IMPLEMENTAÇÃO DO PROTOCOLO X-25 NUM CONCENTRADOR DE COMUNICAÇÕES BASEADO NO 8088

EDMUNDO ROBERTO MAURO MADEIRA

UNICAMP

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E CIÊNCIA DA COMPUTAÇÃO

CAMPINAS - SÃO PAULO
BRASIL

IMPLEMENTAÇÃO DO PROTOCOLO X-25 NUM CONCENTRADOR DE COMUNICAÇÕES BASEADO NO 8088

EDMUNDO ROBERTO MAURO MADEIRA

IMPLEMENTAÇÃO DO PROTOCOLO X-25 NUM CONCENTRADOR DE COMUNICAÇÕES BASEADO NO 8088

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida pelo Sr. Edmundo Roberto Mauro Madeira e aprovada pela comissão julgadora.

Campinas, de de 1985.

Prof. Dr. Célio C. Guimarães
Orientador

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Março 1985

UNICAMP
BIBLIOTECA CENTRAL

AGRADECIMENTOS

Ao Prof. Dr. Célio Cardoso Guimarães pela orientação eficiente e dedicada.

Aos professores do Departamento de Ciência da Computação que sempre me apoiaram e incentivaram. Em especial ao Prof. Dr. Nelson C. Machado pelo uso do Laboratório de Microprocessadores do Centro de Computação.

A todos os meus familiares e amigos que me motivaram para esse estudo de Pós-Graduação.

À Lourdes pelo incansável serviço de datilografia.

À FAPESP e ao CNPq pelo apoio financeiro.

Aos meus pais,
Oswaldo e Haydée.

SUMMARY

This work presents an implementation of the X-25 communication protocol, which is defined for the CCITT Recommendation (Committee Consultative International Telegraph and Telephone), on an Intel 8088 microprocessador |20|, |23| and |36|. The proposed system is based on message exchange.

The microprocessor has the function of a concentrator linked to a packet switching network. It will communicate on one side with a network node and on other side with several terminals.

The configuration mode between the concentrator and the network node is balanced aysnchronous.

As an objective, the levels 1 and 2 change through hardware and software a not very reliable point-to-point link into a reliable link. As an second objective, participants in the network exchange traffic information to avoid congestion.

The level 3 executes multiplexing to change a single level 2 logical circuit into several independent logical channels.

At the level 4 (transport level) some end-to-end control functions are implemented.

The levels 2, 3 and 4, in addition to a Manager (Supervisor)

In Chapter III, we present a brief discussion about concurrent processing languages. Where appropriate, comments on Módula 2, the chosen language for the implementation, were included. Communication and synchronization between concurrent processes using monitors and message exchange are presented. Examples of the X-25 protocol implementation using this two models are shown [3], [8] and [16].

In Chapter IV, which is the thesis body, we treat the implementation on the 8088 microprocessor. The system is described as a whole and analysed with respect to the following important details: data structures, process communication, non-interrupt modules, dynamic memory alocation, timers, HDLC 8273 interface, etc.

In Chapter V, we describe the debugging tests made on the system in a stepwise manner and the conclusion of thesis along with several suggestions were presented in Chapter VI.

The main contributions of this thesis were:

- Use of formal protocol analysis in a practical implementation of the X-25 protocol;

- An implementation methodology using message exchange between processes representing the protocol levels and explicit process control transference using the tools implemented in Modula 2.

ÍNDICE

INTRODUÇÃO i

CAPÍTULO I - PROTOCOLOS DE COMUNICAÇÕES E PROTOCOLO X-25 . . 1

1.1 - Protocolos e arquitetura da ISO 1
1.2 - Protocolo X-25 5
1.2.1 - Nível de ligação (nível 2) 5
1.2.2 - Nível de pacotes (nível 3) 14
1.2.3 - Especificação do protocolo X-25 22
1.2.3.1 - Nível 2 22
1.2.3.2 - Nível 3 27
1.2.3.3 - Nível 4 30

CAPÍTULO II - ANÁLISE DO PROTOCOLO 38

2.1 - Árvore de perturbação 38
2.2 - Nível 2 47
2.3 - Nível 4 56

CAPÍTULO III - COMUNICAÇÃO E SINCRONIZAÇÃO DE PROCESSOS . . . 62

3.1 - Introdução 62
3.2 - Linguagens para programação concorrente 62
3.3 - Monitor × Troca de Mensagens 64
3.4 - Implementações do protocolo X-25 67
3.5 - Linguagem de implementação-Módula 2 72

CAPÍTULO IV - IMPLEMENTAÇÃO DO PROTOCOLO X-25 NO MICROPROCESSADOR 8088 76

4.1 - Sistema proposto 76
4.2 - Estrutura de dados - Filas 84
4.3 - Características da implementação 89
4.4 - Alocação dinâmica de memória 96

4.5 - Temporizações 99
4.6 - Descrição do sistema 100
4.7 - Interface HDLC 8273 Intel 101

CAPÍTULO V - TESTE DO SISTEMA 104

5.1 - Testes 104
5.2 - Mapa de Memória 107

CAPÍTULO VI - CONCLUSÕES E SUGESTÕES 110

BIBLIOGRAFIA 114

APÊNDICE 1 - ESQUEMA GERAL E TRECHOS DE PROCEDIMENTOS 123

APÊNDICE 2 - ALOCAÇÃO DINÂMICA DE MEMÓRIA 132

INTRODUÇÃO

Neste trabalho, implementamos o protocolo de comunicações X-25 definido pela recomendação do CCITT (Committee Consultative International Telegraph and Telephone) num microprocessador 8088 da Intel. Este protocolo é definido para os níveis 1, 2 e 3. O microprocessador poderá exercer a função de um concentrador ligado a uma rede de comutação de pacotes comunicando-se de um lado com um nó da rede e de outro lado com vários terminais. O modo de configuração entre o concentrador e o nó da rede é o balanceado assíncrono. Os níveis 1 e 2 tem o objetivo de transformar através de software e hardware uma ligação ponto-a-ponto "pouco" confiável, com tendência a erros de transmissão, num meio de comunicação confiável, ou seja, relativamente isento de erros, que transferem dados entre si, e permitir que ambas as partes troquem informações de fluxo que evitem congestionamento. O nível 3 executa uma multiplexação que transforma um único circuito lógico do nível 2 em vários canais logicamente independentes. Neste sistema, um nível 4 (nível de transporte) é introduzido para realizar algumas funções de controle fim-a-fim e para a ligação física entre o microprocessador e a rede é utilizada a interface HDLC 8273. Os níveis 2, 3 e 4 além de um gerenciador e as rotinas de interrupção de recepção e transmissão da linha X-25 são tratados como processos concorrentes. O sistema proposto é baseado em troca de mensagens.

