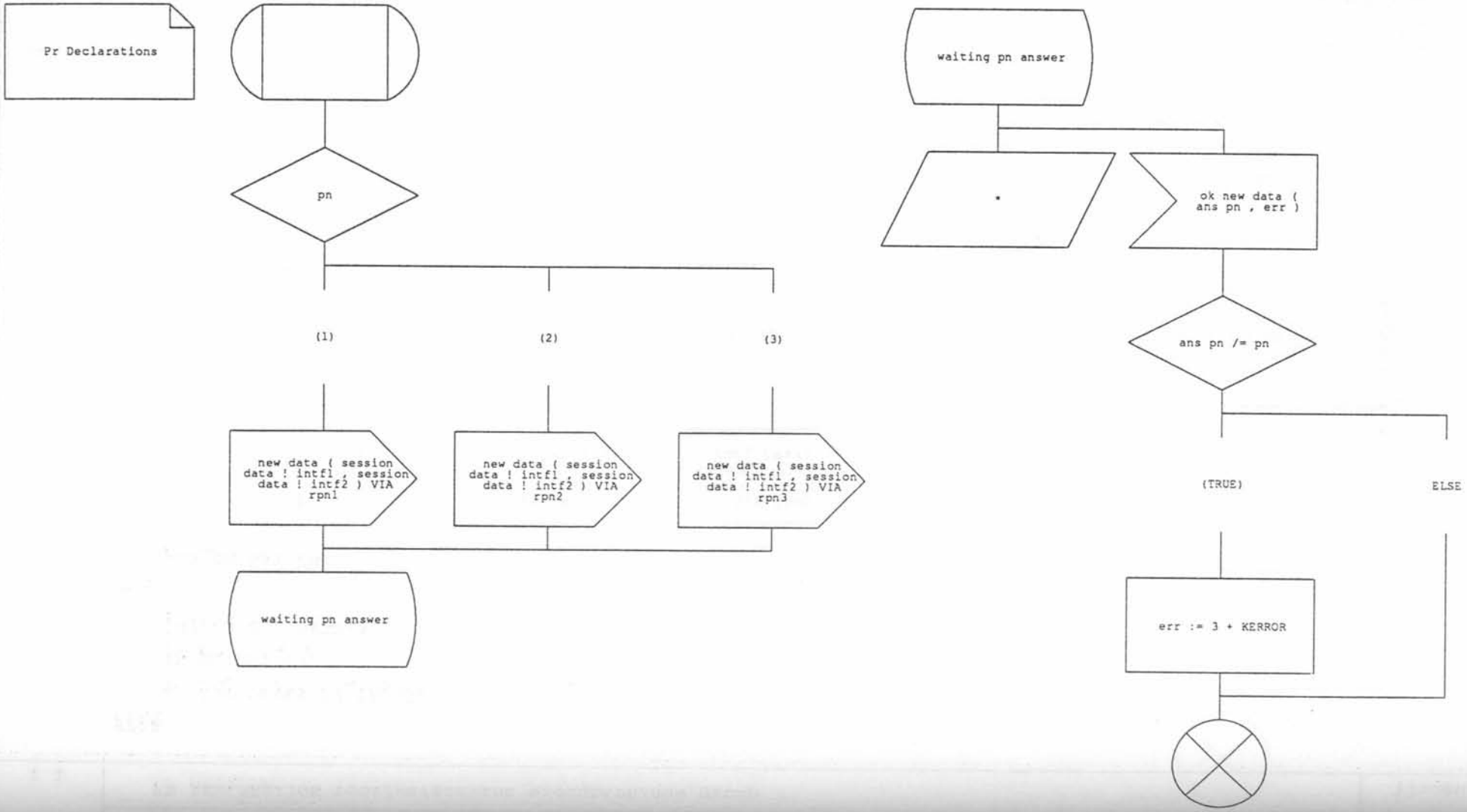
```
FPAR
    IN/OUT ieee PN_ADDRESS,
    ann1, ann2 ANNOUNC,
    pack PACKET,
    IN pn NAT_PN,
    IN/OUT err NATURAL;
DCL
    ans_pn NAT_PN;
```

```
FPAR
    IN/OUT ieee PN_ADDRESS,
    ann1, ann2 ANNOUNC,
    pack PACKET,
    IN pn NAT_PN,
    IN/OUT err NATURAL;
DCL
    ans_pn NAT_PN;
```

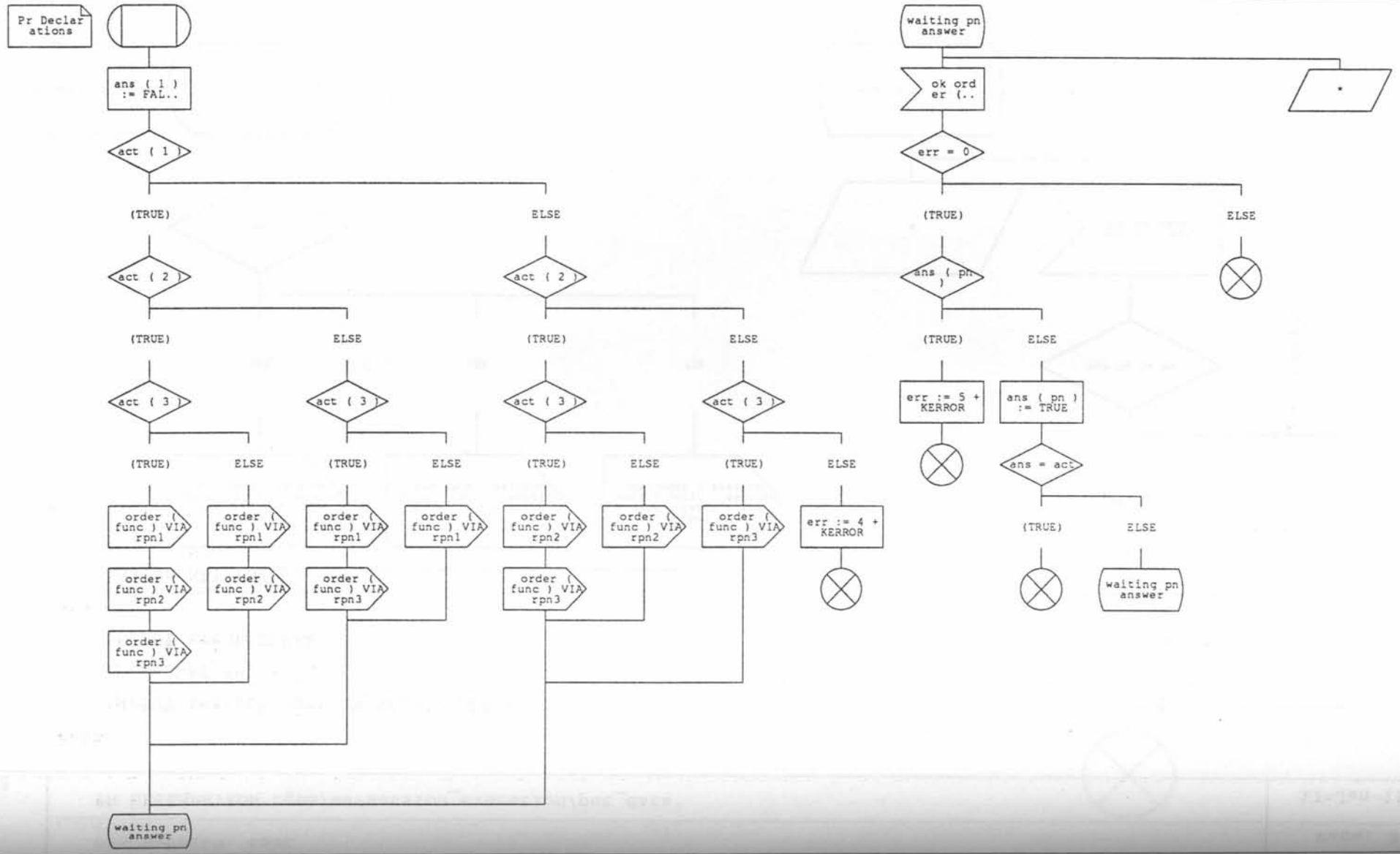

Pr Declarations
pn
(1)
(2)
(3)
intf tasks (tasks ! intf1 , tasks ! intf2) VIA rpn1
intf tasks (tasks ! intf1 , tasks ! intf2) VIA rpn2
intf tasks (tasks ! intf1 , tasks ! intf2) VIA rpn3
waiting pn answer
waiting pn answer
*
ok intf tasks (ans pn , err)
pn /= ans pn
(TRUE)
ELSE
err := 2 + KERROR

```
FPAR
    IN/OUT tasks PN_TASKS,
    IN pn NAT_PN,
    IN/OUT err NATURAL;
DCL
    ans_pn NAT_PN;
```

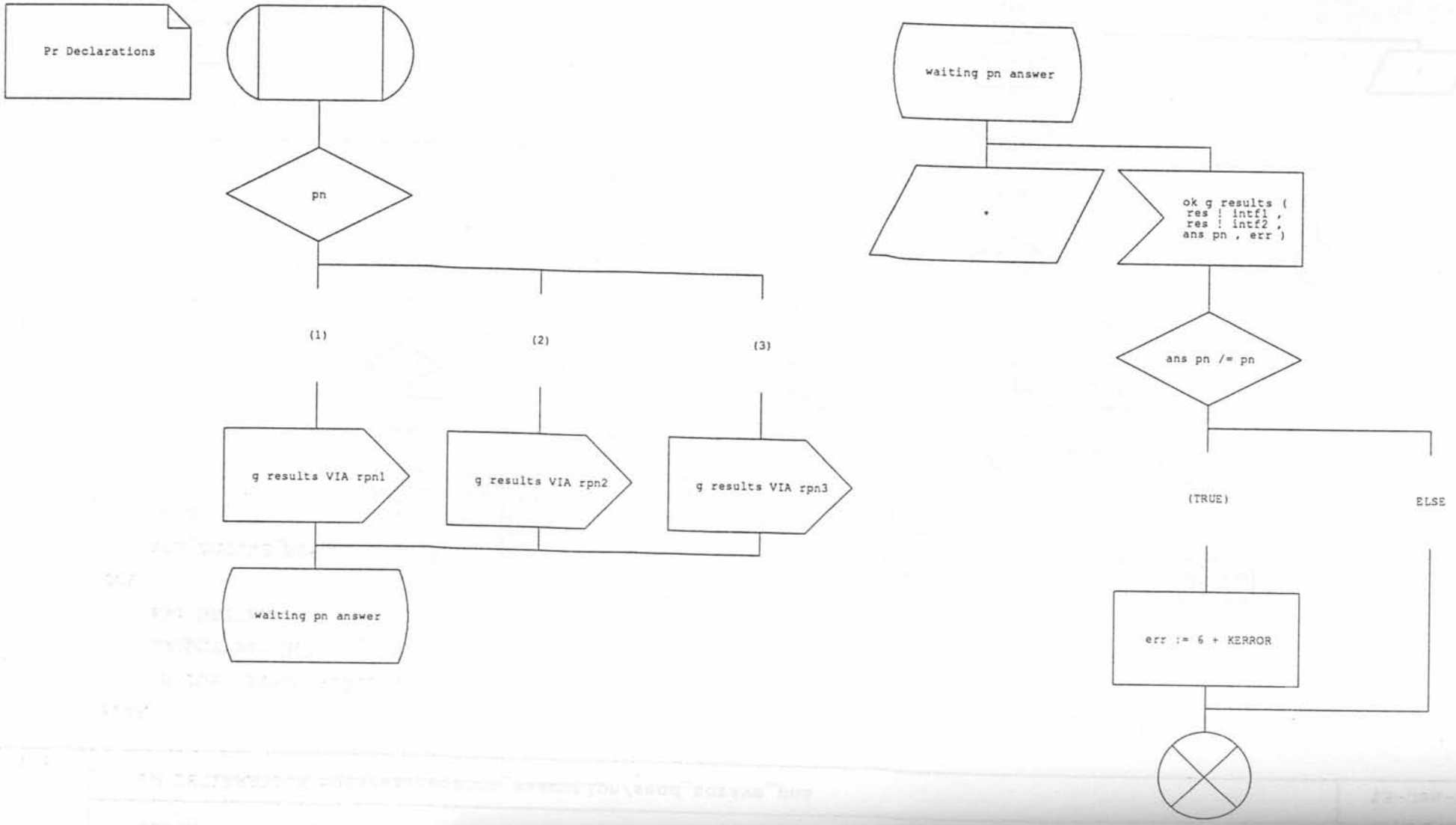
```
FPAR
    IN/OUT tasks PN_TASKS,
    IN pn NAT_PN,
    IN/OUT err NATURAL;
DCL
    ans_pn NAT_PN;
```


Pr Declarations
pn
(1)
(2)
(3)
new data (session data ! intf1 , session data ! intf2) VIA rpn1
new data (session data ! intf1 , session data ! intf2) VIA rpn2
new data (session data ! intf1 , session data ! intf2) VIA rpn3
waiting pn answer
waiting pn answer
*
ok new data (ans pn , err)
ans pn /= pn
(TRUE)
ELSE
err := 3 + KERROR

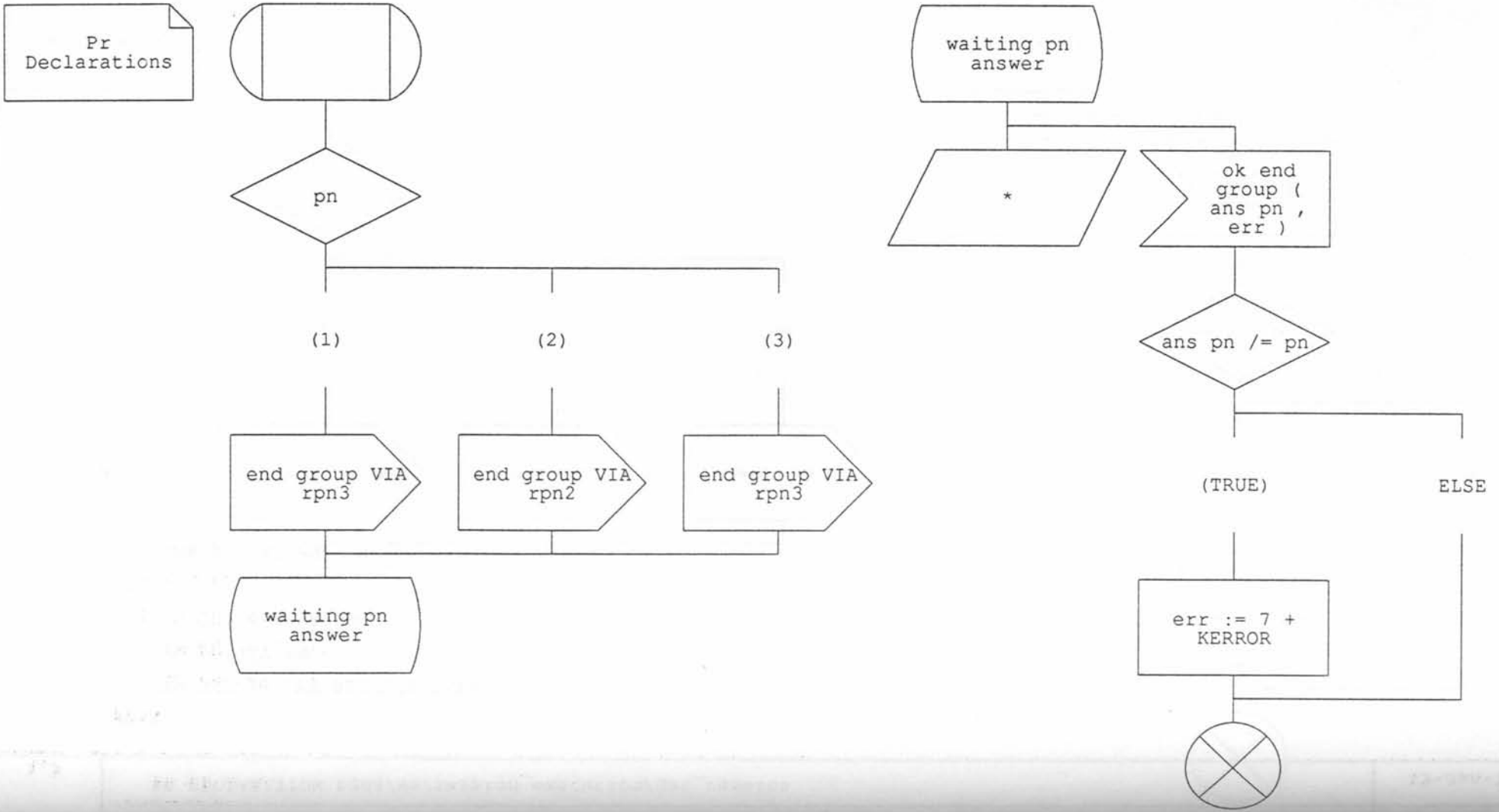
```
FPAR
    IN/OUT session_data PN_SESSION_DATA,
    IN pn NAT_PN,
    IN/OUT err NATURAL;
DCL
    ans_pn NAT_PN;
```


Pr Declar ations
ans (1) := FAL..
act (1)
(TRUE)
ELSE
act (2)
(TRUE)
ELSE
act (3)
order (func) VIA rpn1
order (func) VIA rpn2
order (func) VIA rpn3
err := 4 + KERROR
waiting pn answer
waiting pn answer
ok ord er (..
err = 0
ans (pn)
err := 5 + KERROR
ans (pn) := TRUE
ans = act
.