Utilizamos também neste trabalho uma técnica para deteção de erros em protocolos, tais como: deadlocks, recepções não especificadas, interações não executáveis e ambiguidades de estado. Através desta técnica fizemos uma análise das especificações dos diversos níveis.

O protótipo desenvolvido para esta tese possui duas interfaces HDLC 8273 e 32K de memória. Os programas executados no 8088 são carregados remotamente via sistema DEC-10 do Centro de Computação.

No Capítulo I descrevemos didaticamente a arquitetura da ISO e o protocolo X-25. Especificamos o nível 4 para o propósito desta tese. Diagramas de estados para os diversos níveis foram constituídos: nível 2 - a partir do texto da especificação do protocolo X-25; nível 3 - extraído da especificação X-25 e nível 4 - a partir da especificação desta tese.

No Capítulo II analisamos formalmente o protocolo X-25 para previní-lo da existência de erros. A análise é feita a partir dos diagramas de estados definidos no Capítulo I.

No Capítulo III apresentamos uma breve discussão sobre linguagens para processamento concorrente Módula 2, a linguagem escolhida para a implementação, é comentada. Comunicação e sincronização de processos através de monitores e troca de mensagens são apresentadas. Exemplos de implementações do protocolo X-25 utilizando esses dois modelos são dados.

No Capítulo IV, que é o corpo da tese, tratamos da implementação no microprocessador 8088. O sistema é descrito como um todo e analisado em alguns detalhes: estrutura de dados, módulos não interrompíveis, alocação dinâmica de memória, temporizações, interface HDLC 8273, etc.

No Capítulo V descrevemos os testes realizados no sistema, nas diversas etapas de depuração do mesmo.

As conclusões e sugestões desta tese podem ser vistas no Capítulo VI.

As principais contribuições desta tese foram:

- A análise formal do protocolo em seus diversos níveis apresentada no Capítulo II.

- A metodologia da implementação através de trocas de mensagens entre processos representando os níveis de protocolo e transferência explícita de controle de um processo para outro através do mecanismo implementado em Módula 2 (Capítulo IV).

CAPÍTULO I

PROTOCOLOS DE COMUNICAÇÕES E PROTOCOLO X-25

1.1. PROTOCOLOS E ARQUITETURA DA ISO.

Por protocolo entende-se um conjunto de procedimentos para troca de mensagens entre processos que estão sendo executados em máquinas distintas, em geral fisicamente distantes entre si. Protocolos são organizados de maneira hierárquica em forma de níveis de abstração.

A ISO (Internation Standards Organization) define uma arquitetura para redes composta de sete camadas (ou níveis) mostrada na figura 1.1.

Na figura 1.1 as caixas marcadas N1, N2,...,N6,N7 à esquerda correspondem por exemplo a um nó da rede, enquanto as caixas à direita correspondem a um outro nó ou a um computador hospedeiro.

É importante observar na figura que fisicamente a informação segue as setas cheias, enquanto que logicamente segue as setas tracejadas.

Dependendo da complexidade e da faixa de passagem do sistema, os níveis do protocolo podem ser implementados em um ou mais processadores. Nesta proposta considera-se que os níveis 1, 2, 3

e 4 serão implementados em um único processador.

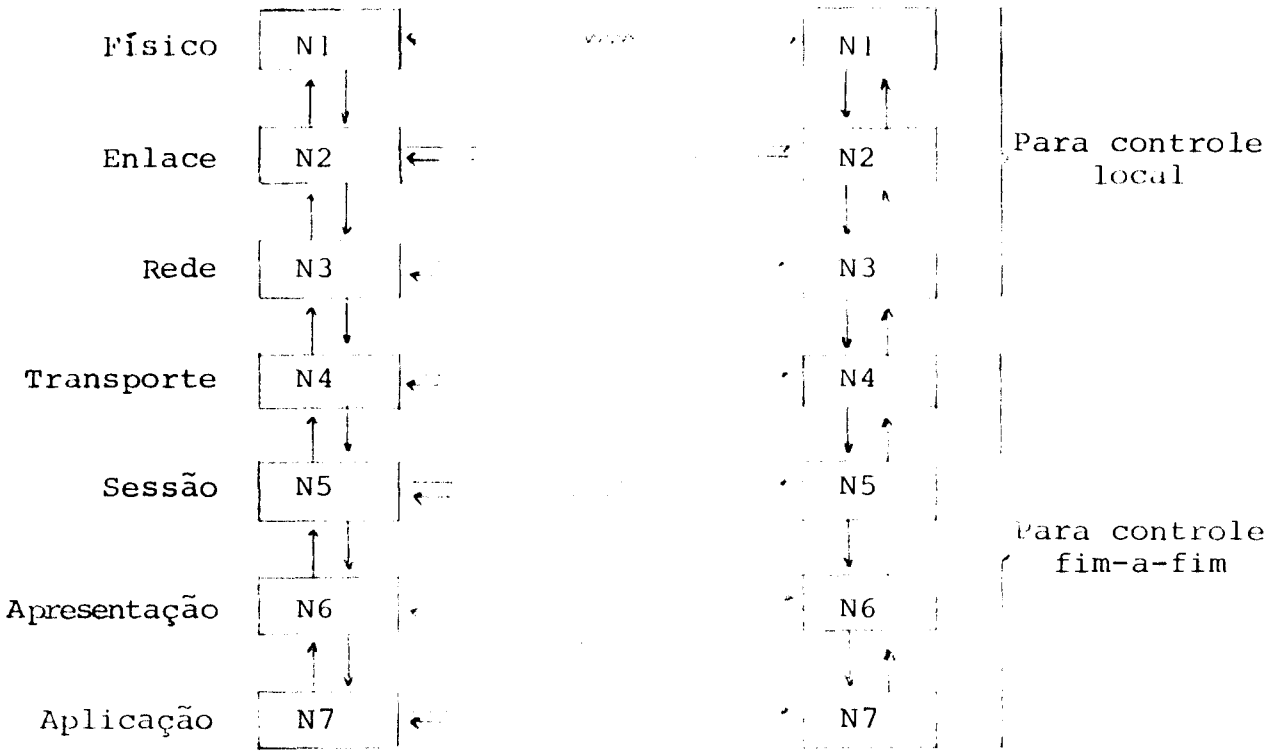


Figura 1.1

As funções dos 7 níveis definidos pela ISO são:

O nível 1 (físico) é o nível que controla a transmissão física dos quadros enviados pelo nível 2. Ex.: voltagem, se a transmissão é full duplex ou não, etc.

No nível 2 (enlace) transmite-se "frames" (que traduziremos por quadros). Um quadro é um conjunto de bytes de 8 bits. Conjuntamente

com o nível 1 transforma o meio de comunicação "pouco confiável" num meio de comunicação "muito confiável".

No nível 3 (rede) transmite-se pacotes. O conteúdo de um quadro de informação do nível 2 é um pacote. Este nível controla o encaminhamento dos pacotes através da rede e portanto também funciona como controlador de congestionamento.

O nível 4 (transporte) é o "primeiro" nível de controle fim-a-fim, isto é, permite a transferência controlada de informação entre 2 processos em computadores hospedeiros distintos. Pode, ou não, usar multiplexação de canais.

O nível 5 (sessão) permite a um usuário acessar a rede, estabelecendo uma conexão com outro usuário. O usuário deve fornecer o endereço de quem ele quer se conectar, parâmetros da transferência, etc.

O nível 6 (apresentação) realiza a conversão de código de representação dos dados que estão sendo transmitidos em uma sessão.

O nível 7 (aplicação) é o próprio programa de aplicação.

O protocolo X-25 define apenas os três primeiros níveis e nessa implementação trataremos somente dos quatro primeiros níveis obtendo a organização da figura 1.2.

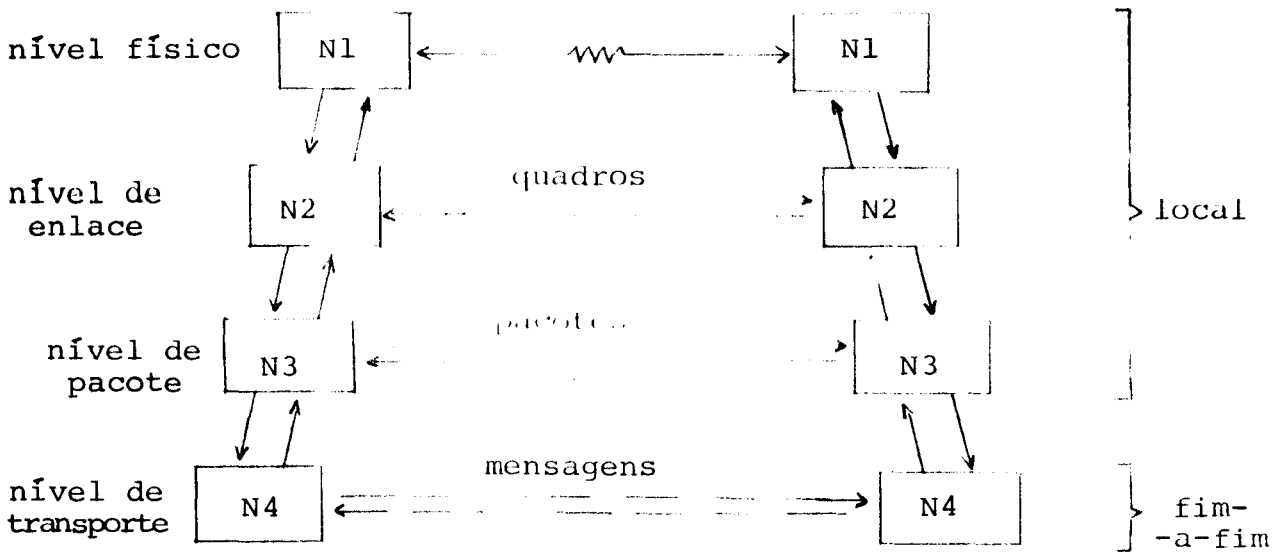


Figura 1.2

O nível 3 ao receber uma mensagem do nível 4, coloca um cabeçalho com informação do nível 3, e envia este pacote (mensagem do nível 4 + cabeçalho) para o nível 2. O nível 2 ao receber um pacote do nível 3, coloca um cabaçalho com informação do nível 2, e envia este quadro (pacote do nível 3 + cabeçalho) para o nível 1. O nível 1 controla a transmissão física do quadro. Na outra parte, o procedimento inverso é executado, em que um nível retira o cabeçalho correspondente (obtendo as informações necessárias para o processamento neste nível) e passa o restante para o nível seguinte. (ver figura 1.2).

1.2. PROTOCOLO X-25.

Nesta seção descrevemos alguns conceitos da Recomendação X-25 necessários para a compreensão de nossa proposta, já considerando algumas características da implementação. Para um entendimento total do protocolo, recomendamos a leitura de [20].

O protocolo X-25, padronizado pelo CCITT, define três níveis: nível físico, nível de enlace e nível de pacote, que correspondem respectivamente aos níveis 1, 2 e 3 da ISO descritos anteriormente.

Segundo esta Recomendação, o nível físico deve estar de acordo com a Recomendação X-21, e esta define as características físicas da ligação.

Introduzimos inicialmente dois termos utilizados nesta recomendação:

- DTE (Data Terminal Equipament) é o computador hospedeiro (host, concentrador de terminais) que se liga a um só nó na rede.

- DCE (Data Circuit Terminating Equipament) é o nó da rede, que geralmente está conectado com um ou mais DTE's, e que necessariamente está conectado com um ou mais DCE's.

1.2.1. NÍVEL DE ENLACE (NÍVEL 2).

A comunicação neste nível é realizada através de quadros.

A figura 1.3 apresenta o formato destes quadros.

flag	endereço	controle	inform.	FCS	flag
01111110	8 bits	8 bits	N bits	16 bits	01111110

Figura 1.3

onde flag: byte pré-definido (01111110) que indica início ou fim de quadro.

endereço: igual a 3 ou 1 como veremos adiante.

controle: byte que define o tipo do quadro e que contém informações relativas ao controle de fluxo.

informação: apenas em quadros de informação

FCS: frame checking sequence

O transmissor deve poder enviar qualquer sequência de bits (informação), inclusive um byte igual ao byte flag pré-estabelecido.

Analogamente o receptor deve também poder receber qualquer sequência de bits (informação), inclusive o mencionado acima.

Portanto, devemos ter algum mecanismo para que um byte igual ao "flag" não seja interpretado como final de quadro no receptor.

Portanto, o protocolo deve ser transparente à ocorrência desse "flag" dentro do quadro.

Este problema de transparência (ocorrência de um flag

01111110 dentro do quadro propriamente dito) é resolvido pelo transmissor inserindo um zero após cinco 1's consecutivos e pelo receptor retirando qualquer bit 0 após cinco 1's consecutivos.

O FCS é um campo de 16 bits, calculado pelo transmissor em função da sequência de bits (1's e 0's) do quadro a ser transmitido, entre, mas não incluído, o último bit do flag inicial e o primeiro bit do próprio FCS (que é transmitido do 16º bit para o 1º bit), excluindo obviamente os bits inseridos para transparência. Esta função é definida pela Recomendação X-25, e é semelhante a uma implementação típica de CRC. É definida como o complemento da soma (módulo 2) de dois restos de divisões de polinômios. A primeira divisão leva em consideração o número de bits do quadro e a segunda leva em consideração o conteúdo do quadro. O receptor analisa se o FCS recebido está correto, e em caso contrário descarta o quadro. Este é um primeiro controle de erro.