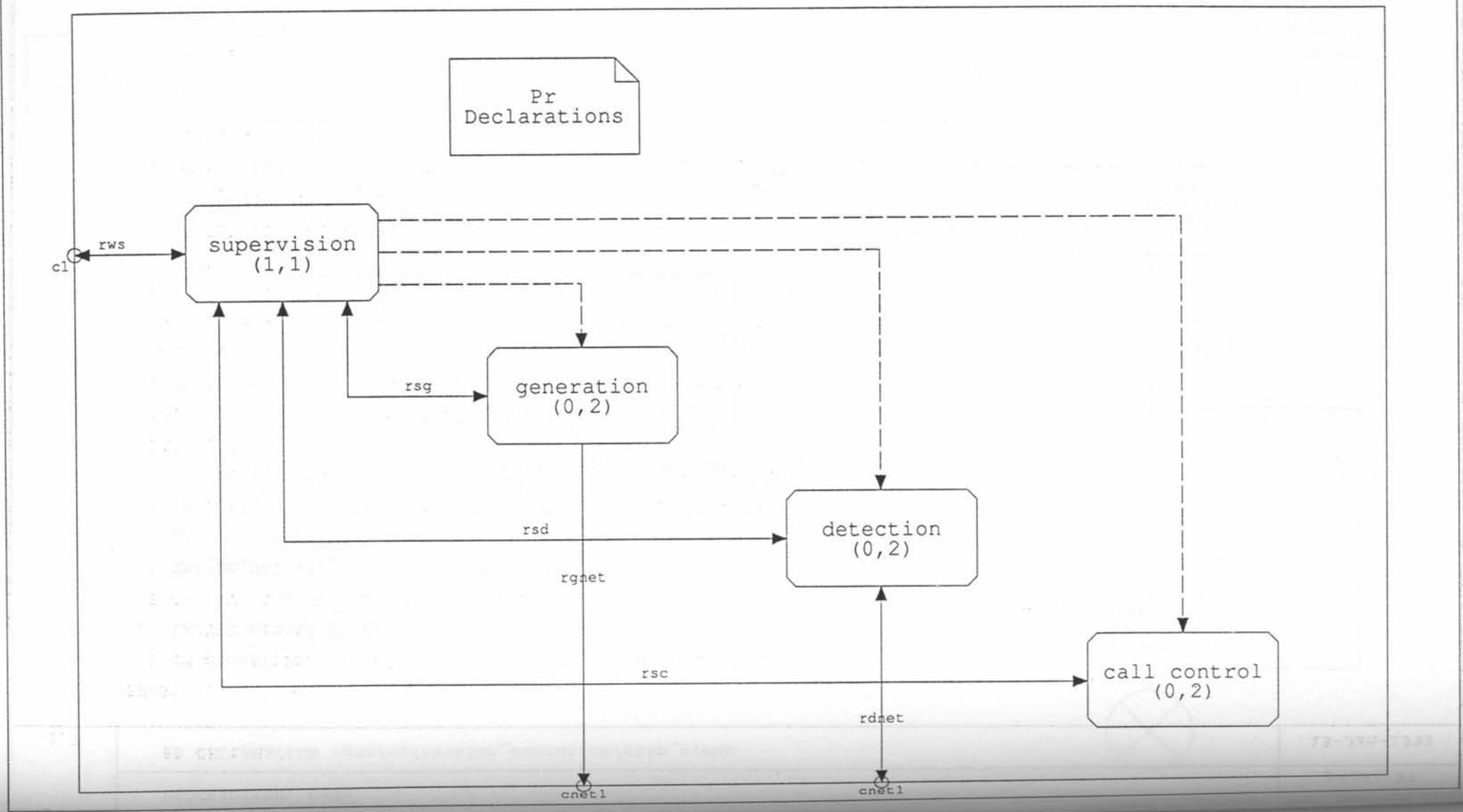
```
FPAR
    IN func STRG_BROAD,
    IN/OUT act ACTIVE_SYS,
    err NATURAL;
DCL
    ans ACTIVE_SYS,
    pn NAT_PN;
```


Pr Declarations
pn
(1)
(2)
(3)
g results VIA rpn1
g results VIA rpn2
g results VIA rpn3
waiting pn answer
waiting pn answer
*
ok g results (res ! intf1 , res ! intf2 , ans pn , err)
ans pn /= pn
(TRUE)
ELSE
err := 6 + KERROR

```
FPAR
    IN/OUT res PN_RESULTS_DATA,
    IN pn NAT_PN,
    IN/OUT err NATURAL;
DCL
    ans_pn NAT_PN;
```


Pr Declarations
pn
(1)
(2)
(3)
end group VIA rpn3
end group VIA rpn2
end group VIA rpn3
waiting pn answer
waiting pn answer
*
ok end group (ans pn , err)
ans pn /= pn
(TRUE)
ELSE
err := 7 + KERROR

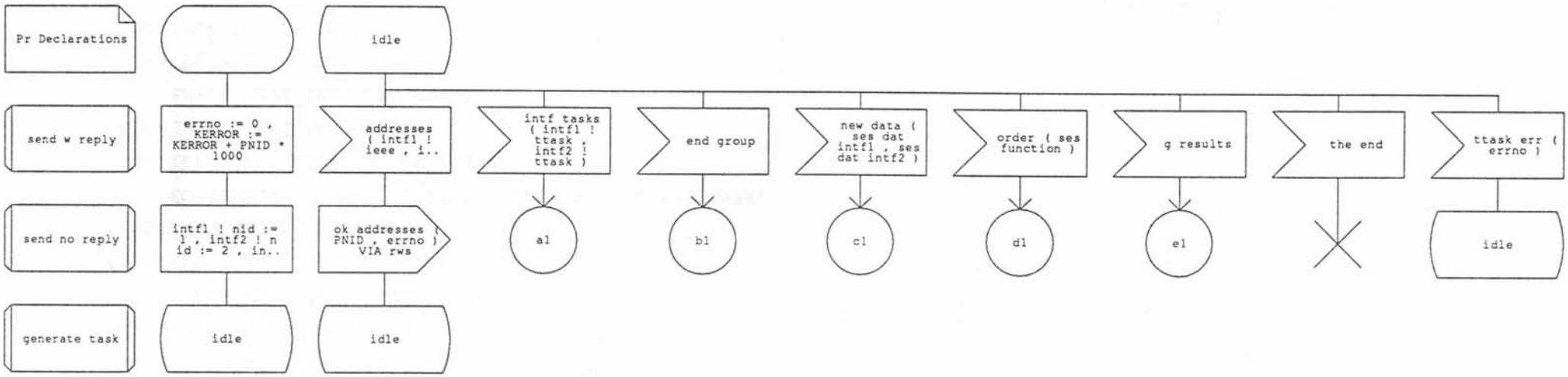
```
FPAR
   IN pn NAT_PN,
   IN/OUT err NATURAL;
DCL
   ans_pn NAT_PN;
```

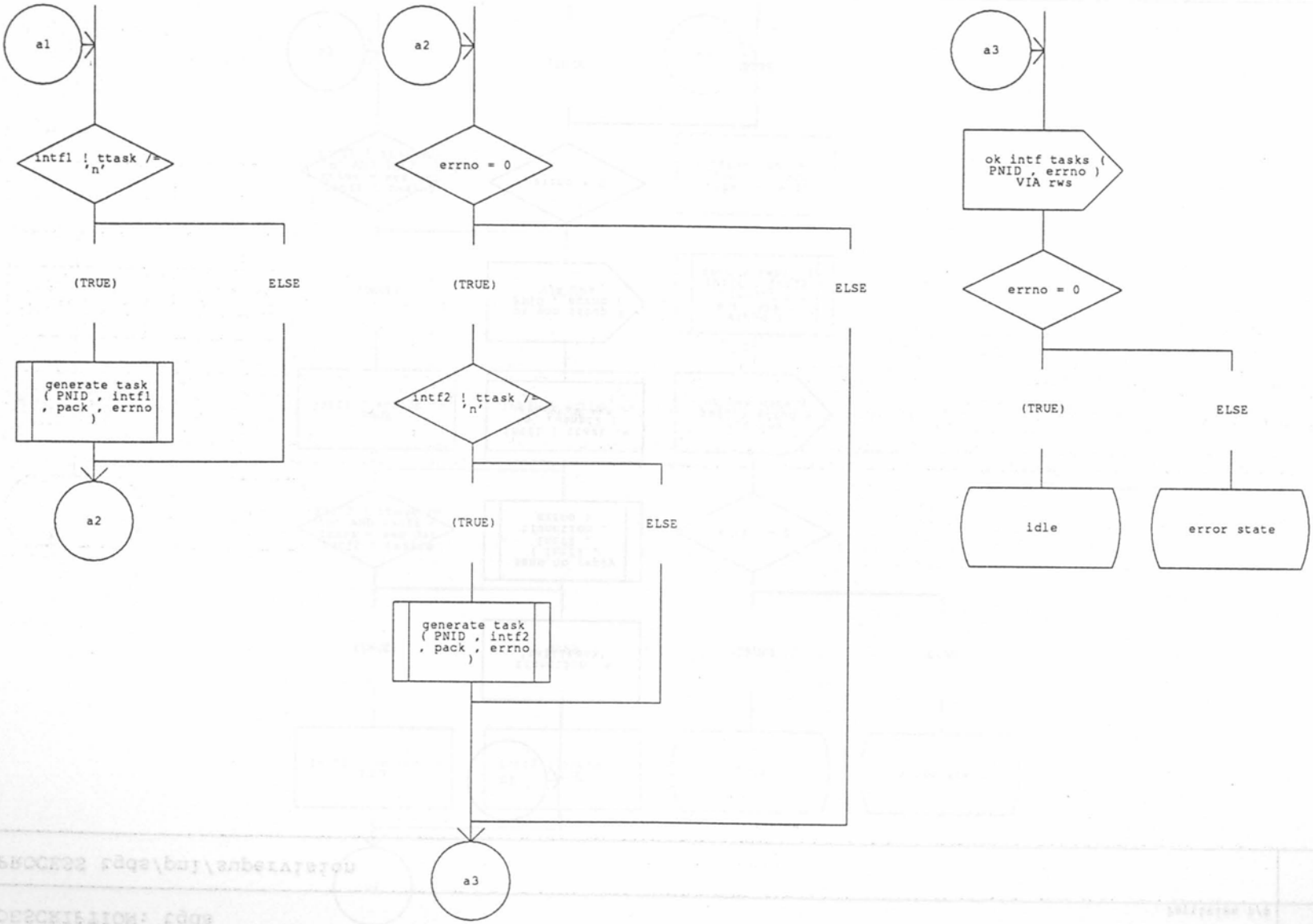

Pr
Declarations
supervision
(1,1)
c1
rws
generation
(0,2)
rsg
detection
(0,2)
rsd
rgnet
call control
(0,2)
rsc
rdnet
cnet1
cnet1

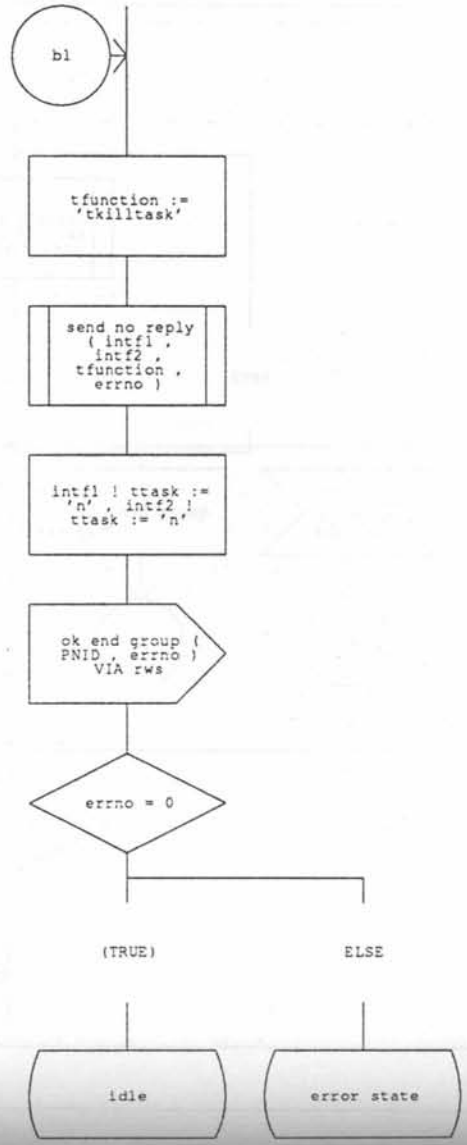
```
SIGNALROUTE  rgnet
   FROM generation  TO ENV WITH tgds_announc , tgds_packet;
SIGNALROUTE  rdnet
   FROM ENV TO detection  WITH tgds_packet;
   FROM detection  TO ENV WITH tgds_announc;
SIGNALROUTE  rsc
   FROM call_control  TO supervision  WITH ok_announc , ok_msg;
   FROM supervision  TO call_control  WITH msg , fatal_error;
SIGNALROUTE  rsd
   FROM detection  TO supervision  WITH ok_announc , ttask_err , ok_msg;
   FROM supervision  TO detection  WITH msg , fatal_error;
SIGNALROUTE  rsg
   FROM generation  TO supervision  WITH ok_announc , ttask_err , ok_msg;
   FROM supervision  TO generation  WITH msg , fatal_error;
SIGNALROUTE  rws
   FROM ENV TO supervision  WITH addresses , intf_tasks , new_data , g_results , end_group , order ,
      the_end , fatal_error;
   FROM supervision  TO ENV WITH ok_addresses , ok_intf_tasks , ok_new_data , ok_g_results , ok_end_group ,
      ok_order;
```

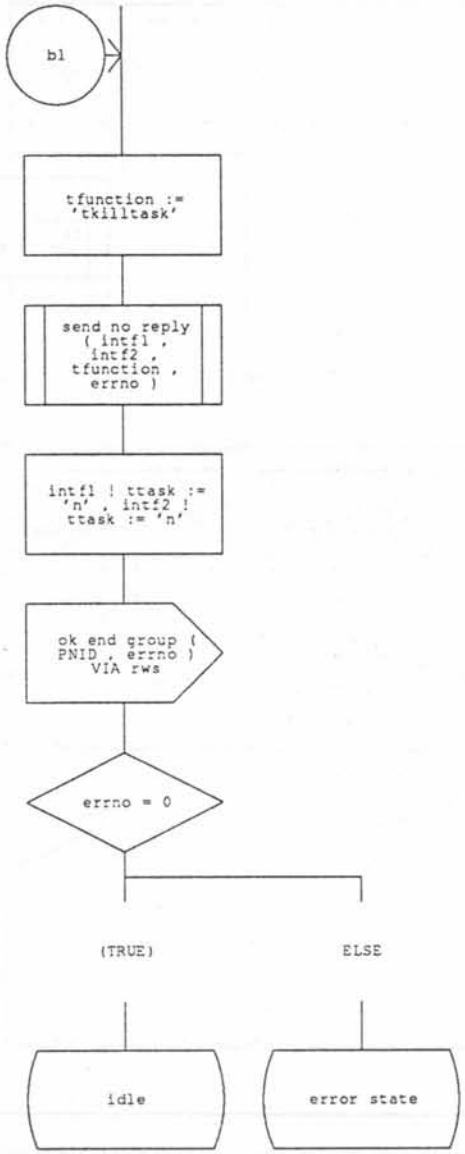
```
NEWTYPE DATA_INTF
    STRUCT
        nid NAT_INTF;
        ttask CHAR_TASK;
        taskpid PID;
        action BOOLEAN;
        ieee NATURAL;
        ann ANNOUNC;
ENDNEWTYPE DATA_INTF;
NEWTYPE ACTIVE_INTF
    ARRAY (NAT_INTF, BOOLEAN);
ENDNEWTYPE ACTIVE_INTF;
SYNTYPE STRG_ACTION = CHARSTRING
    CONSTANTS 'tinit', 'tstop', 'tstart', 'tkilltask'
ENDSYNTYPE STRG_ACTION;
SYNTYPE STRG_MAIN_ACT = CHARSTRING
   CONSTANTS 'gen', 'det', 'all'
ENDSYNTYPE STRG_MAIN_ACT;
/* ^ data types for the supervision task ^ */
SIGNAL
    msg(STRG_ACTION),
    ok_msg(STRG_ACTION, NATURAL),
    ok_announc(NATURAL),
    ttask_err(NATURAL);
/* signals exchanged between the supervision and the ttasks ^ */
SIGNAL
```

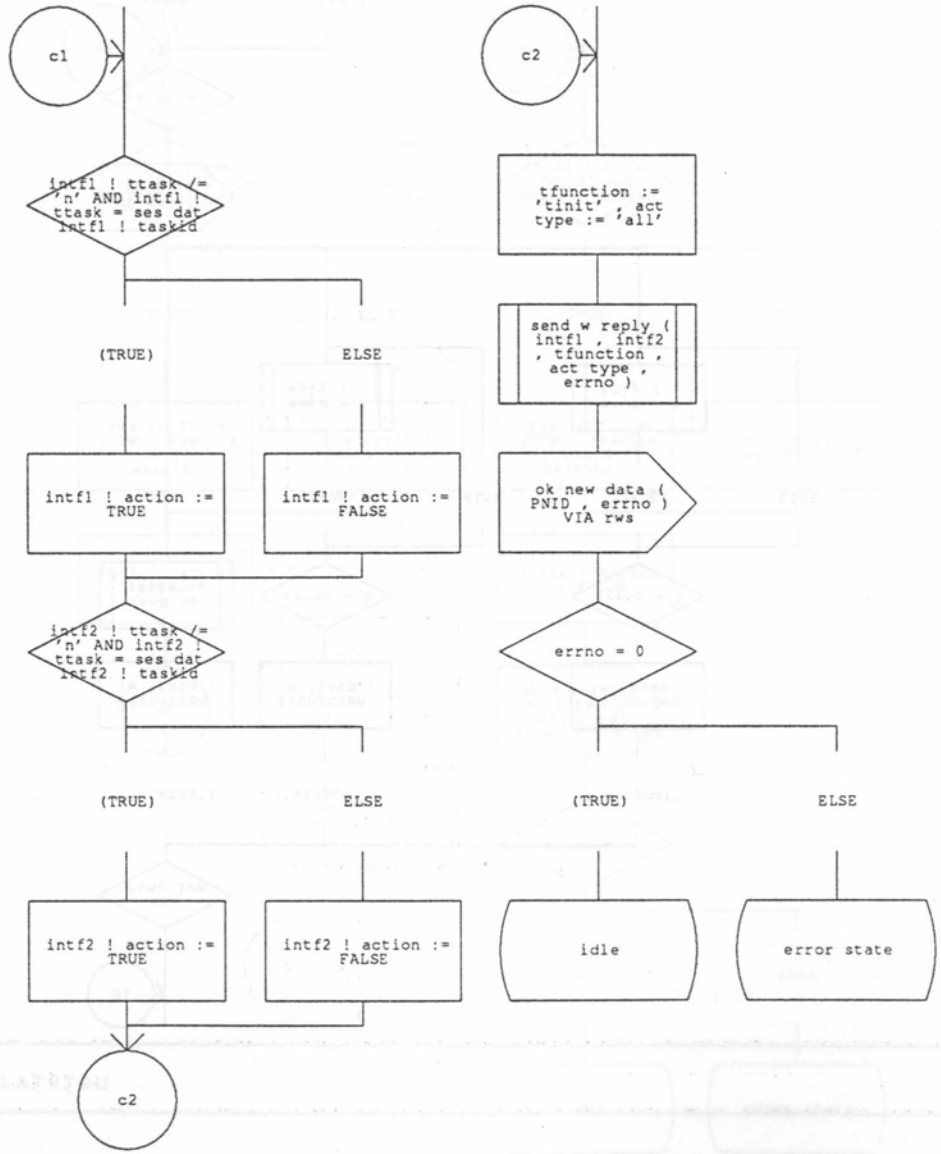
```
    stop_rx;
/* ^ signal exchanged between detection and itself ^ */
```

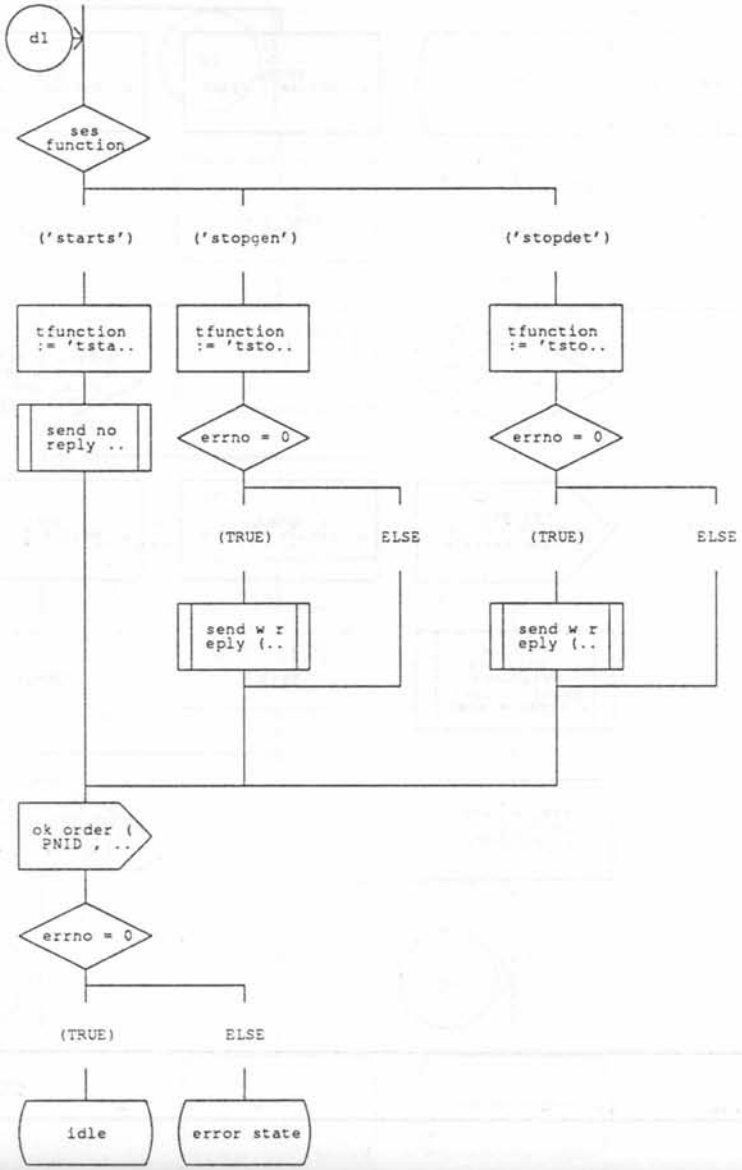

Pr Declarations
send w reply
send no reply
generate task
errno := 0 , KERROR := KERROR + PNID * 1000
intf1 ! nid := 1 , intf2 ! nid := 2 , in..
idle
idle
addresses (intf1 ! ieee , i..
ok addresses (PNID , errno) VIA rws
idle
intf tasks (intf1 ! ttask , intf2 ! ttask)
a1
end group
b1
new data (ses dat intf1 , ses dat intf2)
c1
order (ses function)
d1
g results
e1
the end
ttask err (errno)
idle

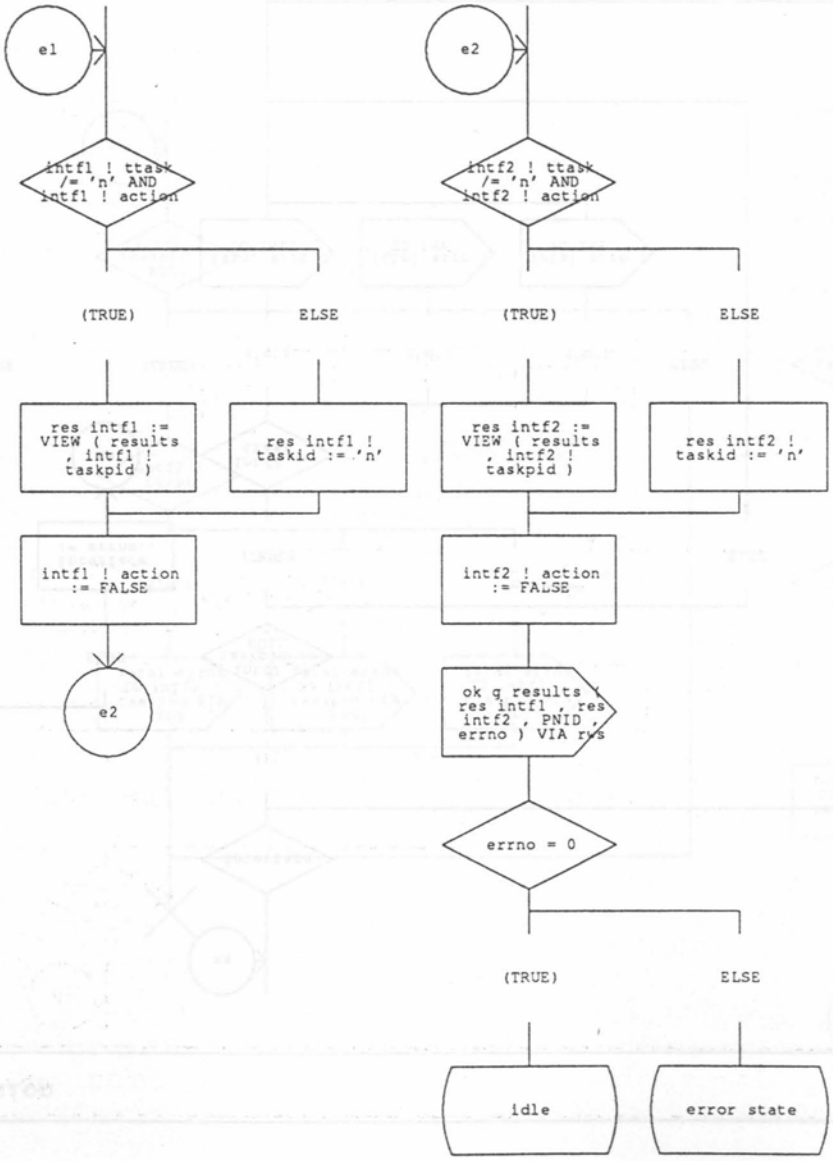

a1
intf1 ! ttask /= 'n'
(TRUE)
ELSE
generate task (PNID , intf1 , pack , errno)
a2
a2
errno = 0
(TRUE)
ELSE
intf2 ! ttask /= 'n'
(TRUE)
ELSE
generate task (PNID , intf2 , pack , errno)
a3
a3
ok intf tasks (PNID , errno) VIA rws
errno = 0
(TRUE)
ELSE
idle
error state

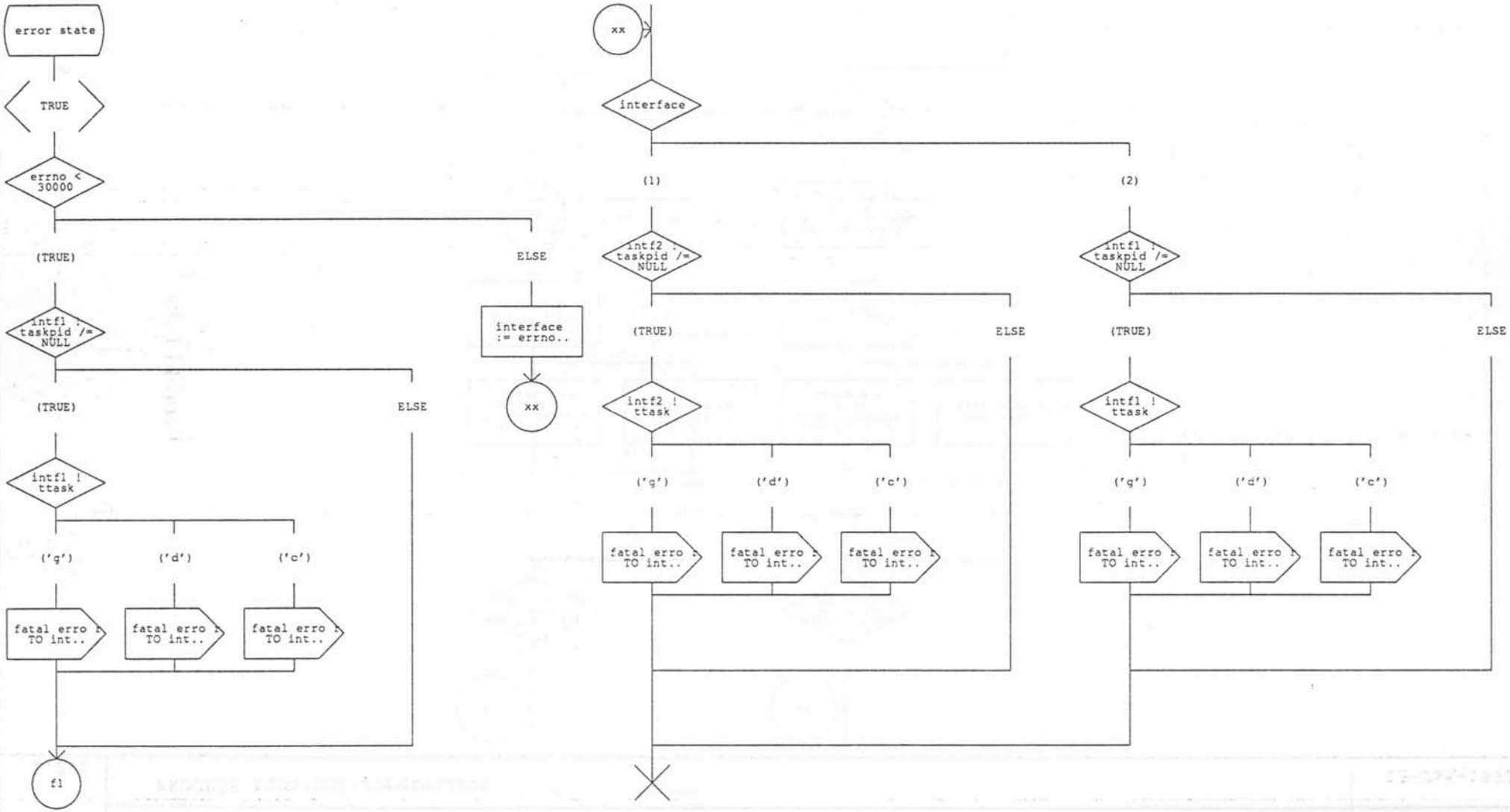

b1
tfunction := 'tkilltask'
send no reply (intf1 , intf2 , tfunction , errno)
intf1 ! ttask := 'n' , intf2 ! ttask := 'n'
ok end group (PNID , errno) VIA rws
errno = 0
(TRUE)
ELSE
idle
error state

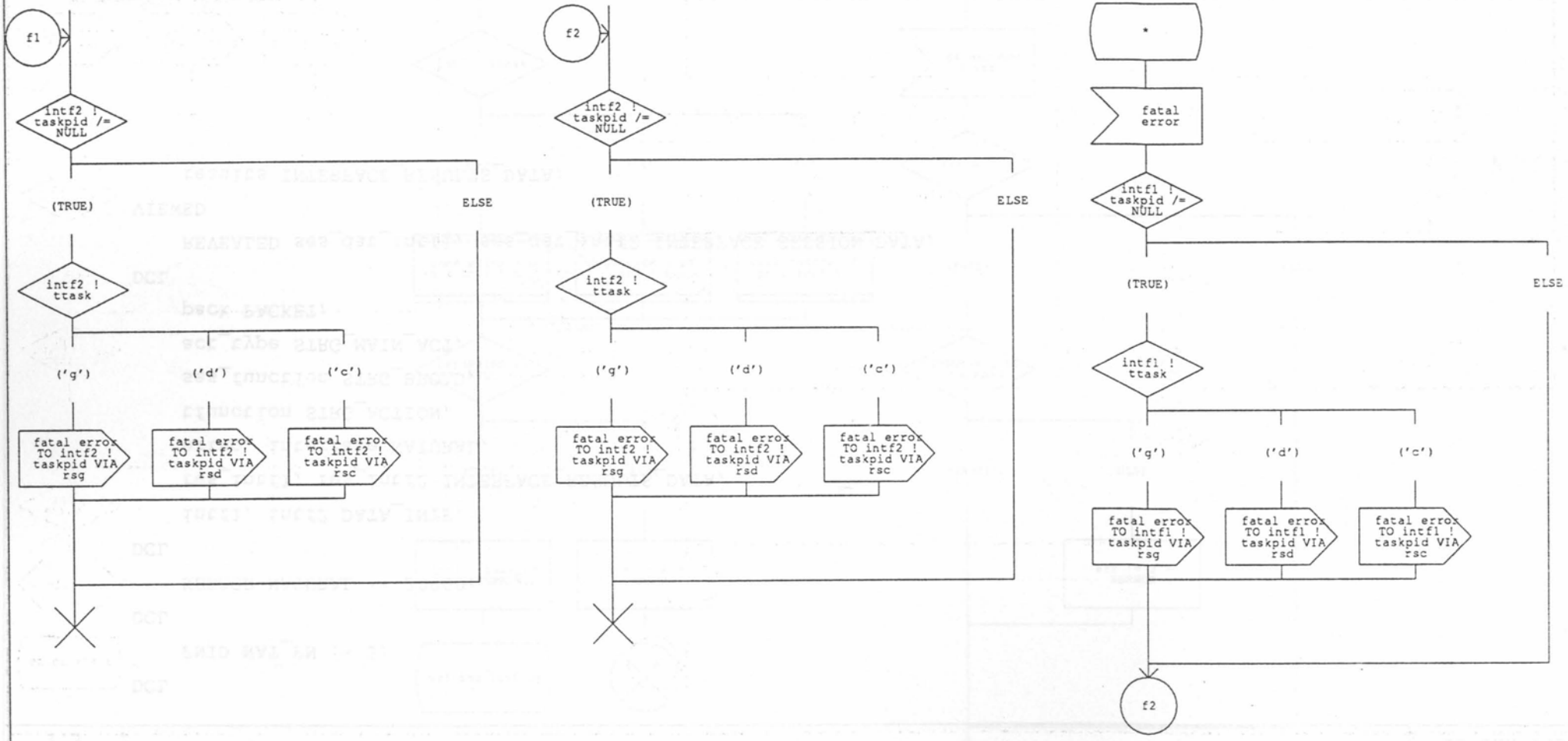

b1
tfunction := 'tkilltask'
send no reply (intf1 , intf2 , tfunction , errno)
intf1 ! ttask := 'n' , intf2 ! ttask := 'n'
ok end group (PNID , errno) VIA rws
errno = 0
(TRUE)
ELSE
idle
error state


c1
intf1 ! ttask /= 'n' AND intf1 ! ttask = ses dat intf1 ! taskid
(TRUE)
ELSE
intf1 ! action := TRUE
intf1 ! action := FALSE
intf2 ! ttask /= 'n' AND intf2 ! ttask = ses dat intf2 ! taskid
(TRUE)
ELSE
intf2 ! action := TRUE
intf2 ! action := FALSE
c2
c2
tfunction := 'tinit' , act type := 'all'
send w reply (intf1 , intf2 , tfunction , act type , errno)
ok new data (PNID , errno) VIA rws
errno = 0
(TRUE)
ELSE
idle
error state


d1
ses function
('starts')
('stopgen')
('stopdet')
tfunction := 'tsta..
tfunction := 'tsto..
tfunction := 'tsto..
send no reply ..
errno = 0
errno = 0
(TRUE)
ELSE
(TRUE)
ELSE
send w r eply (..
send w r eply (..
ok order (PNID , ..
errno = 0
(TRUE)
ELSE
idle
error state

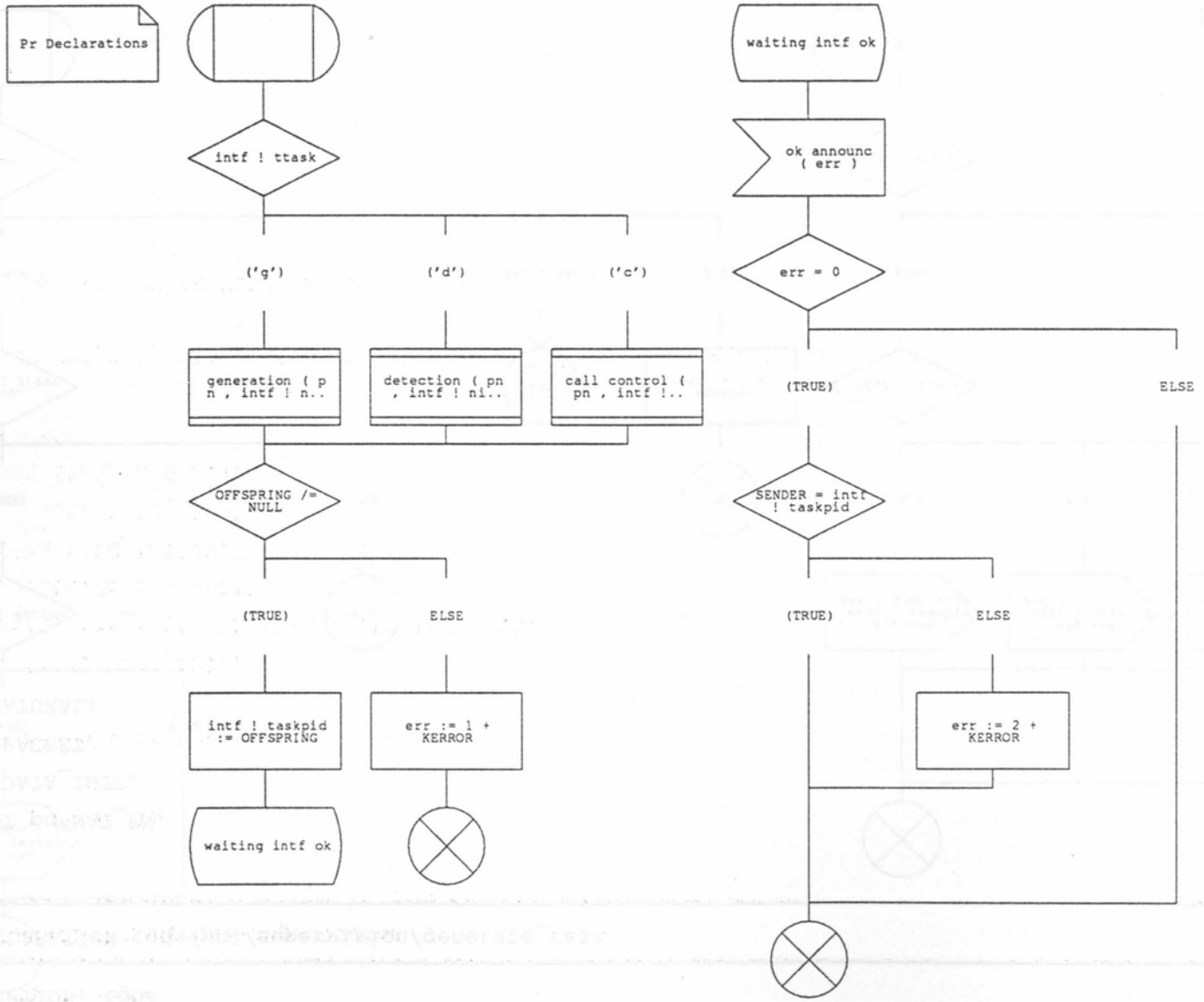

e1
intf1 ! ttask /= 'n' AND intf1 ! action
(TRUE)
ELSE
res intf1 := VIEW (results , intf1 ! taskpid)
res intf1 ! taskid := 'n'
intf1 ! action := FALSE
e2
e2
intf2 ! ttask /= 'n' AND intf2 ! action
(TRUE)
ELSE
res intf2 := VIEW (results , intf2 ! taskpid)
res intf2 ! taskid := 'n'
intf2 ! action := FALSE
ok q results (res intf1 , res intf2 , PNID , errno) VIA rvs
errno = 0
(TRUE)
ELSE
idle
error state


error state
TRUE
errno < 30000
(TRUE)
ELSE
intf1 . taskpid /= NULL
(TRUE)
ELSE
intf1 ! ttask
('g')
('d')
('c')
fatal erro r TO int..
fatal erro r TO int..
fatal erro r TO int..
f1
interface := errno..
xx
xx
interface
(1)
(2)
intf2 . taskpid /= NULL
(TRUE)
ELSE
intf2 ! ttask
('g')
('d')
('c')
fatal erro r TO int..
fatal erro r TO int..
fatal erro r TO int..
intf1 . taskpid /= NULL
(TRUE)
ELSE
intf1 ! ttask
('g')
('d')
('c')
fatal erro r TO int..
fatal erro r TO int..
fatal erro r TO int..