Normalmente o cálculo e a verificação do FCS é feito por hardware, o que é verdade em nossa implementação, pois estamos utilizando a interface 8273 da Intel, à qual tem hardware que realiza este cálculo e esta verificação.

OBS: Como não é necessário o auxílio do software, o cálculo é obviamente feito rapidamente.

Em sistemas que não possuem hardware para o cálculo do FCS, a utilização de tabelas ajuda na rapidez desse cálculo por software [24].

A recomendação X-25 define três tipos de quadros: informação, supervisão e não numerados. Os quadros de supervisão são três:

RR - Receive Ready (C,R)

RNR - Receive not Ready (C,R)

REJ - Reject (C,R)

Na implementação a ligação é balanceada, ou seja, qualquer uma das partes pode iniciar ou terminar a ligação, e os quadros chegam assincronamente. Os quadros não numerados necessários para estabelecer uma ligação balanceada assíncrona são:

SABM - Set asynchronous balanced Mode (C)

DM - Disconnected mode (R)

DISC - Disconnected (C)

UA - Unnumbered acknowledgement (R)

FRMR - frame reject (R).

Temos ainda o quadro de informação:

I - Information (C).

Os quadros podem ser comandos e/ou respostas, de acordo com a sua função. Na descrição anterior, entre parênteses, C indica que o quadro pode ser usado como comando e R indica que pode ser usado como resposta.

Os quadros I (informação) são numerados sequencialmente em

módulo 8, ou seja, recebem valores de 0 a 7 ciclicamente, sendo que o primeiro quadro recebe o valor 0.

A Recomendação X-25 define então variáveis de estado para controlarem esta sequencialização em cada uma das partes:

1) V(S) - contém o número de sequência do próximo quadro de informação a ser transmitido.

2) V(R) - contém o número de sequência do próximo quadro de informação a ser recebido.

Os formatos dos campos de controle dos quadros são mostrados na figura 1.4.

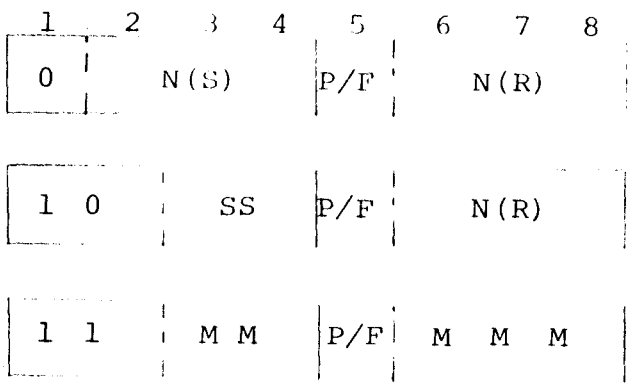


Figura 1.4

onde: N(S) - número de sequência do quadro I

N(R) - número de sequência do próximo quadro de informação esperado.

S,M - bits que distinguem os diferentes quadros.

P/F - bit poll-final.

As funções dos quadros implementados são:

I: - transmissão de informação.
- confirma a recepção dos quadros I até (N(R)-1) mod 8

RR: - indica que o transmissor deste quadro (DCE ou DTE) está pronto para receber quadros de informação (Ex: quando sai de uma condição de ocupado. - transmissão de RNR anteriormente).
- confirma a recepção dos quadros I até (N(R)-1) mod 8
- como comando e com p = 1, pergunta à outra parte (DCE ou DTE) sobre o seu estado.

RNR: - indica que o transmissor desta quadro está ocupado.
- confirma a recepção dos quadros I até (N(R)-1) mod 8 e informa que não aceitará mais quadros I.
- como comando e com p = 1, pergunta à outra parte (DCE ou DTE) sobre o seu estado.

REJ: - indica a recepção de quadros I fora de ordem, e portanto rejeitados (descartados).
- confirma a recepção dos quadros I até (N(R)-1) mod 8 e pede a retransmissão dos quadros I seguintes.
- como comando e com p = 1, pergunta à outra parte (DCE ou DTE) sobre o seu estado.

SABM: - pedido de conexão do DTE (ou DCE) ao DCE (ou DTE).

DISC: - pedido de desconexão do DTE (ou DCE) ao DCE (ou DTE).

UA: - resposta afirmativa aos comandos não numerados (SABM, DISC). Ex: aceitação de conexão, aceitação de desconexão.

DM: - indica a condição de desconectado da ligação.

FRMR: - indica uma condição de erro que não pode ser recuperável pela retransmissão de um quadro idêntico. Exemplos:

- Recepção de um quadro inválido ou não implementado.
- Recepção de um quadro I com campo de informação maior do que o comprimento permitido.
- Recepção de um N(R) inválido.
- Recepção de um quadro com campo de informação, o qual não é permitido, ou recepção de um quadro de supervisão ou não numerado com comprimento incorreto.

No campo de endereço (figura 1.3) são usados (A = 3) ou (B = 1) da seguinte maneira:

Quadros que são comandos transferidos do DCE para o DTE: A
Quadros que são respostas transferidos do DTE para o DCE: A
Quadros que são comandos transferidos do DTE para o DCE: B
Quadros que são respostas transferidos do DCE para o DTE: B.

Um comando transmitido pelo DTE (ou DCE) com bit p = 1 causa uma resposta do DCE (ou DTE) com bit F = 1.

A lista de parâmetros do sistema para o nível 2, segundo a Recomendação X-25 são:

- Timer: T_1
- Máximo número de transmissões: N2
- Máximo número de bits em um quadro I: N1
- janela: k ($k \leq 7$).

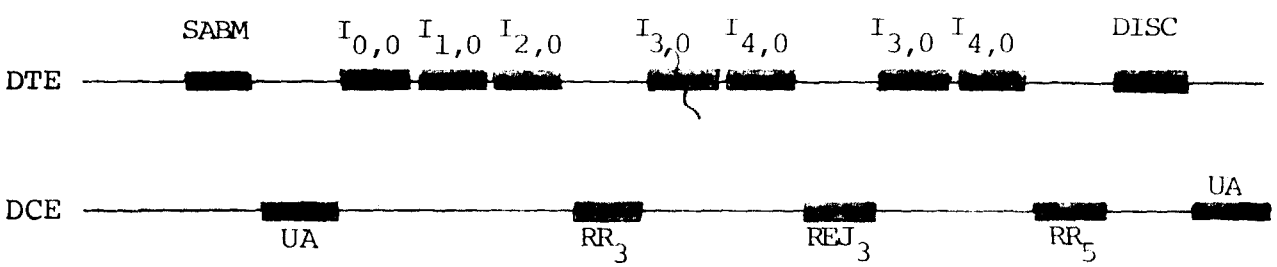
Define-se janela como sendo o número máximo de quadros de informação que se permite estar em trânsito antes de receber confirmação. Como desejamos receber os quadros I estritamente em ordem e a numeração é módulo n = 8: janela $k \leq n-1 = 8-1 = 7$

$k \leq 7$.