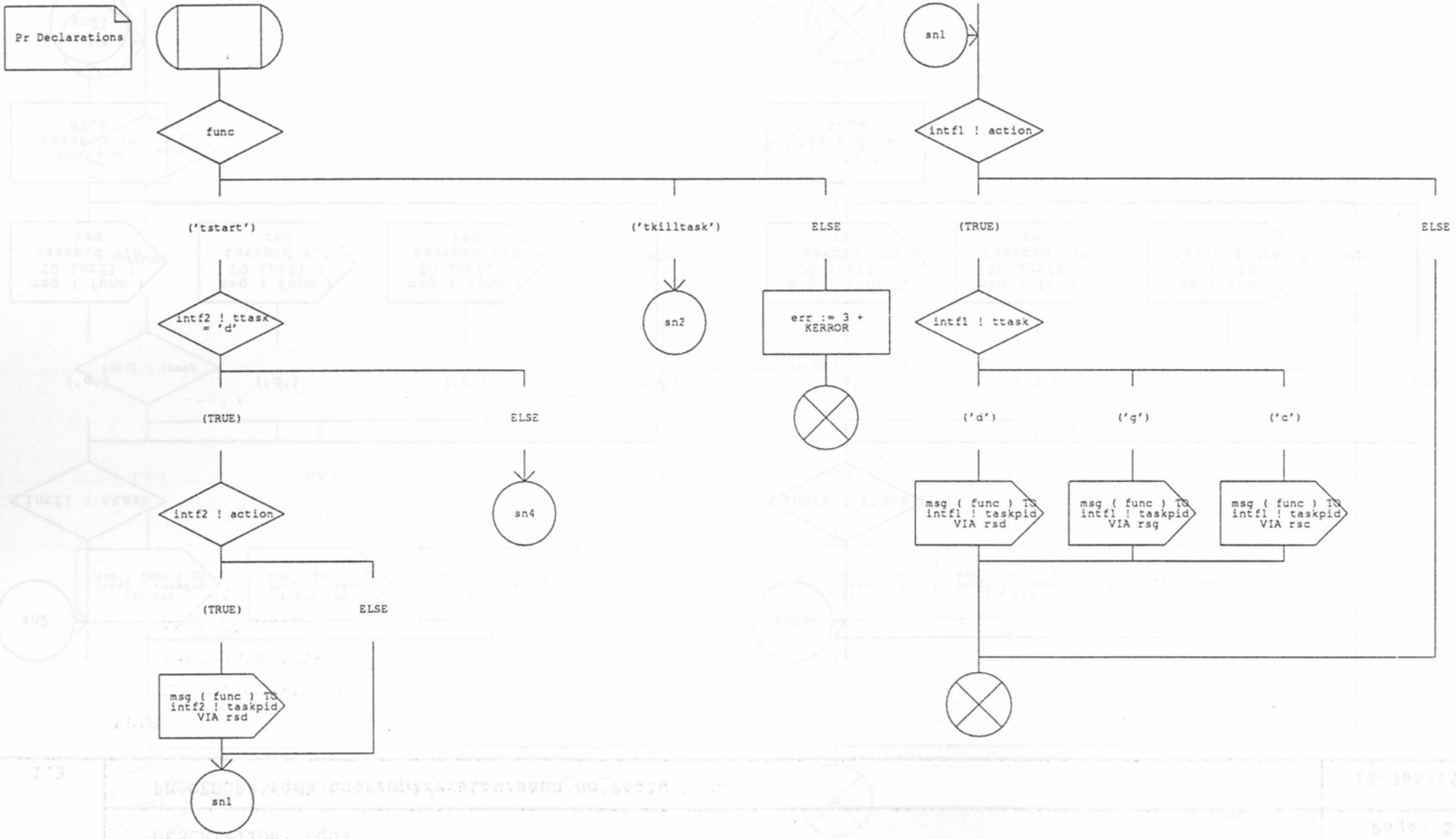

f1
intf2 ! taskpid /= NULL
(TRUE)
ELSE
intf2 ! ttask
('g')
('d')
('c')
fatal error TO intf2 ! taskpid VIA rsg
fatal error TO intf2 ! taskpid VIA rsd
fatal error TO intf2 ! taskpid VIA rsc
f2
intf2 ! taskpid /= NULL
(TRUE)
ELSE
intf2 ! ttask
('g')
('d')
('c')
fatal error TO intf2 ! taskpid VIA rsg
fatal error TO intf2 ! taskpid VIA rsd
fatal error TO intf2 ! taskpid VIA rsc
*
fatal error
intf1 ! taskpid /= NULL
(TRUE)
ELSE
intf1 ! ttask
('g')
('d')
('c')
fatal error TO intf1 ! taskpid VIA rsg
fatal error TO intf1 ! taskpid VIA rsd
fatal error TO intf1 ! taskpid VIA rsc
f2

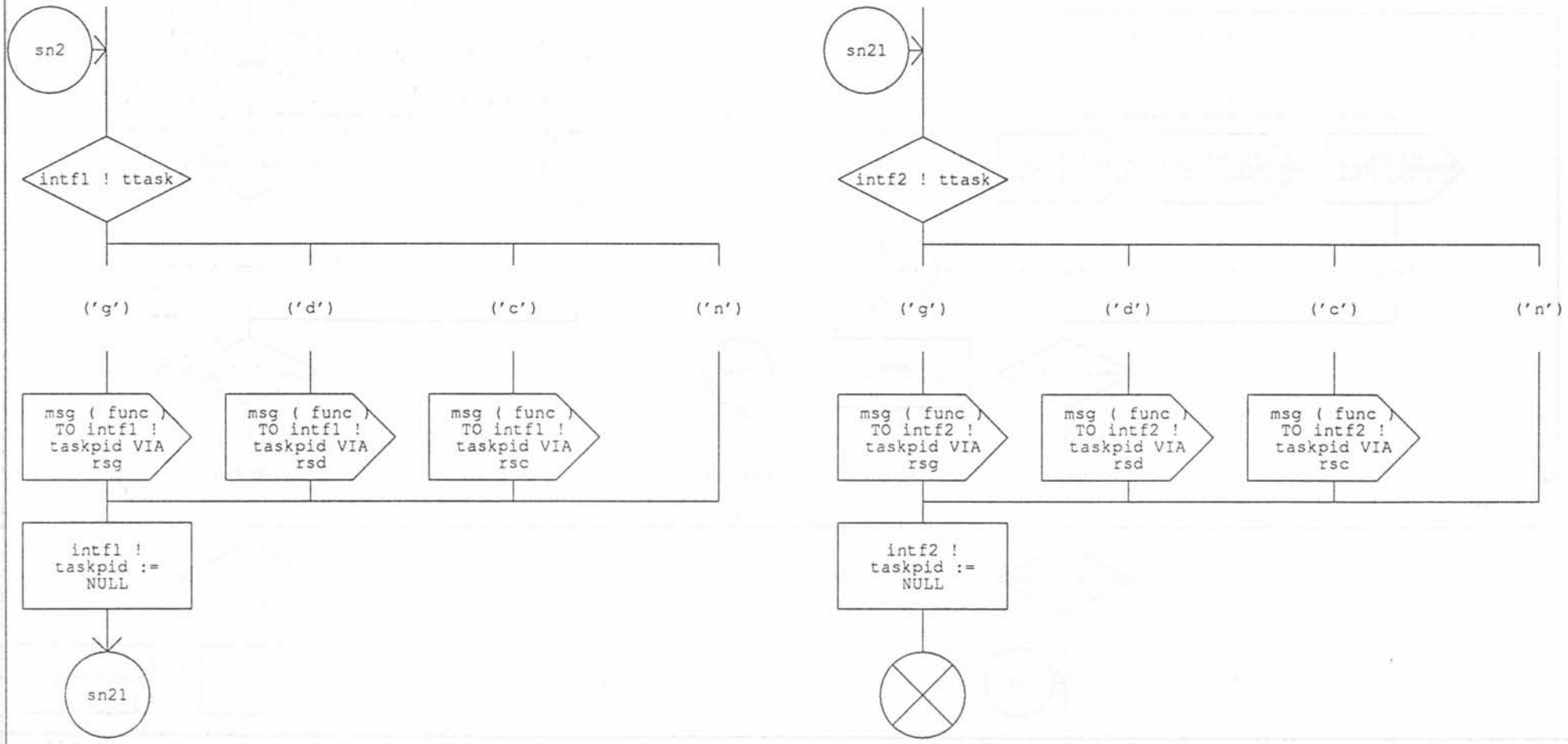
```
DCL
    PNID NAT_PN := 1;
DCL
    KERROR NATURAL := 20000;
DCL
    intf1, intf2 DATA_INTF,
    res_intf1, res_intf2 INTERFACE_RESULTS_DATA,
    errno, interface NATURAL,
    tfunction STRG_ACTION,
    ses_function STRG_BROAD,
    act_type STRG_MAIN_ACT,
    pack PACKET;
DCL
    REVEALED ses_dat_intf1, ses_dat_intf2 INTERFACE_SESSION_DATA;
VIEWED
    results INTERFACE_RESULTS_DATA;
```

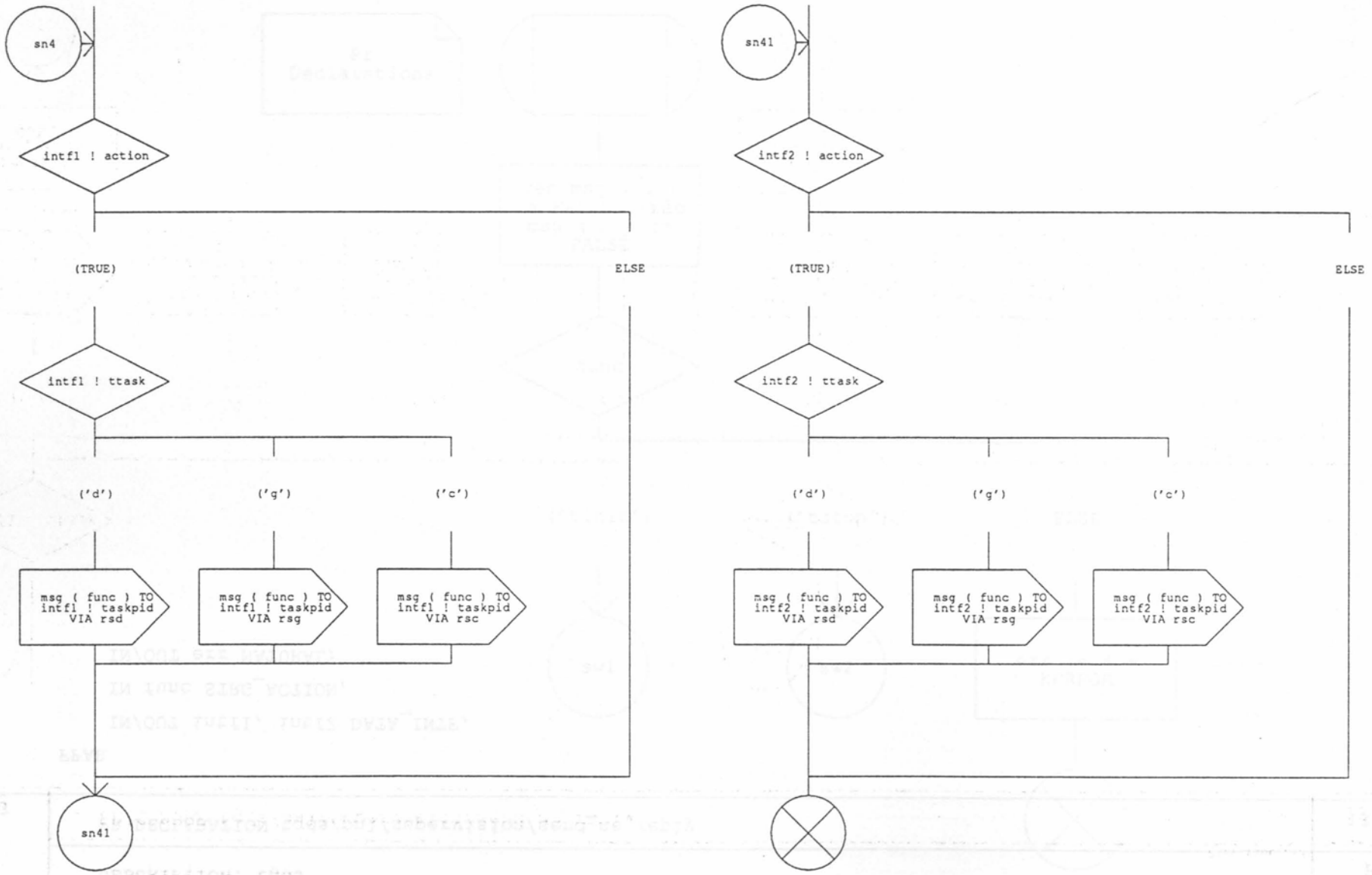
	DESCRIPTION: tgds Partition 1/1	Page: 51
1.3	PROCEDURE tgds/pn1/supervision/generate_task	13-Jan-1992



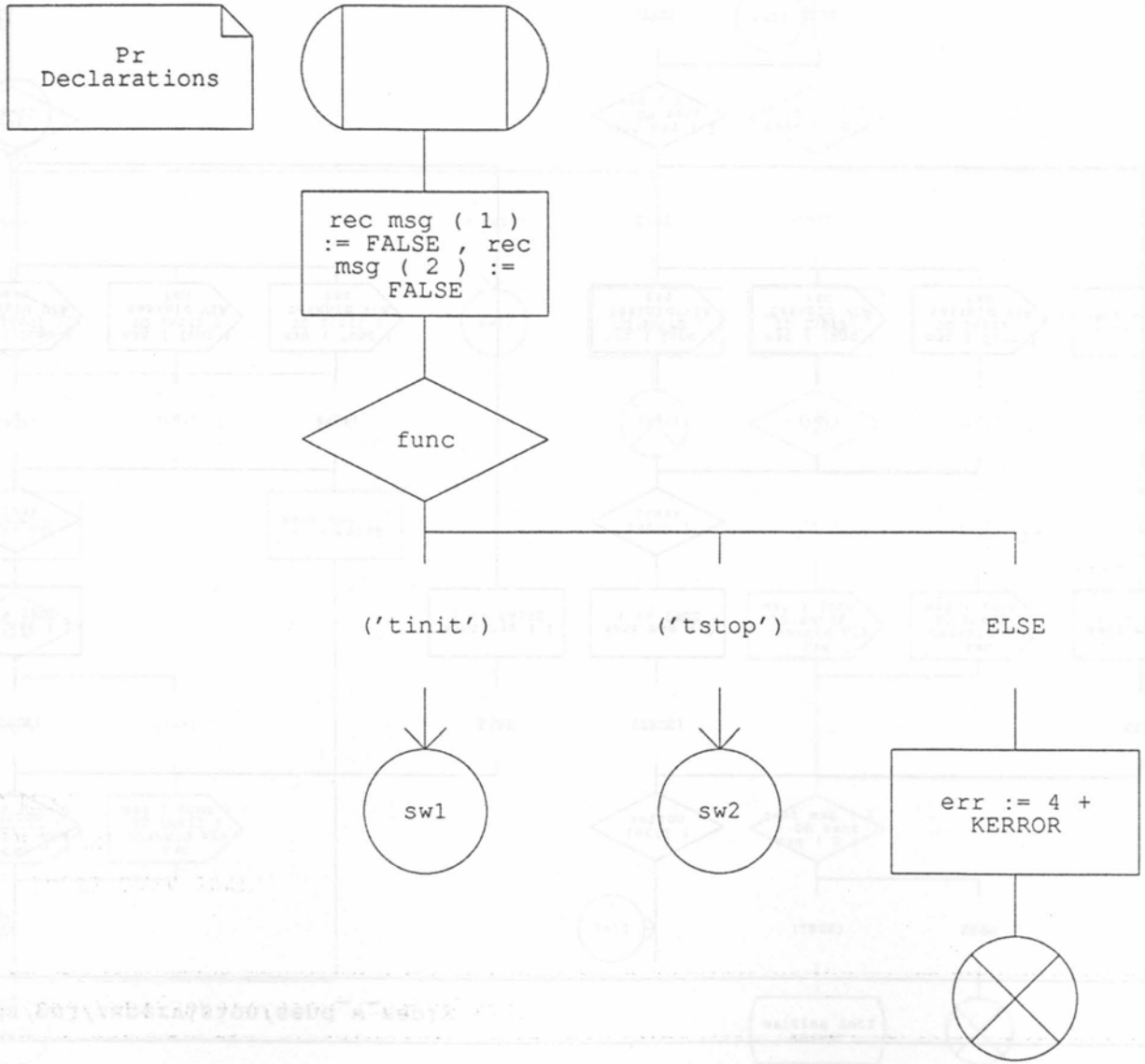
```
FPAR
    IN/OUT pn NAT_PN,
    intf DATA_INTF,
    pack PACKET,
    err NATURAL;
```

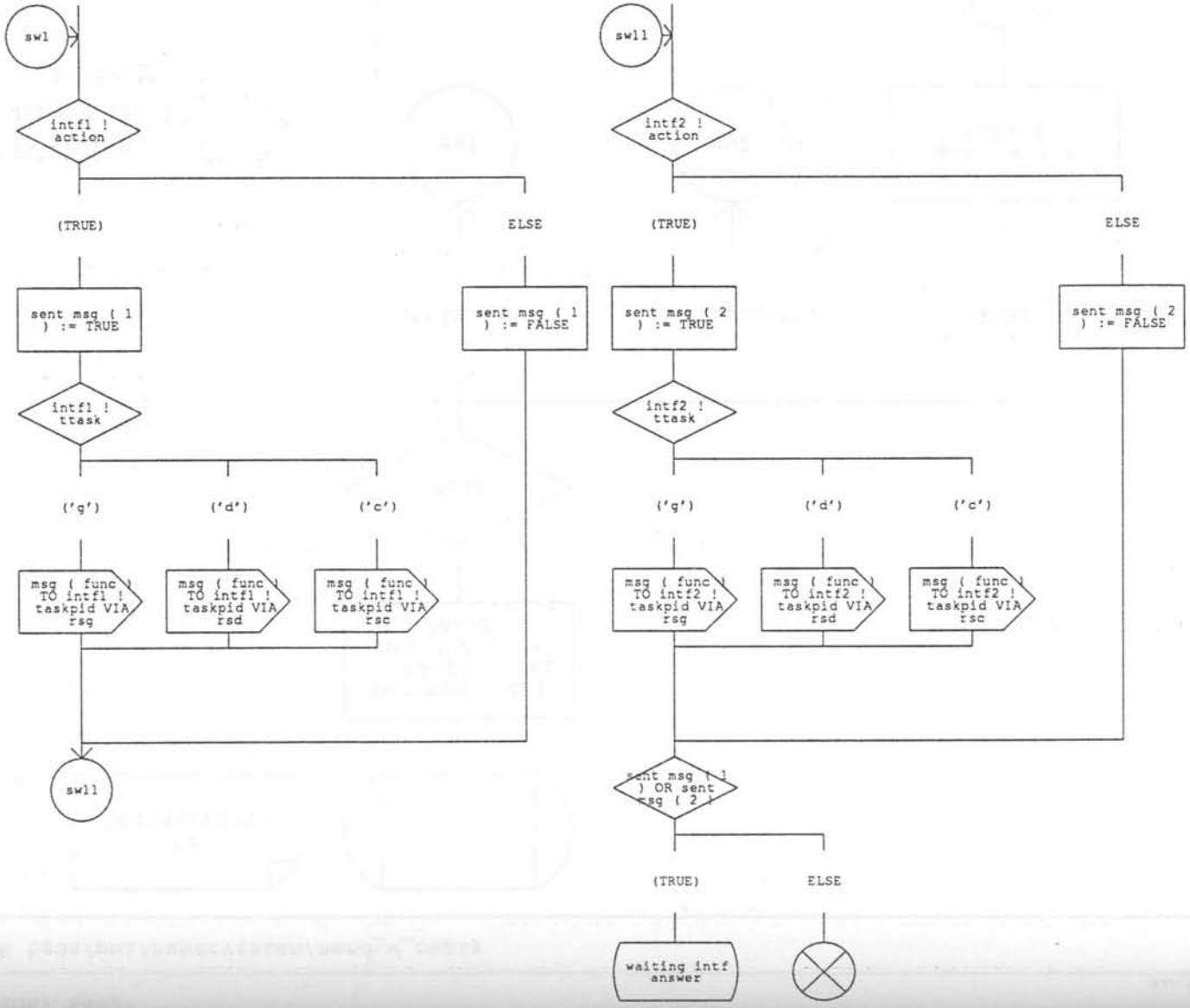

Pr Declarations
func
('tstart')
('tkilltask')
ELSE
intf2 ! ttask = 'd'
sn2
err := 3 + KERROR
(TRUE)
ELSE
intf2 ! action
sn4
(TRUE)
ELSE
msg (func) TO intf2 ! taskpid VIA rsd
sn1
sn1
intf1 ! action
(TRUE)
ELSE
intf1 ! ttask
('d')
('g')
('c')
msg (func) TO intf1 ! taskpid VIA rsd
msg (func) TO intf1 ! taskpid VIA rsg
msg (func) TO intf1 ! taskpid VIA rsc

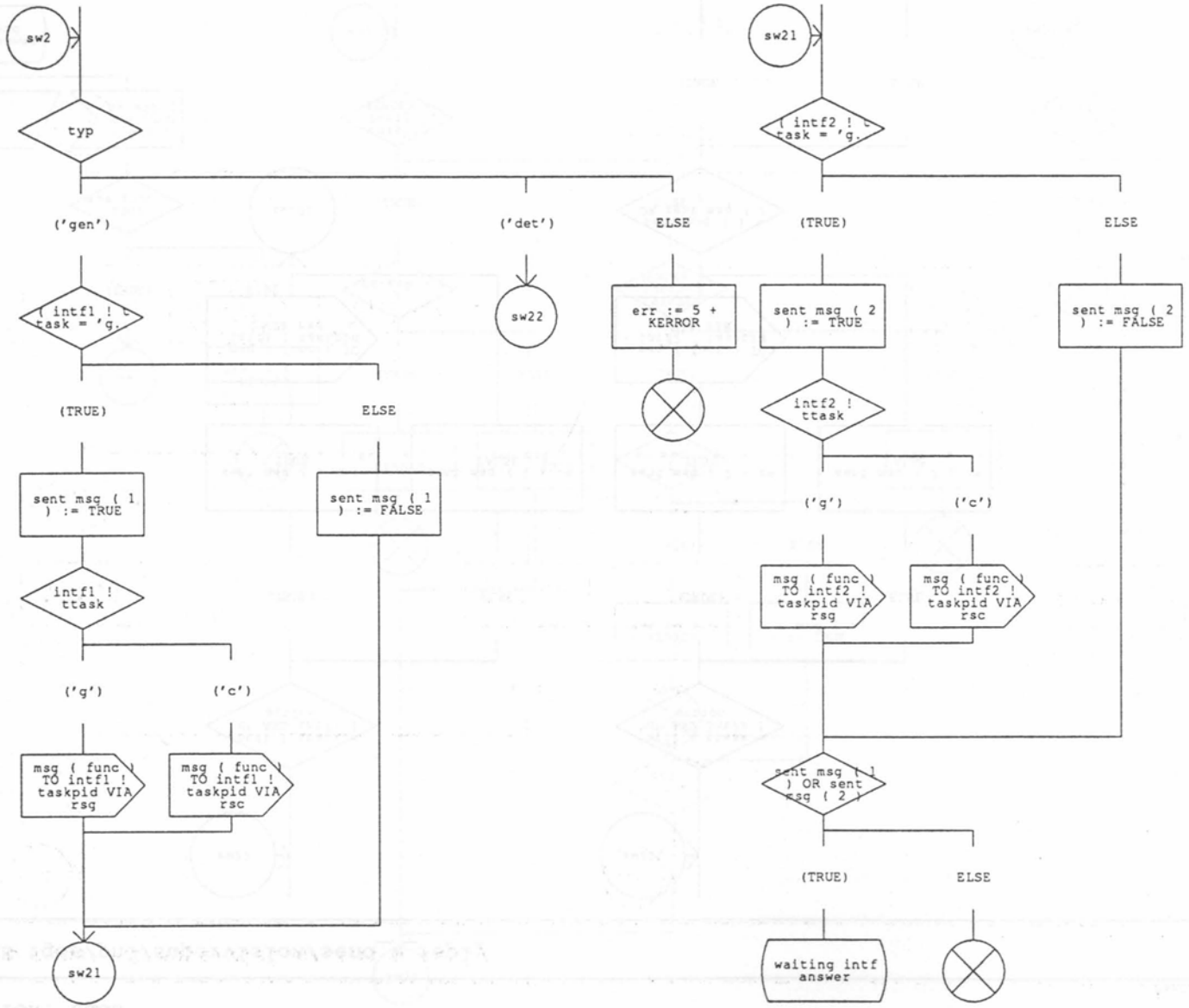

sn2
intf1 ! ttask
('g')
('d')
('c')
('n')
msg (func) TO intf1 ! taskpid VIA rsg
msg (func) TO intf1 ! taskpid VIA rsd
msg (func) TO intf1 ! taskpid VIA rsc
intf1 ! taskpid := NULL
sn21
sn21
intf2 ! ttask
('g')
('d')
('c')
('n')
msg (func) TO intf2 ! taskpid VIA rsg
msg (func) TO intf2 ! taskpid VIA rsd
msg (func) TO intf2 ! taskpid VIA rsc
intf2 ! taskpid := NULL

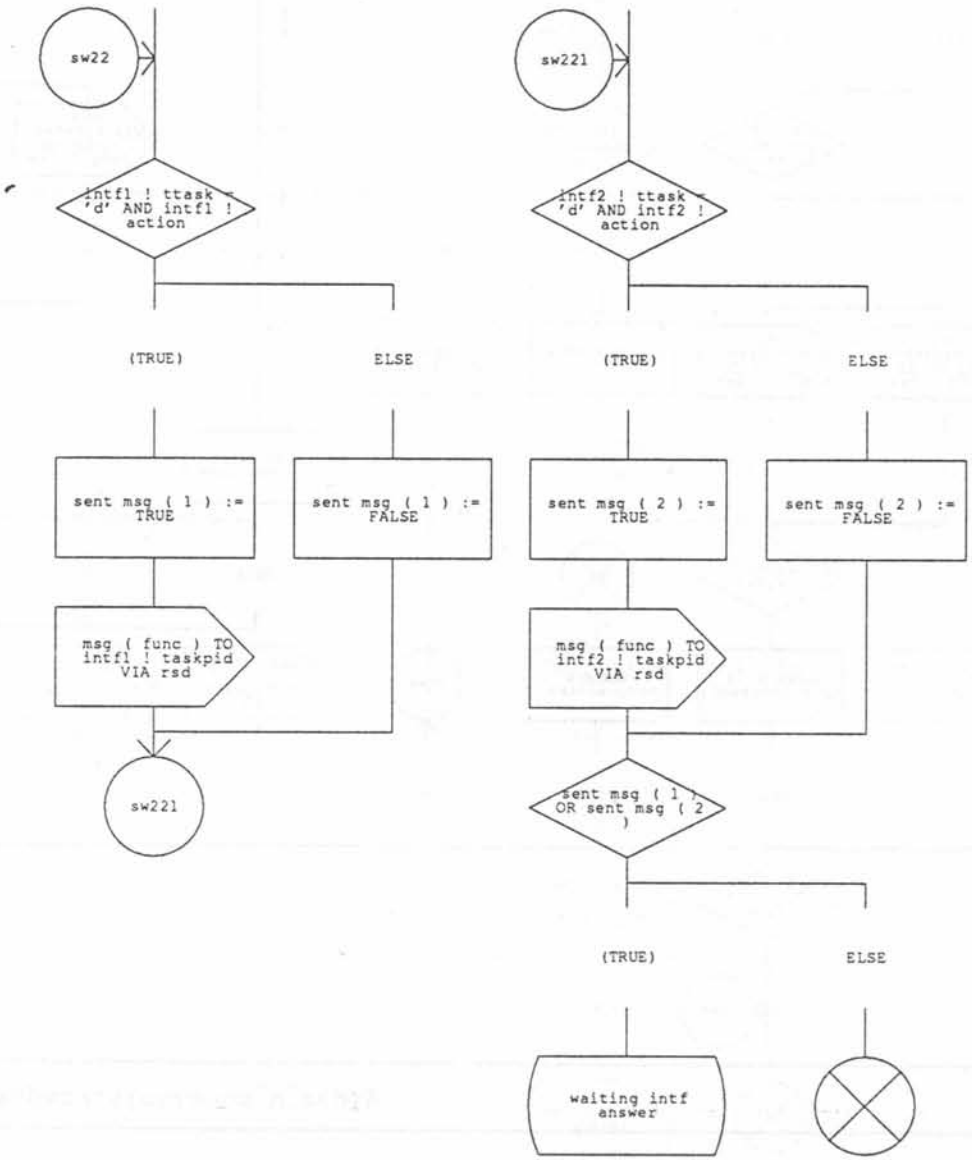

sn4
intf1 ! action
(TRUE)
ELSE
intf1 ! ttask
('d')
('g')
('c')
msg (func) TO intf1 ! taskpid VIA rsd
msg (func) TO intf1 ! taskpid VIA rsg
msg (func) TO intf1 ! taskpid VIA rsc
sn41
sn41
intf2 ! action
(TRUE)
ELSE
intf2 ! ttask
('d')
('g')
('c')
msg (func) TO intf2 ! taskpid VIA rsd
msg (func) TO intf2 ! taskpid VIA rsg
msg (func) TO intf2 ! taskpid VIA rsc

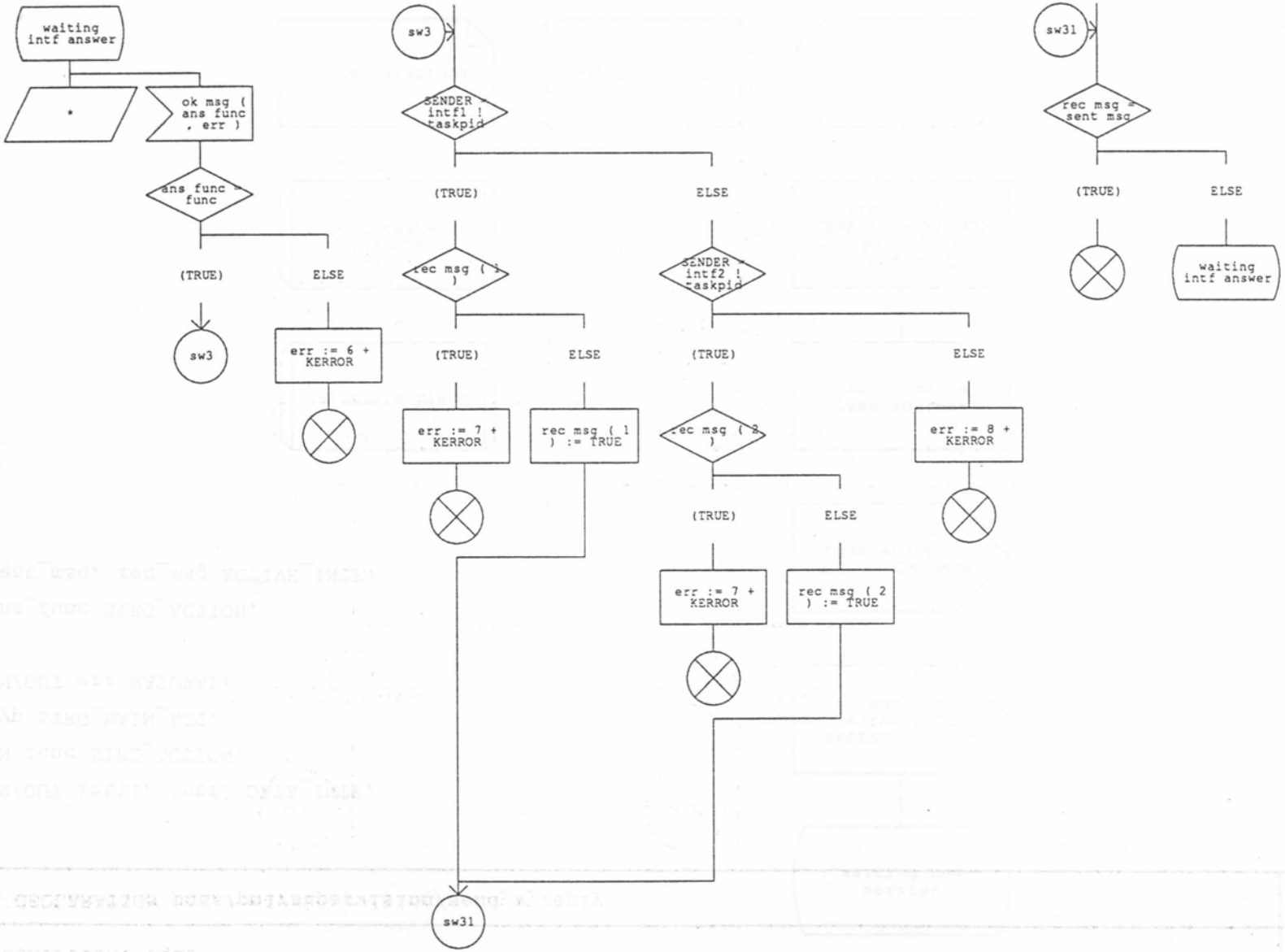
```
FPAR
    IN/OUT intf1, intf2 DATA_INTF,
    IN func STRG_ACTION,
    IN/OUT err NATURAL;
```


Pr
Declarations
rec msg (1) := FALSE , rec msg (2) := FALSE
func
('tinit')
('tstop')
ELSE
sw1
sw2
err := 4 + KERROR

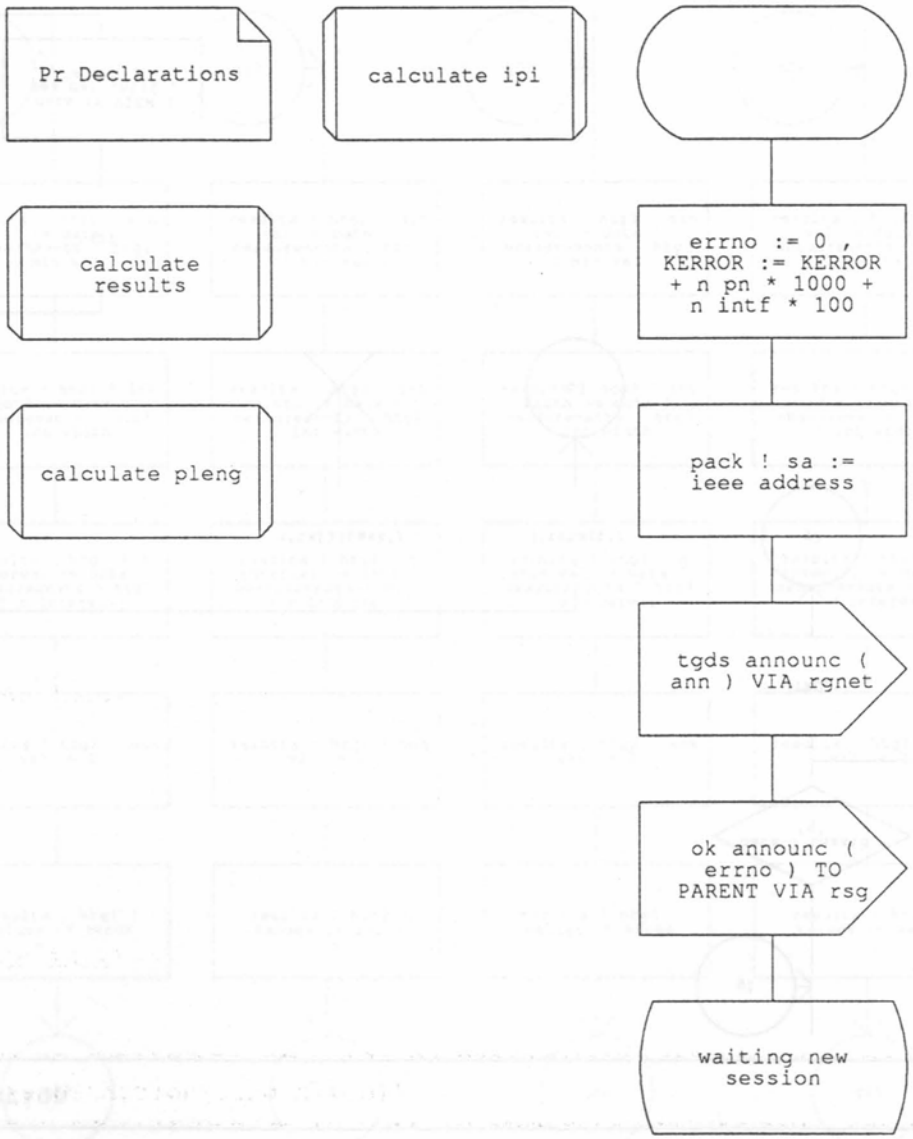

sw1
intf1 ! action
(TRUE)
ELSE
sent msg (1) := TRUE
sent msg (1) := FALSE
intf1 ! ttask
('g')
('d')
('c')
msg (func) TO intf1 ! taskpid VIA rsg
msg (func) TO intf1 ! taskpid VIA rsd
msg (func) TO intf1 ! taskpid VIA rsc
sw11
sw11
intf2 ! action
(TRUE)
ELSE
sent msg (2) := TRUE
sent msg (2) := FALSE
intf2 ! ttask
('g')
('d')
('c')
msg (func) TO intf2 ! taskpid VIA rsg
msg (func) TO intf2 ! taskpid VIA rsd
msg (func) TO intf2 ! taskpid VIA rsc
sent msg (1) OR sent msg (2)
(TRUE)
ELSE
waiting intf answer

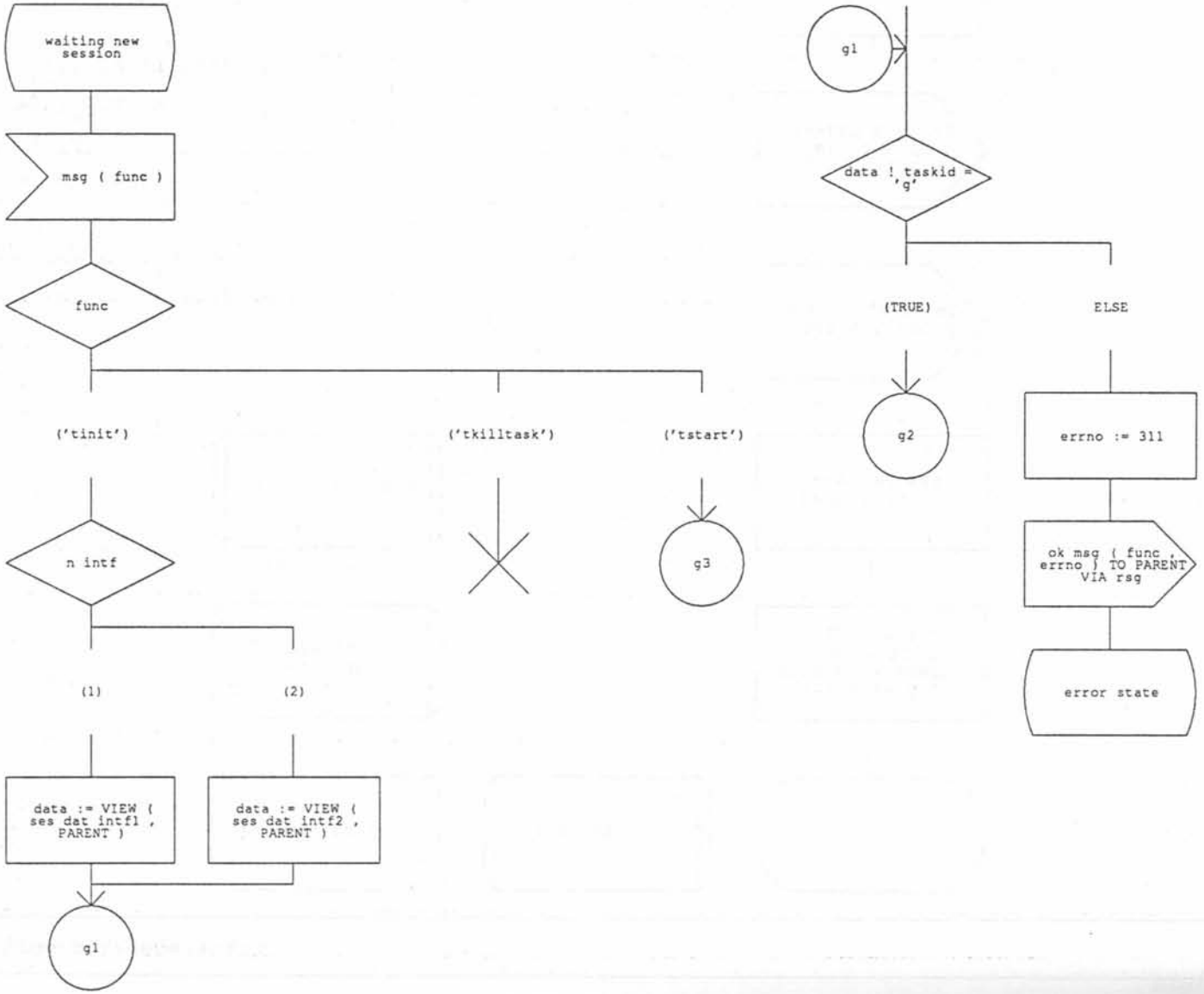

sw2
typ
('gen')
('det')
ELSE
(intf1 ! ttask = 'g.
sw22
err := 5 + KERROR
(TRUE)
ELSE
sent msg (1) := TRUE
sent msg (1) := FALSE
intf1 ! ttask
('g')
('c')
msg (func) TO intf1 ! taskpid VIA rsg
msg (func) TO intf1 ! taskpid VIA rsc
sw21
sw21
(intf2 ! ttask = 'g.
(TRUE)
ELSE
sent msg (2) := TRUE
sent msg (2) := FALSE
intf2 ! ttask
('g')
('c')
msg (func) TO intf2 ! taskpid VIA rsg
msg (func) TO intf2 ! taskpid VIA rsc
sent msg (1) OR sent msg (2)
(TRUE)
ELSE
waiting intf answer