Na nossa implementação utilizamos k = 7.

A seguir damos dois exemplos de utilização dos quadros descritos anteriormente. As figuras representam a transmissão (e recepção) dos quadros no tempo.

EXEMPLO 1:

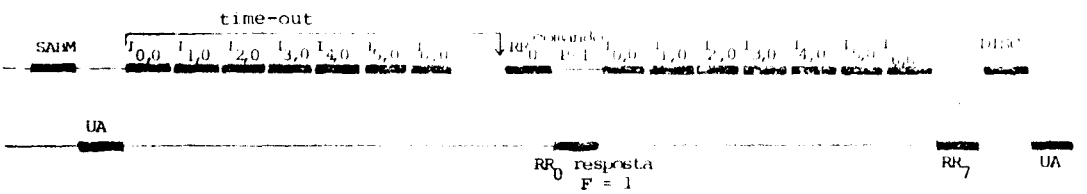


onde $I_{a,b}$ representa um quadro I com N(S) = a e N(R) = b.

S_a representa um quadro de supervisão com N(R) = a.

O DTE transmite inicialmente SABM. O DCE podendo conectar-se com este DTE envia UA como resposta. Depois de recebido este UA, o DTE pode enviar quadros de informação $I_{0,0}$, $I_{1,0}$, $I_{2,0}$. O DCE recebendo os quadros de informação 0, 1 e 2 pode ou confirmar cada um deles, ou confirmar todos através de um RR_3. O DTE envia os quadros $I_{3,0}$ que se perde e $I_{4,0}$. O DCE recebendo apenas o quadro $I_{4,0}$ e esperando o quadro $I_{3,0}$ entra numa condição de rejeição enviando o REJ_3. O DTE recebendo REJ_3 retransmite $I_{3,0}$ e $I_{4,0}$. Para encerrar a conexão o DTE enviou DISC, ao qual o DCE aceitando enviou UA.

EXEMPLO 2:



Depois de realizada a conexão, o DCE enviou os quadros $I_{0,0}$, $I_{1,0}$, $I_{2,0}$, $I_{3,0}$, $I_{4,0}$, $I_{5,0}$, $I_{6,0}$ os quais não foram confirmados pelo DTE. Este é um exemplo que mostra a necessidade de se ter um *timer*. Expirado o time-out, o DCE pergunta ao DTE sobre o seu estado enviando um comando RR_0 com p = 1, ao qual o DTE responde com um RR_0 com F = 1, o que significa que o DTE não recebeu os sete quadros de informação. Então, o DCE retransmite todos os quadros de informação novamente. Ao receber a confirmação em RR_7 o

DCE pede o encerramento d conexão, ao qual e atendido com o envio de um UA.

1.2.2. NÍVEL DE PACOTES (NÍVEL 3).

A comunicação neste nível é realizada através de pacotes e é implementado o serviço de circuitos (chamadas) virtuais.

A chamada virtual estabelece um caminho lógico provisório entre dois DTE's. Existe também o conceito de chamada virtual permanente que estabelece um caminho lógico permamente entre dois DTE's. Define-se canal lógico como sendo o ponto de acesso de um circuito virtual. A recomendação X-25 define quinze grupos de 255 canais lógicos.

Um DTE pode ter simultaneamente várias chamadas virtuais em progresso ou circuitos virtuais permanentes.

A figura 1.5 ilustra alguns destes conceitos.

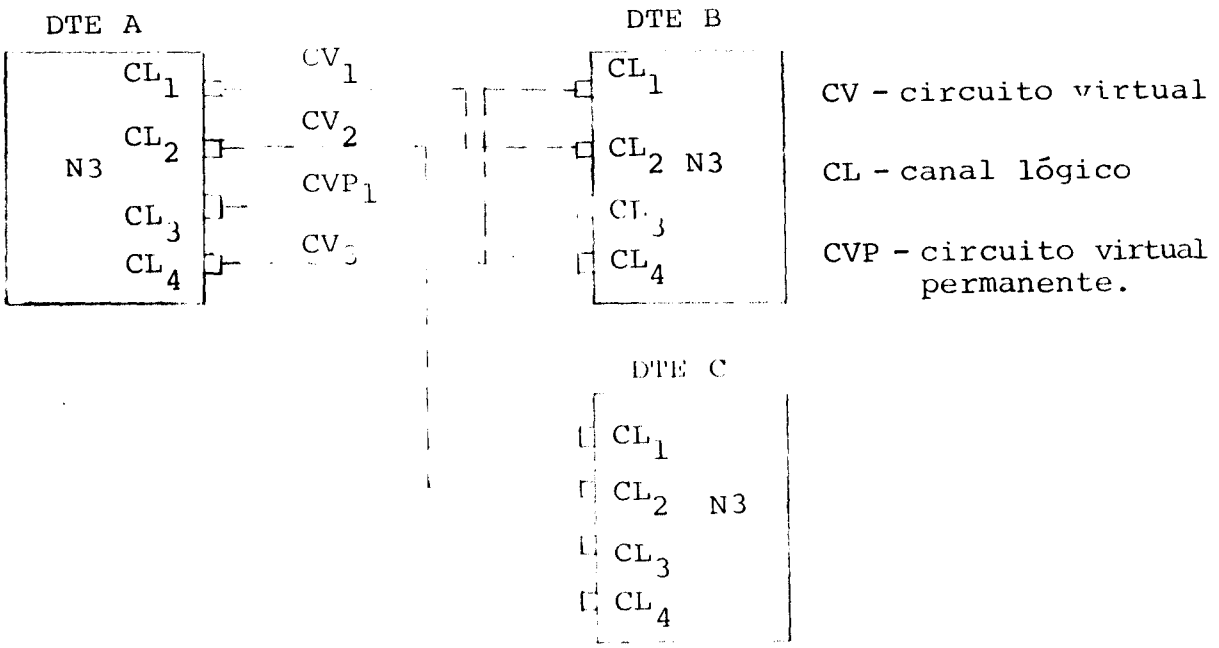


Figura 1.5

Uma chamada virtual possui três fases: estabelecimento da chamada (conexão), troca de dados e fechamento da chamada (desconexão). A chamada virtual permanente possui somente a segunda fase.