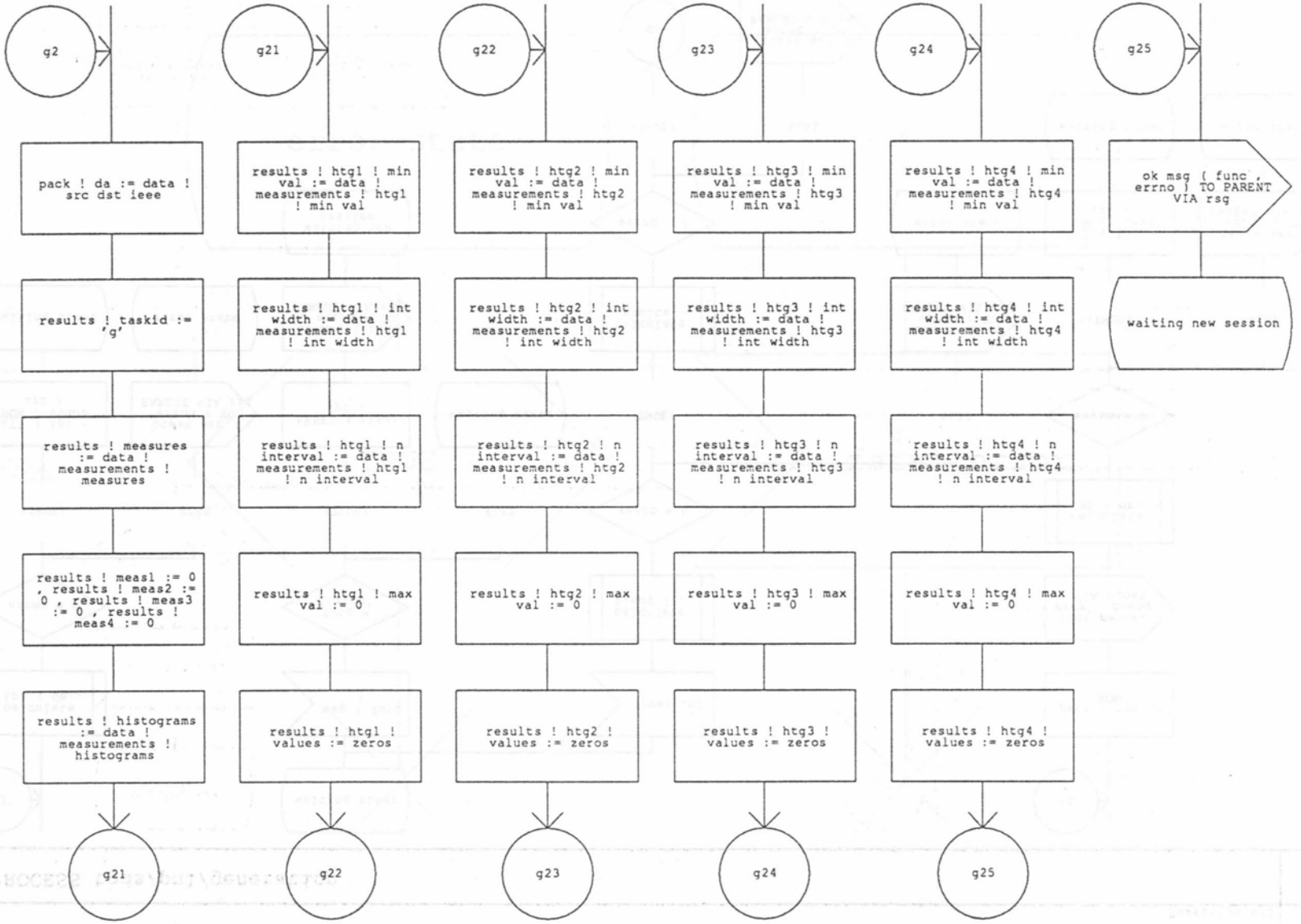

sw22
intf1 ! ttask = 'd' AND intf1 ! action
(TRUE)
ELSE
sent msg (1) := TRUE
sent msg (1) := FALSE
msg (func) TO intf1 ! taskpid VIA rsd
sw221
sw221
intf2 ! ttask = 'd' AND intf2 ! action
(TRUE)
ELSE
sent msg (2) := TRUE
sent msg (2) := FALSE
msg (func) TO intf2 ! taskpid VIA rsd
sent msg (1) OR sent msg (2)
(TRUE)
ELSE
waiting intf answer

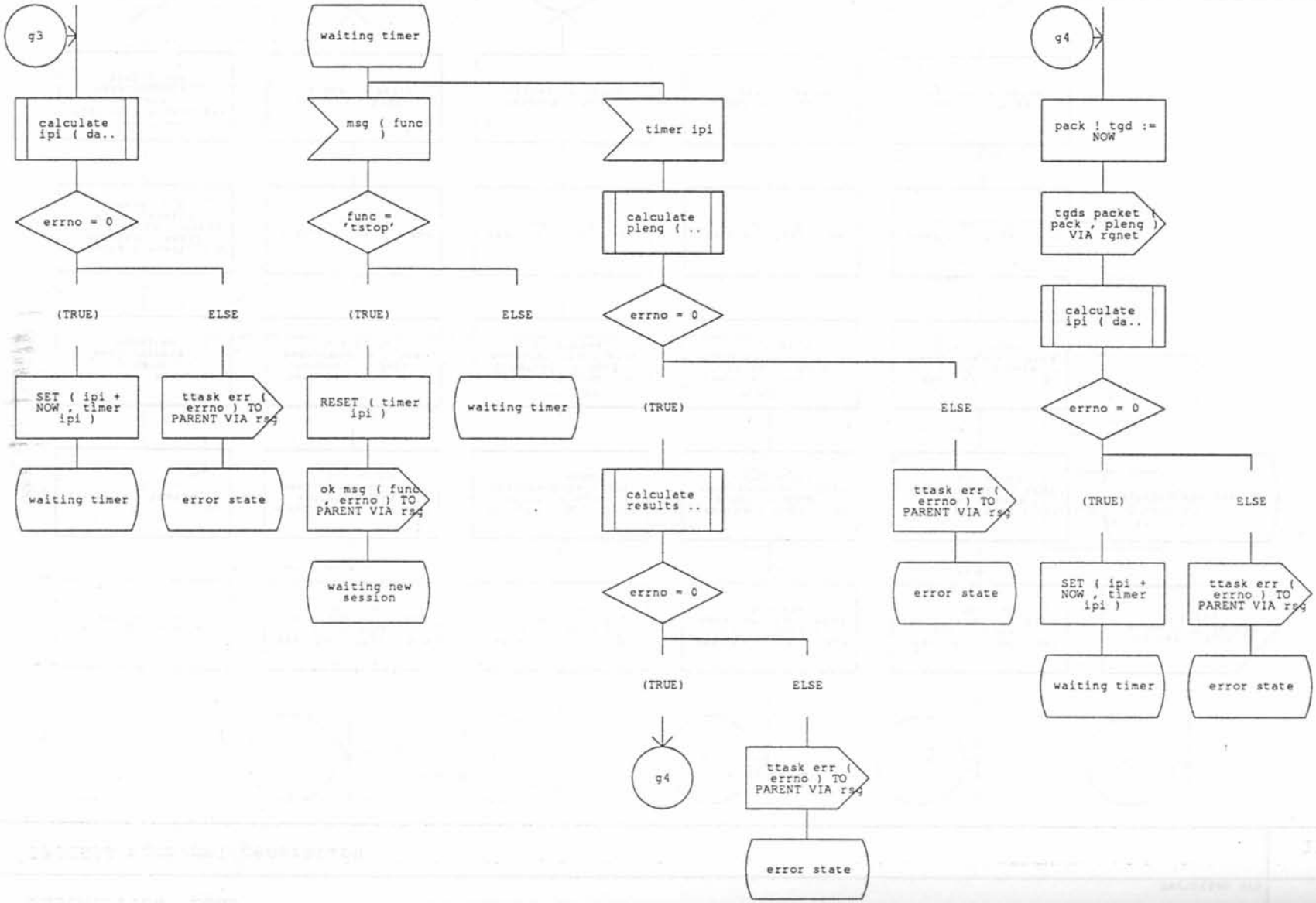

waiting intf answer
*
ok msg (ans func , err)
ans func = func
(TRUE)
ELSE
sw3
err := 6 + KERROR
sw3
SENDER = intf1 ! taskpid
(TRUE)
ELSE
rec msg (1)
(TRUE)
ELSE
err := 7 + KERROR
rec msg (1) := TRUE
SENDER = intf2 ! taskpid
(TRUE)
ELSE
rec msg (2)
(TRUE)
ELSE
err := 7 + KERROR
rec msg (2) := TRUE
err := 8 + KERROR
sw31
sw31
rec msg = sent msg
(TRUE)
ELSE
waiting intf answer

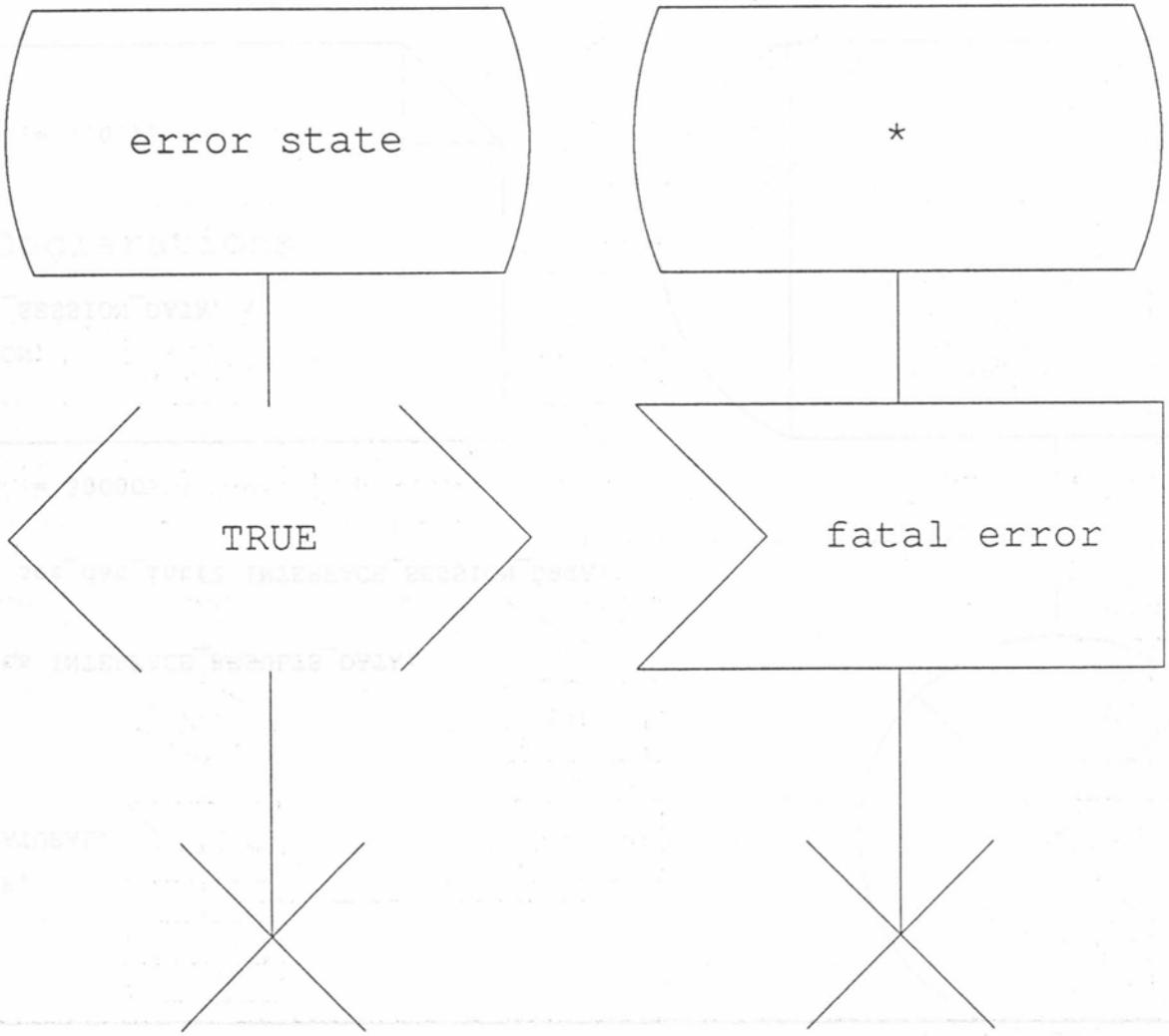
```
FPAR
    IN/OUT intf1, intf2 DATA_INTF,
    IN func STRG_ACTION,
    typ STRG_MAIN_ACT,
    IN/OUT err NATURAL;
DCL
    ans_func STRG_ACTION,
    sent_msg, rec_msg ACTIVE_INTF;
```


Pr Declarations
calculate ipi
calculate results
calculate pleng
errno := 0 , KERROR := KERROR + n pn * 1000 + n intf * 100
pack ! sa := ieee address
tgds announc (ann) VIA rgnet
ok announc (errno) TO PARENT VIA rsg
waiting new session


waiting new session
msg (func)
func
('tinit')
('tkilltask')
('tstart')
n intf
(1)
(2)
data := VIEW (ses dat intf1 , PARENT)
data := VIEW (ses dat intf2 , PARENT)
g1
g3
g1
data ! taskid = 'g'
(TRUE)
ELSE
g2
errno := 311
ok msg (func , errno) TO PARENT VIA rsg
error state

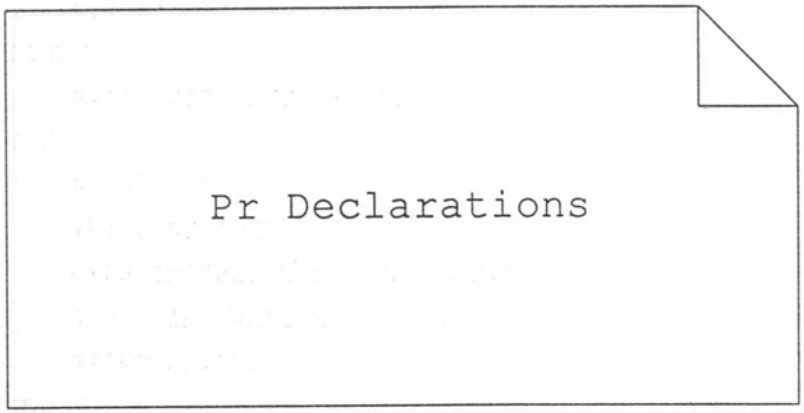

g2
g21
g22
g23
g24
g25
pack ! da := data ! src dst ieee
results ! htg1 ! min val := data ! measurements ! htg1 ! min val
results ! htg2 ! min val := data ! measurements ! htg2 ! min val
results ! htg3 ! min val := data ! measurements ! htg3 ! min val
results ! htg4 ! min val := data ! measurements ! htg4 ! min val
ok msg (func , errno) TO PARENT VIA rsg
results ! taskid := 'g'
results ! htg1 ! int width := data ! measurements ! htg1 ! int width
results ! htg2 ! int width := data ! measurements ! htg2 ! int width
results ! htg3 ! int width := data ! measurements ! htg3 ! int width
results ! htg4 ! int width := data ! measurements ! htg4 ! int width
waiting new session
results ! measures := data ! measurements ! measures
results ! htg1 ! n interval := data ! measurements ! htg1 ! n interval
results ! htg2 ! n interval := data ! measurements ! htg2 ! n interval
results ! htg3 ! n interval := data ! measurements ! htg3 ! n interval
results ! htg4 ! n interval := data ! measurements ! htg4 ! n interval
results ! meas1 := 0 , results ! meas2 := 0 , results ! meas3 := 0 , results ! meas4 := 0
results ! htg1 ! max val := 0
results ! htg2 ! max val := 0
results ! htg3 ! max val := 0
results ! htg4 ! max val := 0
results ! histograms := data ! measurements ! histograms
results ! htg1 ! values := zeros
results ! htg2 ! values := zeros
results ! htg3 ! values := zeros
results ! htg4 ! values := zeros
g21
g22
g23
g24
g25

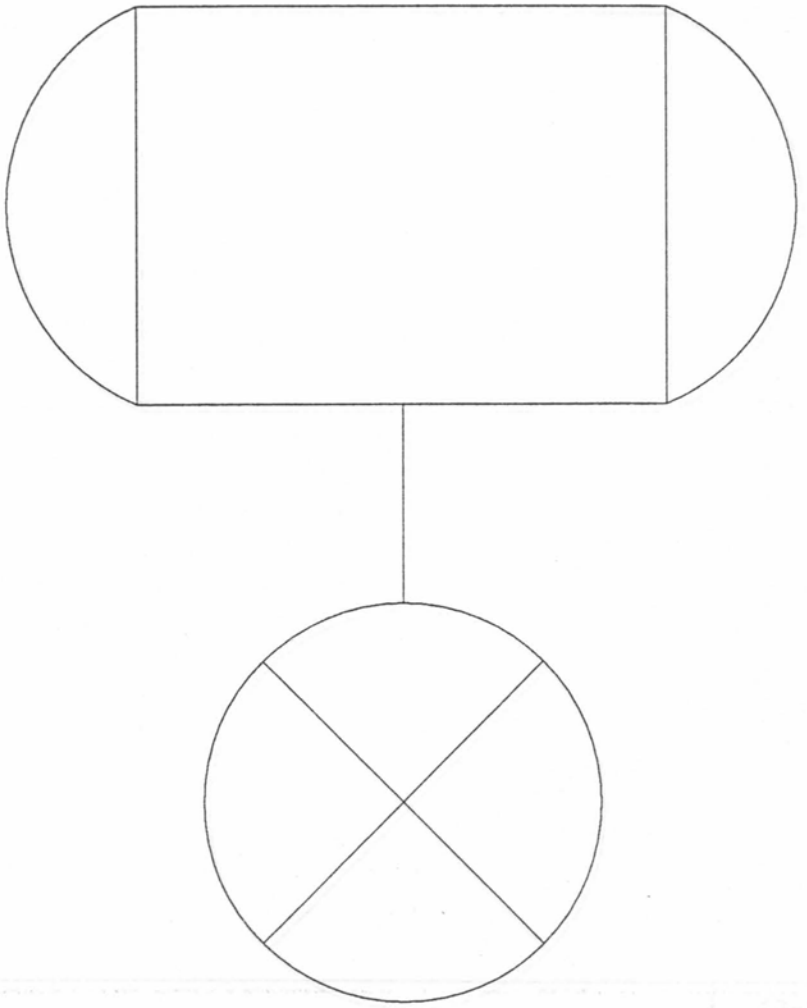

g3
calculate ipi (da..
errno = 0
(TRUE)
ELSE
SET (ipi + NOW , timer ipi)
waiting timer
ttask err (errno) TO PARENT VIA rsg
error state
waiting timer
msg (func)
func = 'tstop'
(TRUE)
ELSE
RESET (timer ipi)
ok msg (func , errno) TO PARENT VIA rsg
waiting new session
waiting timer
timer ipi
calculate pleng (..
errno = 0
(TRUE)
ELSE
calculate results ..
errno = 0
(TRUE)
ELSE
g4
ttask err (errno) TO PARENT VIA rsg
error state
ttask err (errno) TO PARENT VIA rsg
error state
g4
pack ! tgd := NOW
tgds packet (pack , pleng) VIA rgnet
calculate ipi (da..
errno = 0
(TRUE)
ELSE
SET (ipi + NOW , timer ipi)
waiting timer
ttask err (errno) TO PARENT VIA rsg
error state


error state
*
TRUE
fatal error

```
FPAR
    n_pn NAT_PN,
    n_intf NAT_INTF,
    ieee_address NATURAL,
    ann ANNOUNC,
    pack PACKET;
DCL
    REVEALED results INTERFACE_RESULTS_DATA;
VIEWED
    ses_dat_intf1, ses_dat_intf2 INTERFACE_SESSION_DATA;
DCL
    KERROR NATURAL := 30000;
DCL
    errno NATURAL,
    func STRG_ACTION,
    data INTERFACE_SESSION_DATA,
    ipi DURATION,
    pleng NATURAL;
DCL
    zeros HIST_VAL := (.0.);
TIMER
    timer_ipi;
```