Os pacotes necessários para o serviço de circuitos virtuais são:

Do DCE para o DTE	Do DTE para o DCE
Conexão e desconexão	
Incoming call (ICC)	Call request (CAR)
Call connected (CAC)	Call accepted (CAA)
Clear Indication (CLI)	Clear request (CLR)
DCE clear confirmation (CCC)	DTE clear confirmation (CTC)
para transmissão de dado e "interrupt"	
DCE data (DATC)	DTE data (DATT)
DCE interrupt (CIN)	DTE interrupt (TIN)
DCE interrupt confirmation (CIC)	DTE interrupt confirmation (TIC)
para controle de fluxo e "Reset"	
DCE RR (RRC)	DTE RR (RRT)
DCE RNR (RNRC)	DTE RNR (RNRT)
Reset Indication (RSI)	Reset request (RSR)
DCE Reset confirmation (RSCC)	DTE Reset confirmation (RSCT)
para "Restart"	
Restart Indication (RAI)	Restart Request (RAR)
DCE Restart confirmation (RACC)	DTE Retart confirmation (RACT)

Esta divisão em dois grupos de pacotes existe apenas para indicar a origem do pacote (DTE ou DCE). O formato de dois pacotes

correspondentes em grupos diferentes é basicamente o mesmo. (diferem apenas em alguns campos que são opcionais).

A figura 1.6 mostra as três fases de uma chamada virtual, ilustrando a utilização de alguns pacotes

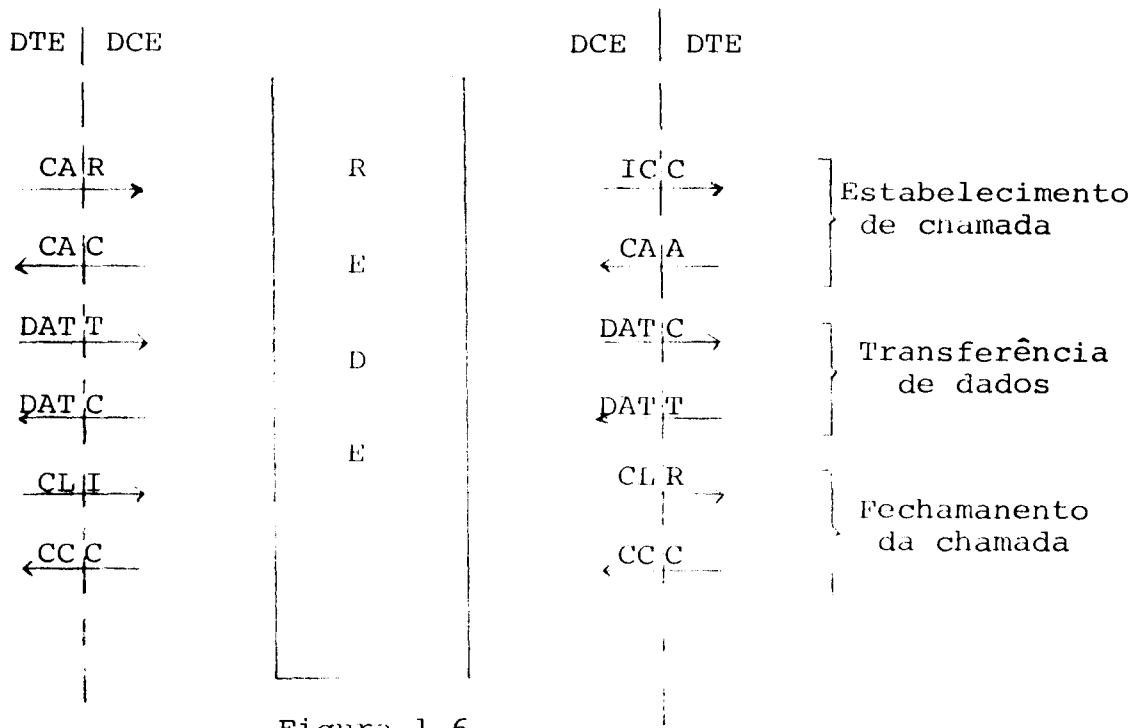


Figura 1.6

As funções dos pacotes são:

Incoming Call (Call Request) - pedido de estabelecimento de um circuito virtual por parte de um DCE (DTE).

Call connected (Call accepted) - aceitação pelo DCE (DTE) ao pedido de estabelecimento do circuito virtual feito pelo DTE (DCE).

Clear indication (Clear Request) - pedido de fechamento do circuito virtual por parte do DCE (DTE).

DCE Clear Confirmation (DTE Clear Confirmation) - aceitação pelo DCE (DTE) ao pedido de fechamento do circuito virtual feito pelo DTE (DCE).

DCE data (DTE data) - transferência de informação para o DTE (DCE).

DCE interrupt (DTE interrupt) - transferência de um byte de informação para o DTE (DCE), o qual não será submetido ao controle de fluxo pela rede. Utilizado, por exemplo, em caso de congestionamento da rede.

DCE interrupt confirmation (DTE interrupt confirmation) - confirmação de um pacote DCE interrupt (DTE interrupt).

DCE RR (DTE RR) - indica que está pronto para receber pacotes de informação do DTE (DCE) em um circuito virtual especificado

- confirma a recepção dos pacotes de informação até (P(R)-1) mod 8 em um circuito virtual especificado.

DCE RNR (DTE RNR) - indica uma condição de ocupado em um circuito virtual especificado.

- confirma a recepção dos pacotes de informação até (P(R)-1) mod 8 em um circuito virtual especificado.

Reset indication (Reset Request) - pedido de RESET pelo DCE (DTE) de um circuito virtual especificado.

DCE reset confirmation (DTE reset confirmation) - confirmação pelo DCE (DTE) do pedido de RESET feito pelo DTE (DCE).

Restart Indication (Restart Request) - pedido de RESTART pelo DCE (DTE). O Restart equivale a um clear em todos os circuitos virtuais e um Reset em todos os circuitos virtuais permanentes.

DCE restart confirmation (DTE restart confirmation) - confirmação pelo DCE (DTE) do pedido de restart feito pelo DTE (DCE).

OBSERVAÇÃO: O envio de um RNR não implica em se poder descartar pacotes de dados que cheguem posteriormente, pois neste nível não temos retransmissão de pacotes de dados. Não existe time-out para pacotes de dados. Este fato implica também na existência de um nível superior (mais alto) para retransmitir dados em caso de RESET ou RESTART do nível 3.

O tamanho mínimo dos pacotes transferidos através da interface é de três octetos (8 bits), como ilustra a figura 1-7.

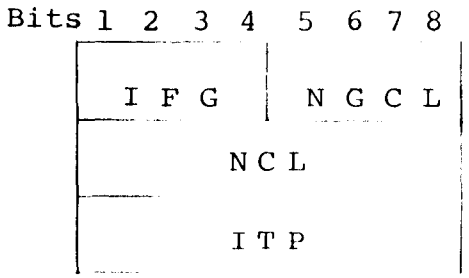


Figura 1.7

1º octeto: 4 bits mais significativos: Identificador de formato geral.

: 4 bits menos significativos: número de grupo de canal lógico.

2º octeto: número do canal lógico.

3º octeto: Identificador de tipo de pacote.

Segundo a recomendação X-25,

1º) A numeração dos pacotes de dados dentro de um circuito virtual pode ser módulo 8 ou módulo 128. Em nossa implementação utilizamos módulo 8.

2º) A confirmação dos pacotes de dados pode ser local à interface ou fim-a-fim. No segundo caso, a confirmação de um pacote significa o seu recebimento pelo ETD remoto. Na nossa implementação utilizamos a primeira opção.

O Identificador de formato geral contém estas duas informações.

Além desses três octetos, alguns pacotes possuem outros campos. Por exemplo, em um call request existem os campos: comprimento

do endereço do DTE local, comprimento do endereço do DTE remoto, DTE local, DTE remoto, comprimento do campo de facilidades, facilidades, dados do usuário. Não descrevemos aqui os formatos dos pacotes, pois a finalidade desta seção é apenas dar uma visão geral do protocolo, em especial aquilo que é importante para nossa implementação. Por este motivo, por exemplo, não descrevemos aqui o serviço de datagrama que a recomendação X-25 define e que nós não implementamos. A própria Rede Pública da Embratel a ser lançada brevemente não dispõe deste serviço.

O mesmo mecanismo de janela descrito para o nível 2 aplica-se ao nível 3. Como a sequencialização utilizada é módulo 8 e como queremos receber os pacotes em ordem, a janela deve ser menor ou igual a sete, como visto anteriormente. Devido ao fato de não haver retransmissão de pacotes de dados, podemos inclusive usar a janela para controle de congestão, ou seja, quando uma das partes está ocupada, pode não confirmar imediatamente os pacotes recebidos, causando um atraso nos pacotes enviados pela outra parte, aliviando este primeiro, sem causar nenhum "overhead" de retransmissão.

Adotamos janela $w = 2$ em nossa implementação, devido ao fato de termos confirmação local e de termos possibilidade de aproveitá-la como controle de congestão. É um valor usual.

Para controlar esta sequencialização cada uma das partes deve ter duas variáveis de estado para cada circuito virtual:

1) P(S) - contém o número de sequência do próximo pacote de dado a ser transmitido no circuito virtual correspondente.

2) P(R) - contém o número de sequência do próximo pacote de dado a ser recebido no circuito virtual correspondente.

Uma observação importante é que a Recomendação X-25 define para este nível diagramas de estado. Para o nível 2 a recomendação não define diagramas de estado, embora várias vezes comente estados que devem existir. Desta maneira, para o nível 3, não é difícil dinamicamente descobrirmos o estado da interface DTE / DCE para um circuito virtual, e também não é difícil sabermos a reação de uma das partes quando ocorre um evento, uma recepção ou uma transmissão.

1.2.3. ESPECIFICAÇÃO DO PROTOCOLO X-25.

1.2.3.1. NÍVEL 2.

A recomendação X-25 não apresenta diagramas de estados para definir o nível 2, embora cite vários estados que devem existir:

a) Fase de trarsfe ncia d informação;

b) Fase desconectado;

c) Condição de ocupado: DTE ou DCE não pode temporariamente receber quadros de informação;

d) Erro na sequência N(S): N(S) do quadro I recebido é diferente da variável de estado V(R);

e) Condição de rejeição: quando ocorre um erro, que não pode ser consertado retransmitindo quadros.

Bochmann e Joachim em [3] apresentam um diagrama de estados para uma ligação no modo de *resposta* assíncrona baseado nos diagramas do HDLC (High-level data link control) [36]. O HDLC é um protocolo padronizado pela ISO. O CCITT fez o protocolo X-25 baseando-se no HDLC. A nossa implementação é de uma ligação *balanceada* assíncrona como já citamos. Porém, fazendo uma analogia entre esses dois modos, podemos utilizar os estados propostos por Bochmann e Joachim: (Fig. 1.8)

- Desconectado
- Esperando conexão
- Conectado
- Esperando desconexão
- Condição de exceção.

Estes também são os estados propostos pelo HDLC e estados semelhantes são propostos pela Telebrás em [21].

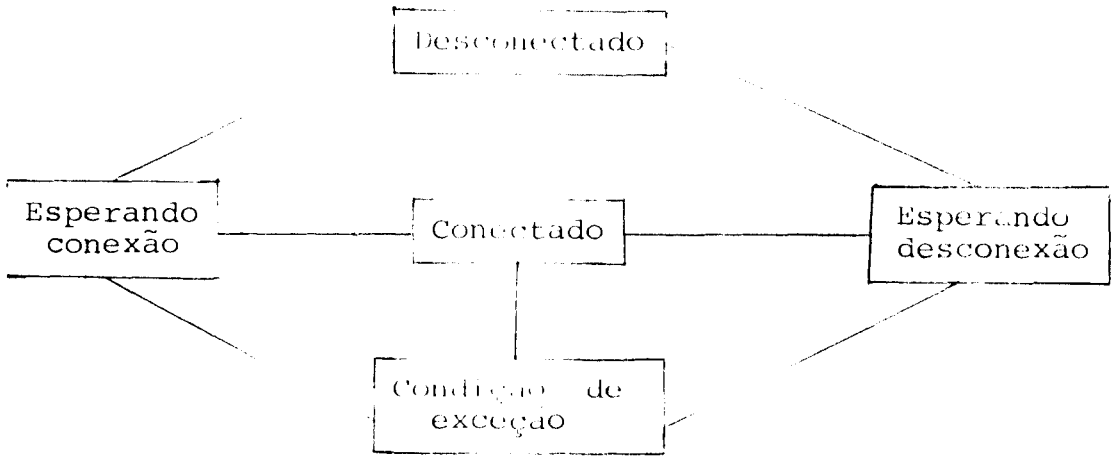


Figura 1.8

Alguns subestados do estado conectado são descritos pelo HDLC. Bochmann e Joachim simplificaram estes subestados (fig.1.9).