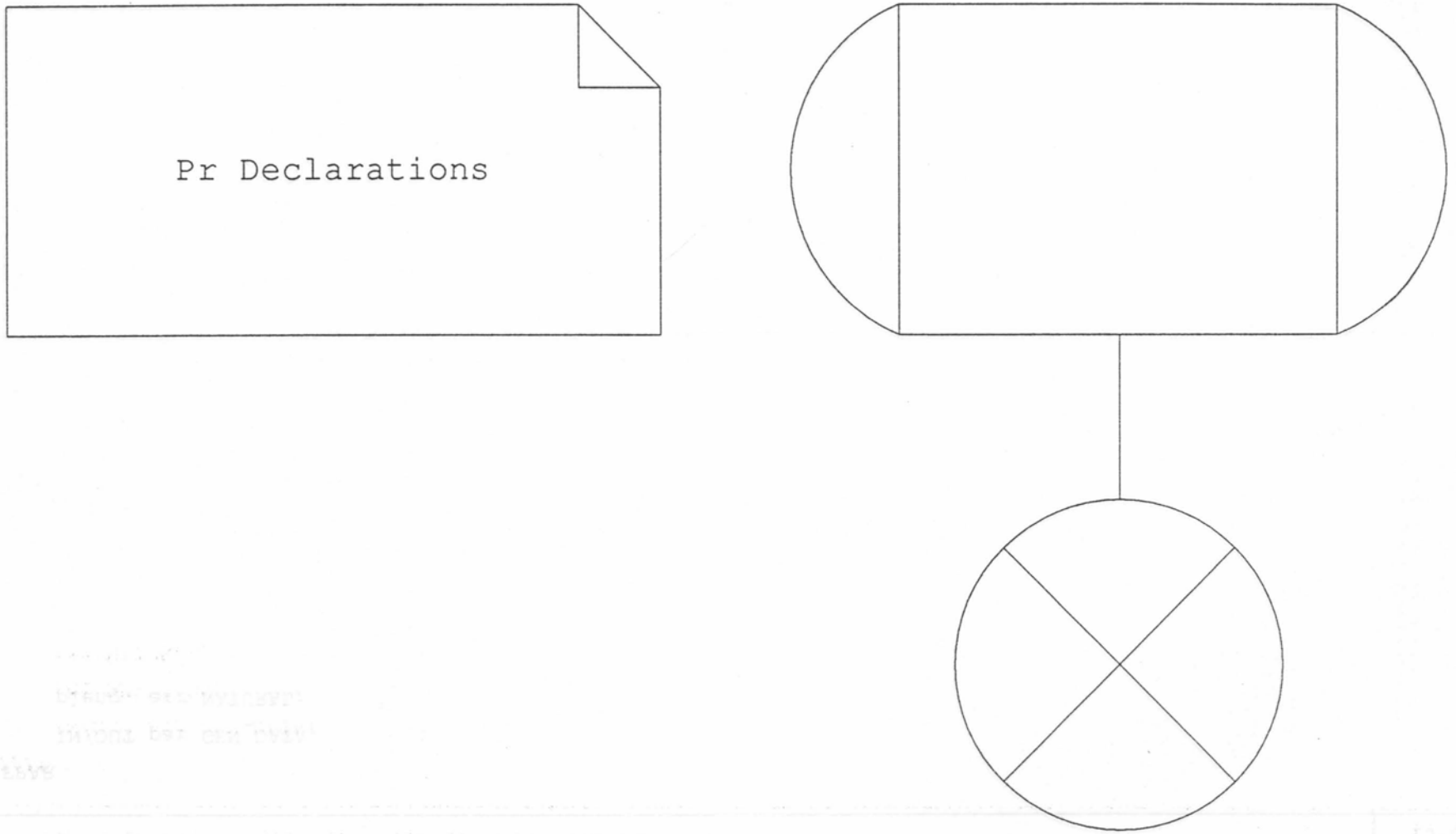
	DESCRIPTION: tgds Partition 1/1	Page: 69
1.3	PROCEDURE tgds/pn1/generation/calculate_ipi	13-Jan-1992



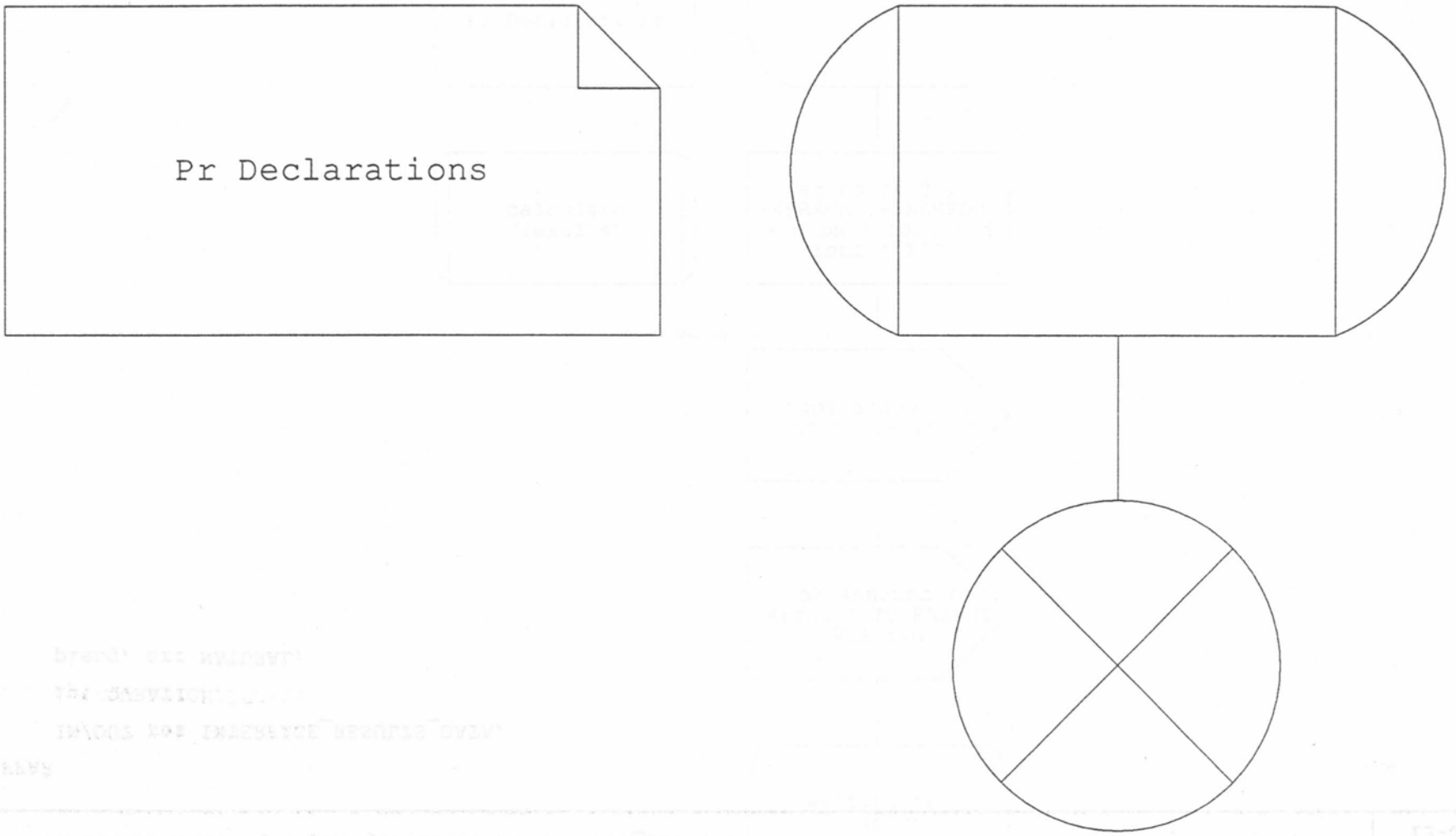


```
FPAR
    IN/OUT par GEN_DATA,
    ipi DURATION,
    err NATURAL;
```

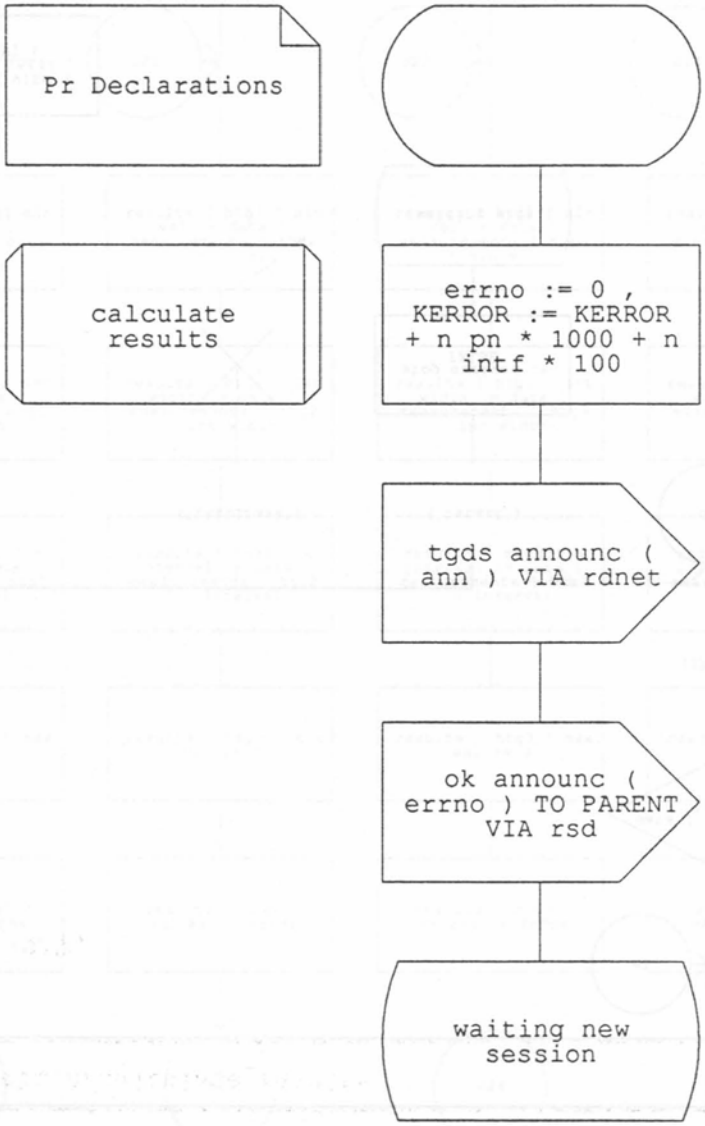
	DESCRIPTION: tgds Partition 1/1	Page: 71
1.3	PROCEDURE tgds/pn1/generation/calculate_pleng	13-Jan-1992

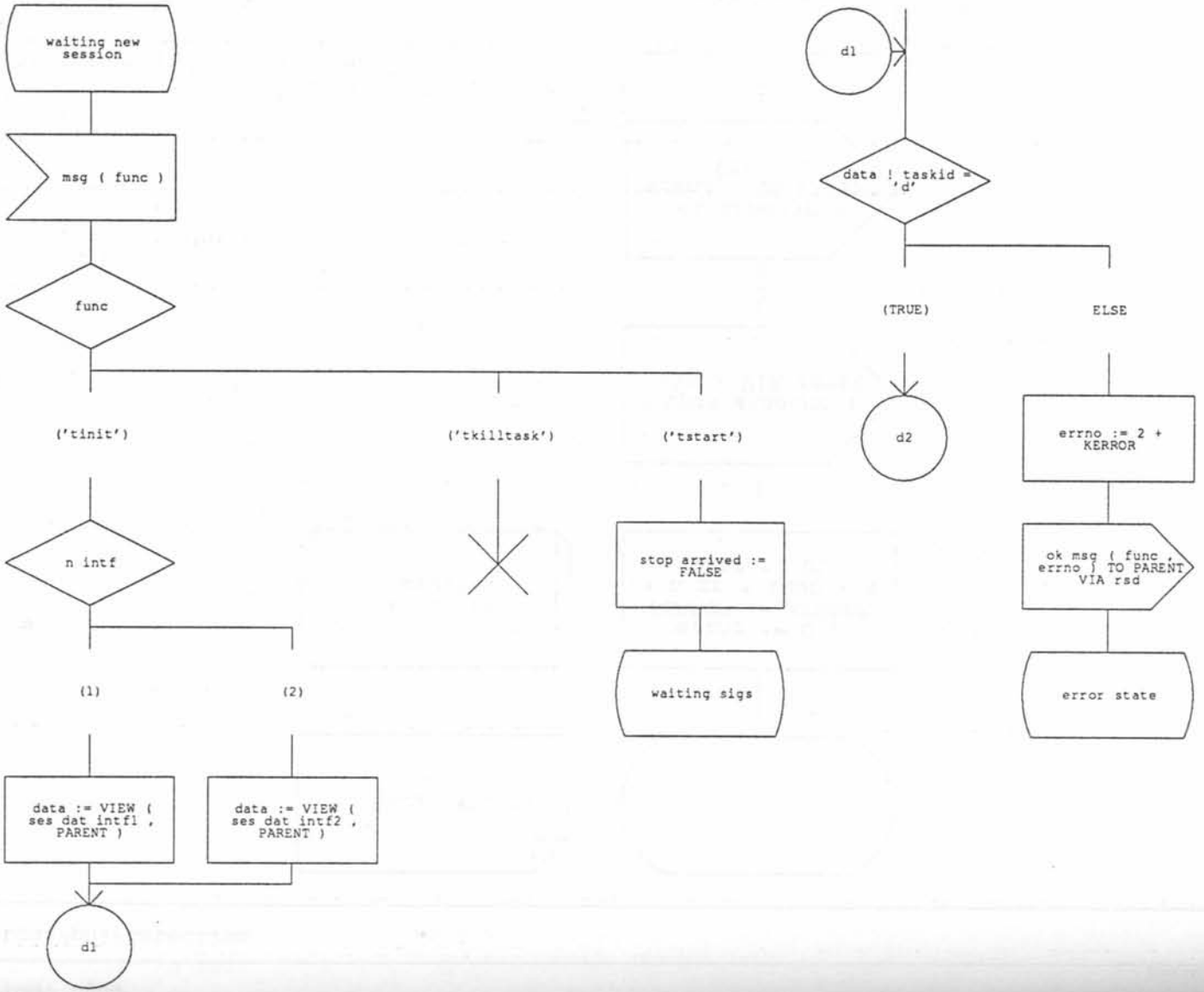


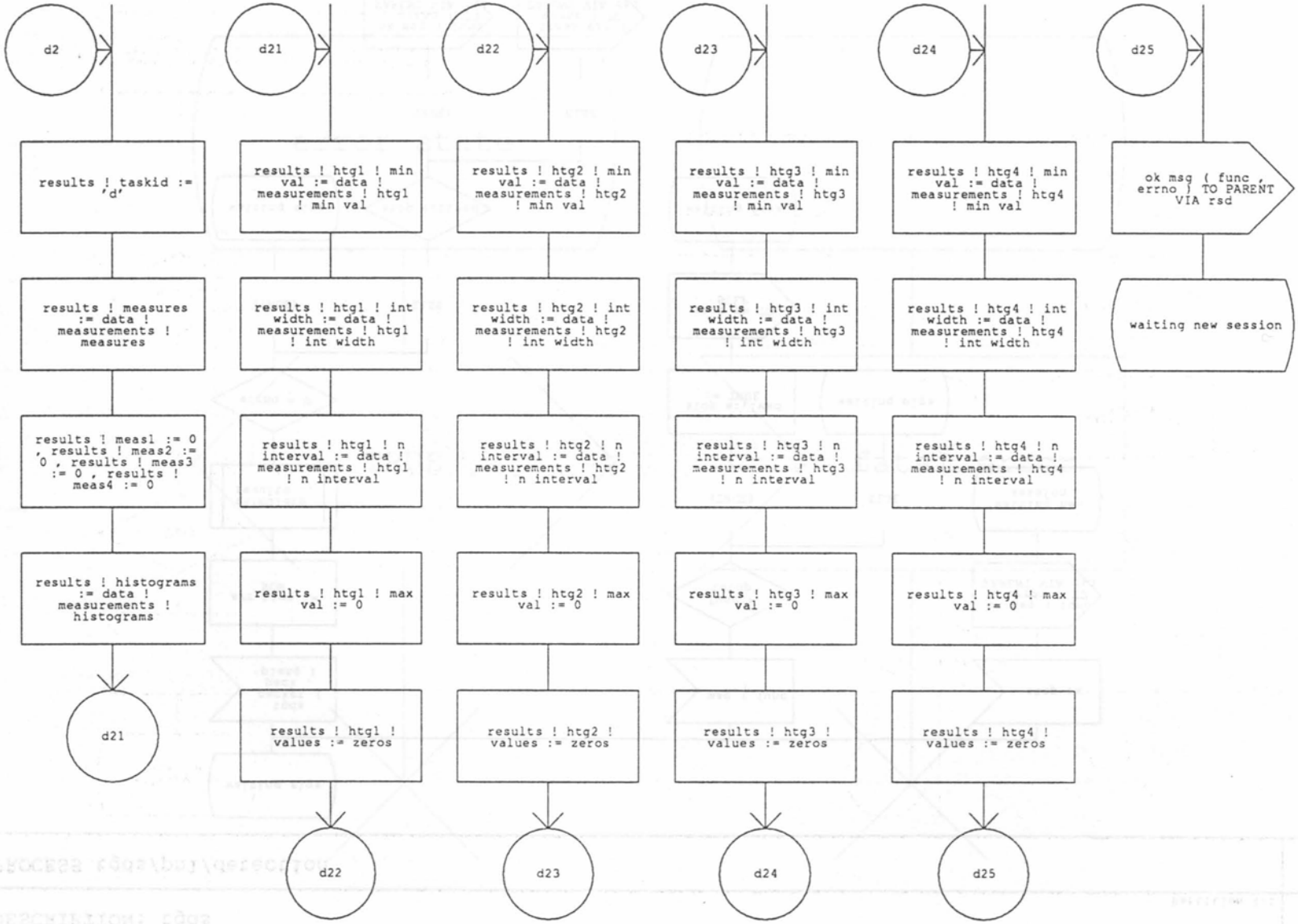
```
FPAR
    IN/OUT par GEN_DATA,
    pleng, err NATURAL;
```

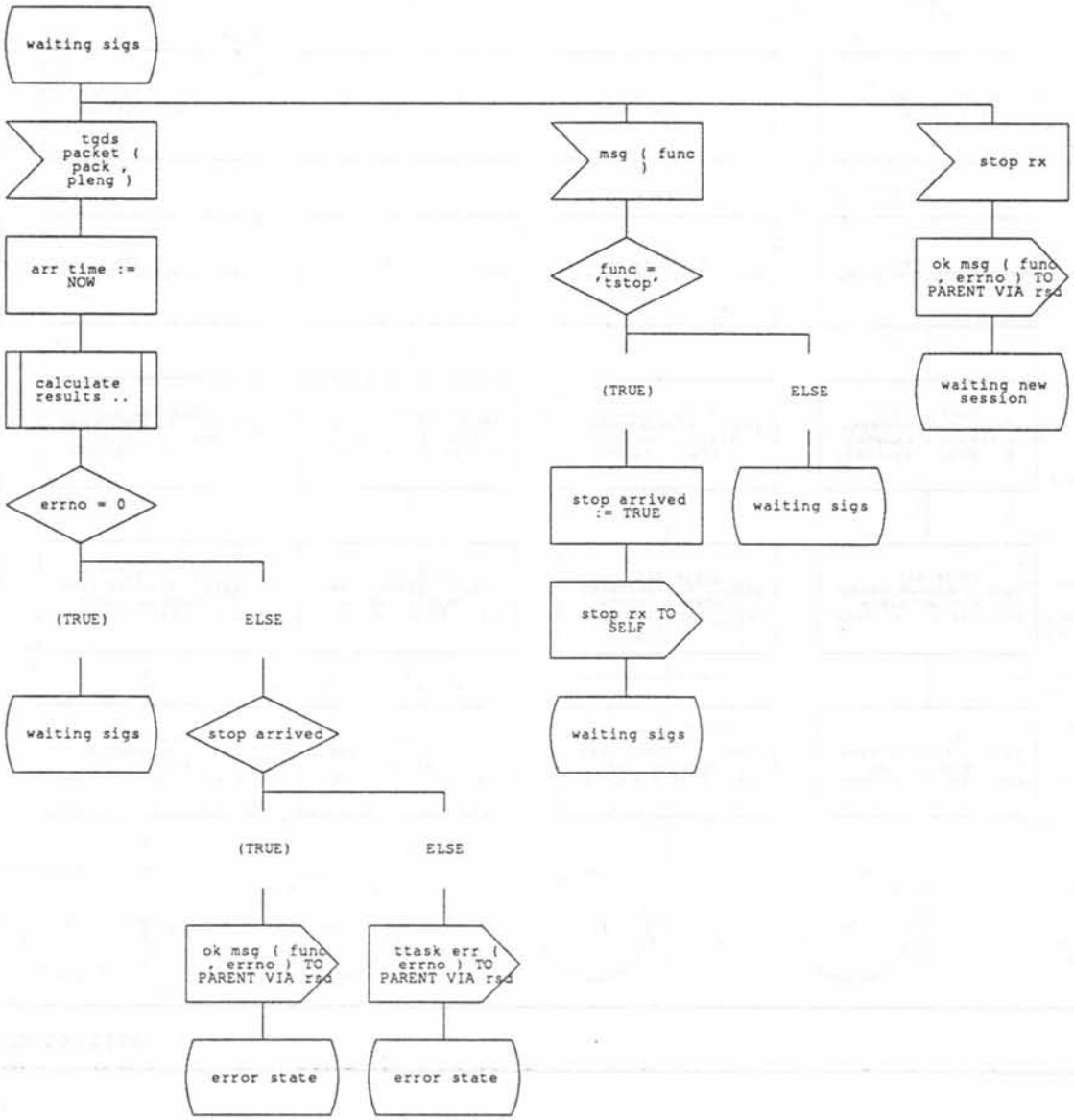

Pr Declarations

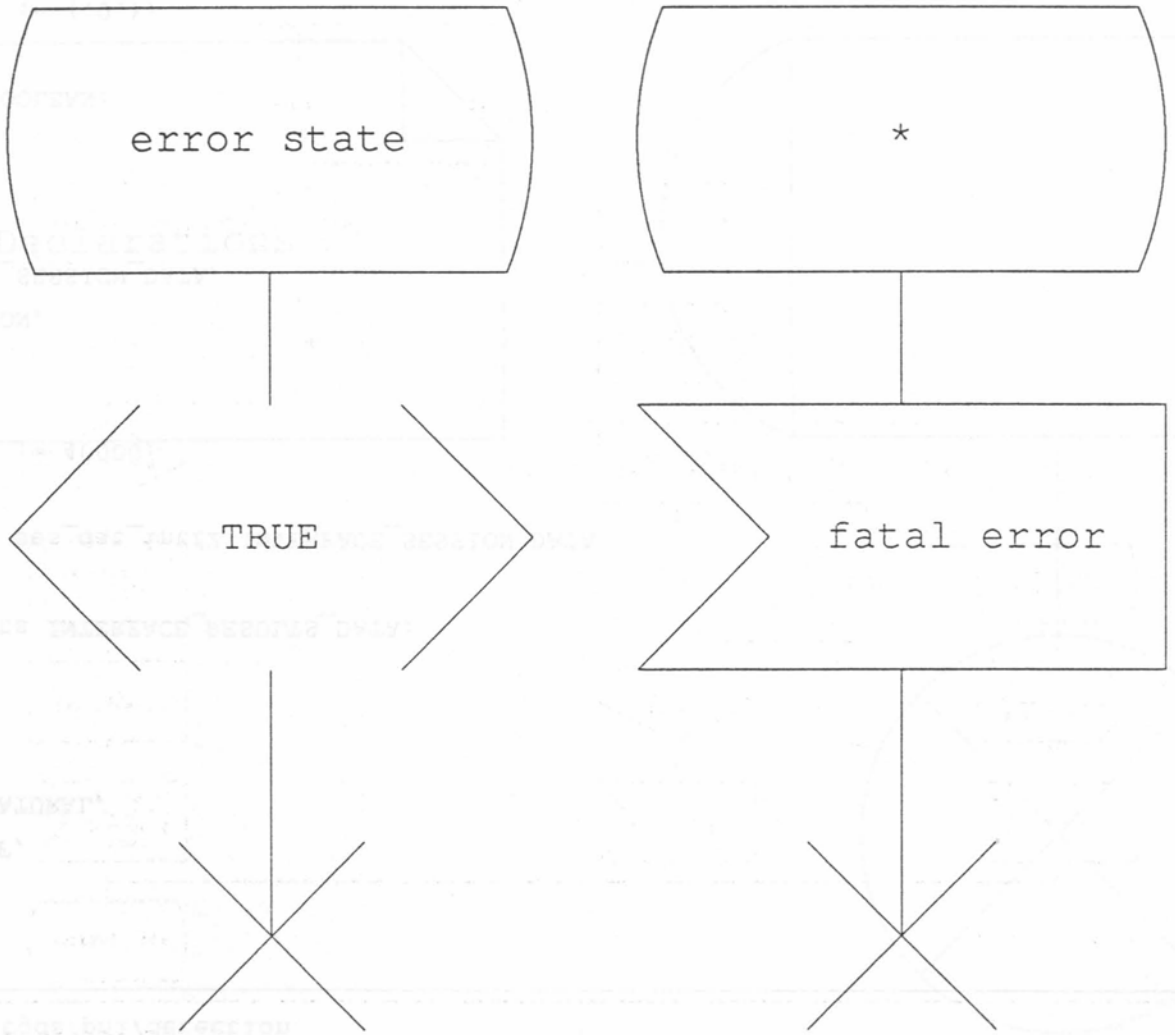
```
FPAR
    IN/OUT res INTERFACE_RESULTS_DATA,
    ipi DURATION,
    pleng, err NATURAL;
```


Pr Declarations
calculate results
errno := 0 , KERROR := KERROR + n pn * 1000 + n intf * 100
tgds announc (ann) VIA rdnet
ok announc (errno) TO PARENT VIA rsd
waiting new session

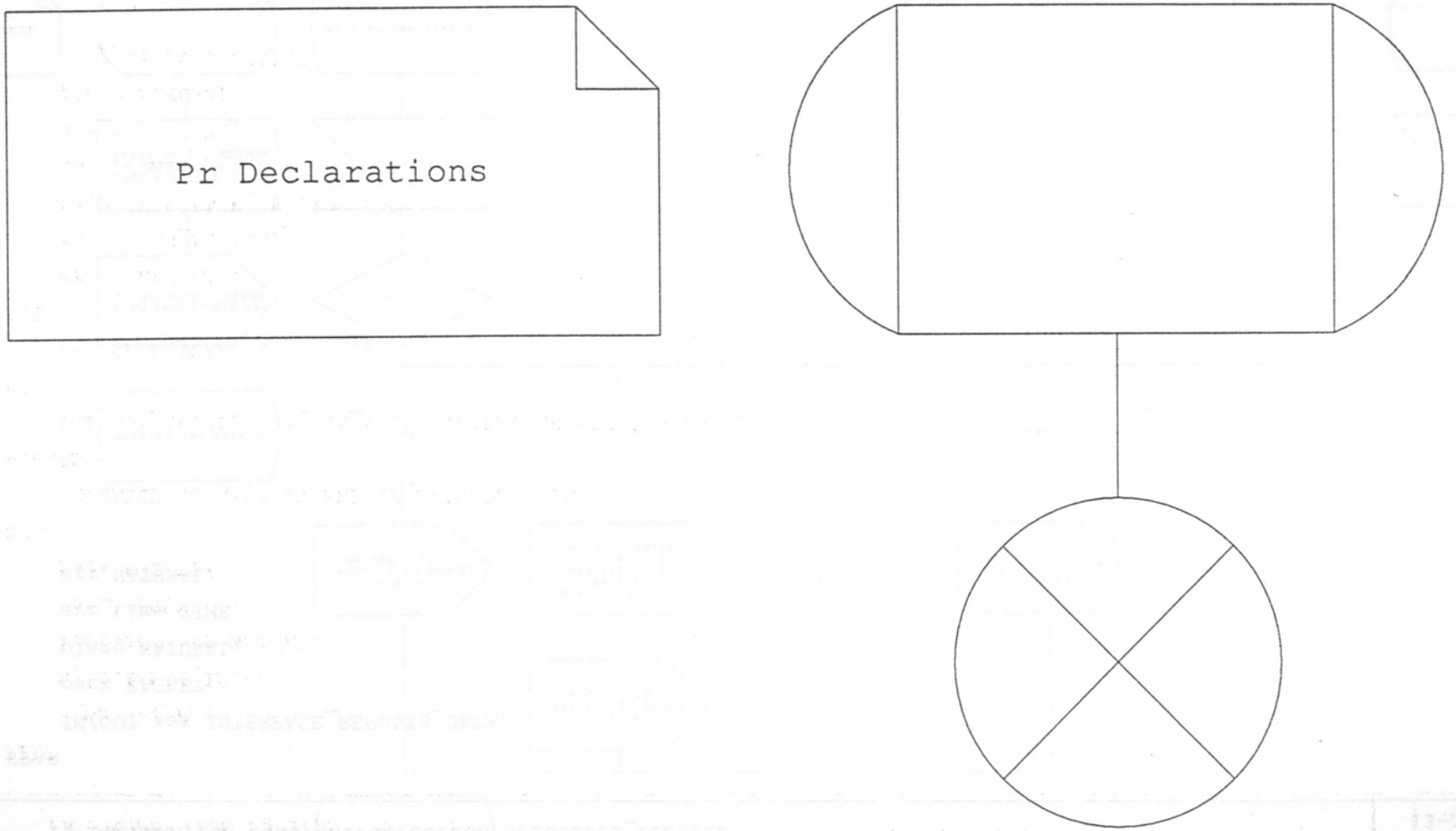

waiting new session
msg (func)
func
('tinit')
('tkilltask')
('tstart')
n intf
(1)
(2)
data := VIEW (ses dat intf1 , PARENT)
data := VIEW (ses dat intf2 , PARENT)
d1
stop arrived := FALSE
waiting sigs
d1
data ! taskid = 'd'
(TRUE)
ELSE
d2
errno := 2 + KERROR
ok msg (func , errno) TO PARENT VIA rsd
error state


d2
d21
d22
d23
d24
d25
results ! taskid := 'd'
results ! htg1 ! min val := data ! measurements ! htg1 ! min val
results ! htg2 ! min val := data ! measurements ! htg2 ! min val
results ! htg3 ! min val := data ! measurements ! htg3 ! min val
results ! htg4 ! min val := data ! measurements ! htg4 ! min val
ok msg (func , errno) TO PARENT VIA rsd
results ! measures := data ! measurements ! measures
results ! htg1 ! int width := data ! measurements ! htg1 ! int width
results ! htg2 ! int width := data ! measurements ! htg2 ! int width
results ! htg3 ! int width := data ! measurements ! htg3 ! int width
results ! htg4 ! int width := data ! measurements ! htg4 ! int width
waiting new session
results ! meas1 := 0 , results ! meas2 := 0 , results ! meas3 := 0 , results ! meas4 := 0
results ! htg1 ! n interval := data ! measurements ! htg1 ! n interval
results ! htg2 ! n interval := data ! measurements ! htg2 ! n interval
results ! htg3 ! n interval := data ! measurements ! htg3 ! n interval
results ! htg4 ! n interval := data ! measurements ! htg4 ! n interval
results ! histograms := data ! measurements ! histograms
results ! htg1 ! max val := 0
results ! htg2 ! max val := 0
results ! htg3 ! max val := 0
results ! htg4 ! max val := 0
d21
results ! htg1 ! values := zeros
results ! htg2 ! values := zeros
results ! htg3 ! values := zeros
results ! htg4 ! values := zeros
d22
d23
d24
d25

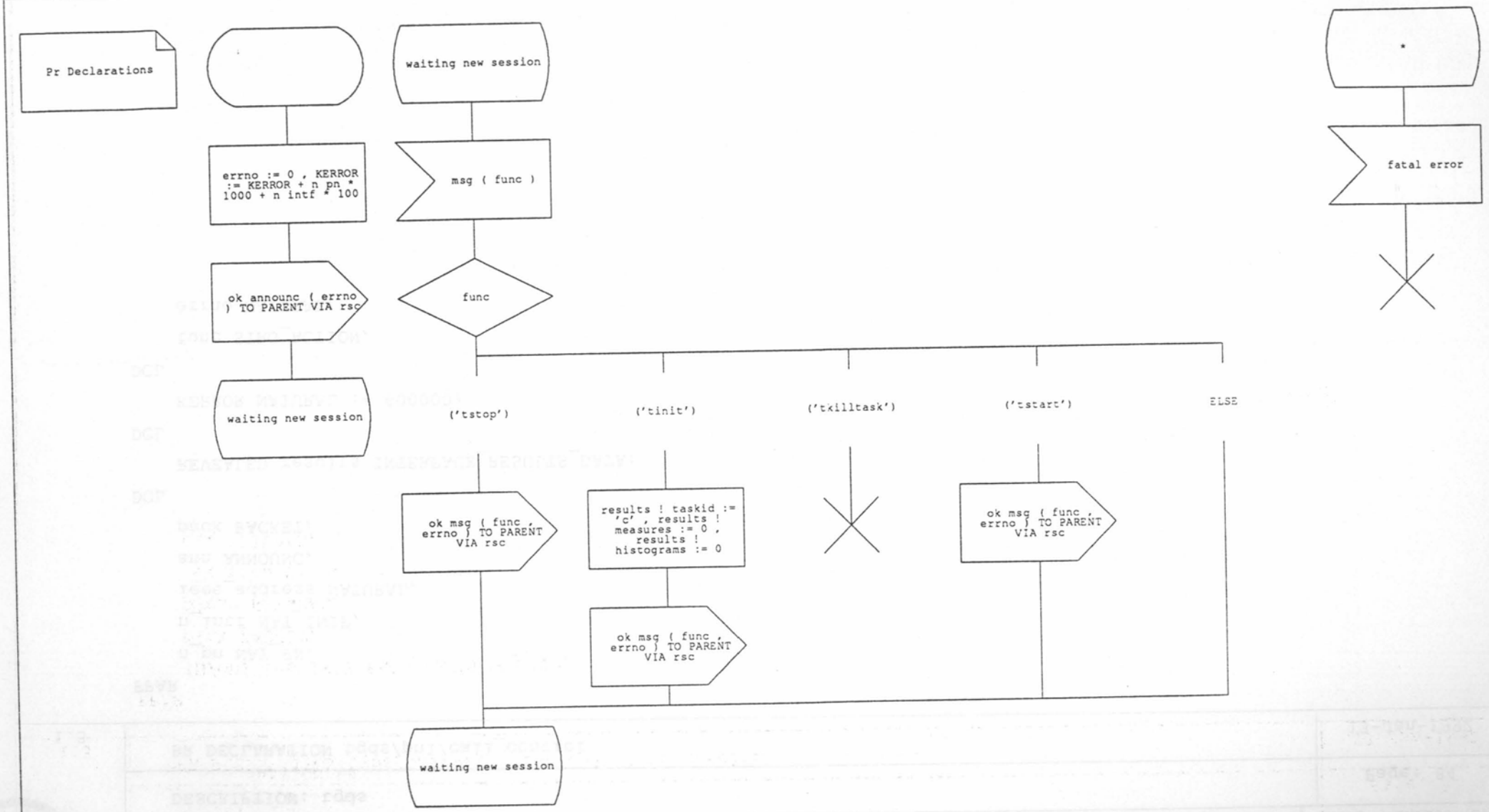

waiting sigs
tgds packet (pack , pleng)
arr time := NOW
calculate results ..
errno = 0
(TRUE)
ELSE
waiting sigs
stop arrived
(TRUE)
ELSE
ok msg (func , errno) TO PARENT VIA rsd
error state
ttask err (errno) TO PARENT VIA rsd
error state
msg (func)
func = 'tstop'
(TRUE)
ELSE
stop arrived := TRUE
waiting sigs
stop rx TO SELF
waiting sigs
stop rx
ok msg (func , errno) TO PARENT VIA rsd
waiting new session


error state
*
TRUE
fatal error

```
FPAR
    n_pn NAT_PN,
    n_intf NAT_INTF,
    ieee_address NATURAL,
    ann ANNOUNC,
    pack PACKET;
DCL
    REVEALED results INTERFACE_RESULTS_DATA;
VIEWED
    ses_dat_intf1, ses_dat_intf2 INTERFACE_SESSION_DATA;
DCL
    KERROR NATURAL := 40000;
DCL
    errno NATURAL,
    func STRG_ACTION,
    data INTERFACE_SESSION_DATA,
    ipi DURATION,
    arr_time TIME,
    pleng NATURAL,
    stop_arrived BOOLEAN;
DCL
    zeros HIST_VAL := (.0.);
```


Pr Declarations

```
FPAR
    IN/OUT res INTERFACE_RESULTS_DATA,
    pack PACKET,
    pleng NATURAL,
    arr_time TIME,
    err NATURAL;
```


Pr Declarations
errno := 0 , KERROR := KERROR + n pn * 1000 + n intf * 100
ok announc (errno) TO PARENT VIA rsc
waiting new session
waiting new session
msg (func)
func
('tstop')
('tinit')
('tkilltask')
('tstart')
ELSE
ok msg (func , errno) TO PARENT VIA rsc
results ! taskid := 'c' , results ! measures := 0 , results ! histograms := 0
ok msg (func , errno) TO PARENT VIA rsc
ok msg (func , errno) TO PARENT VIA rsc
waiting new session
*
fatal error

```
FPAR
    n_pn NAT_PN,
    n_intf NAT_INTF,
    ieee_address NATURAL,
    ann ANNOUNC,
    pack PACKET;
DCL
    REVEALED results INTERFACE_RESULTS_DATA;
DCL
    KERROR NATURAL := 600000;
DCL
    func STRG_ACTION,
    errno NATURAL;
```

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBLIOTECA
0000034781