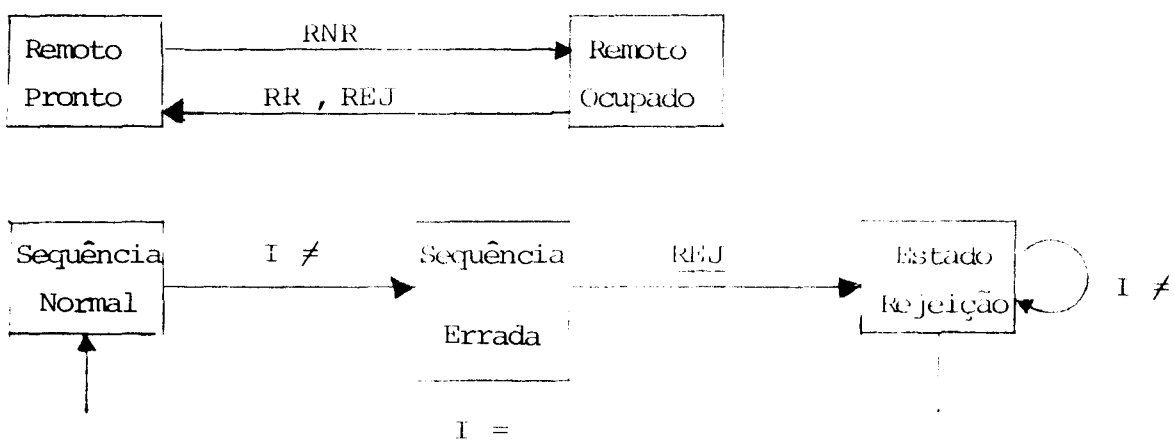


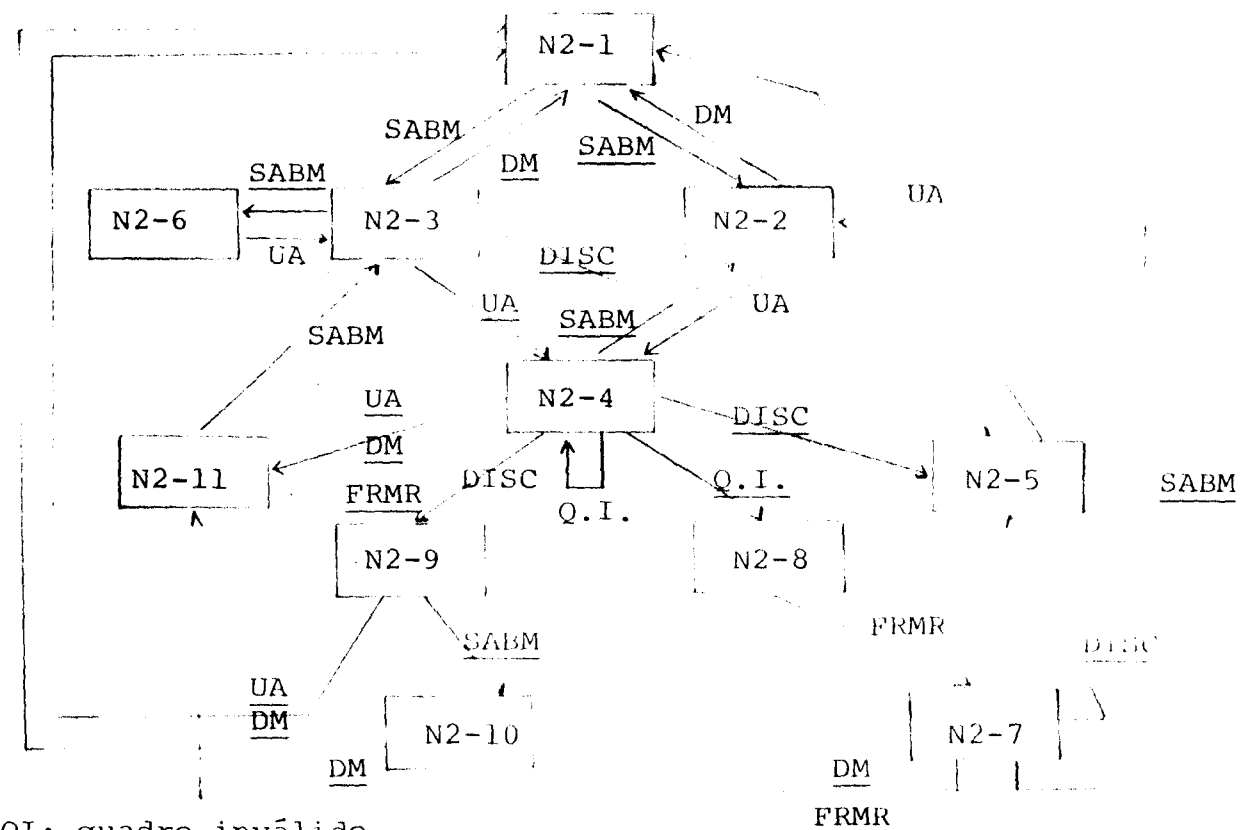
Figura 1.9

MEN : Envio de MEN

ME↓ : Recepção de MEN

Na implementação de [21], são utilizados seis subestados: pronto, sequência errada, DTE ocupado, DCE ocupado, DTE e DCE ocupados, sequência errada e DCE ocupado.

O diagrama que sugerimos resulta da união dessas idéias e da manutenção da simplicidade das mesmas, porém analisando as diversas possibilidades de recepção e transmissão de quadros para os diferentes estados. O diagrama possui os estados descritos anteriormente definidos pelo HDLC: desconectado (N2-1), DTE esperando por conexão (N2-3), conectado (N2-4), esperando desconexão (N2-9), condição de exceção (N2-7) e pode ser visto na fig. 1.10.



QI: quadro inválido

Figura 1.10

O diagrama obedece a Recomendação X-25, e os casos de colisão de SABM's ou DISC's podem ser verificados facilmente. Note também que o envio de um FRMR causa o pedido de uma nova conexão.

Os subestados para o estado conectado na implementação são três: pronto, condição de rejeição e DTE ocupado. A figura 1.11 mostra estes subestados.

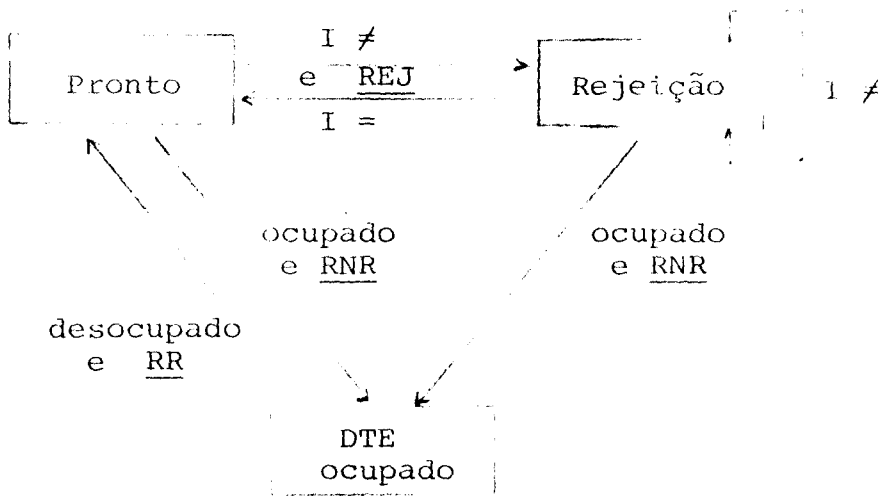


Figura 1.11

O estado de condição de ocupado do DCE é guardado e verificado nos procedimentos de recepção e transmissão do nível 2 respectivamente.

Estes dois diagramas cobrem todos os estados comentados pelo X-25 e descritos no início dessa seção.

1.2.3.2. NÍVEL 3.

A Recomendação X-25 define diagramas de estados para os circuitos virtuais, os quais podem ser vistos nas figuras 1.12a, 1.12b e 1.12c.

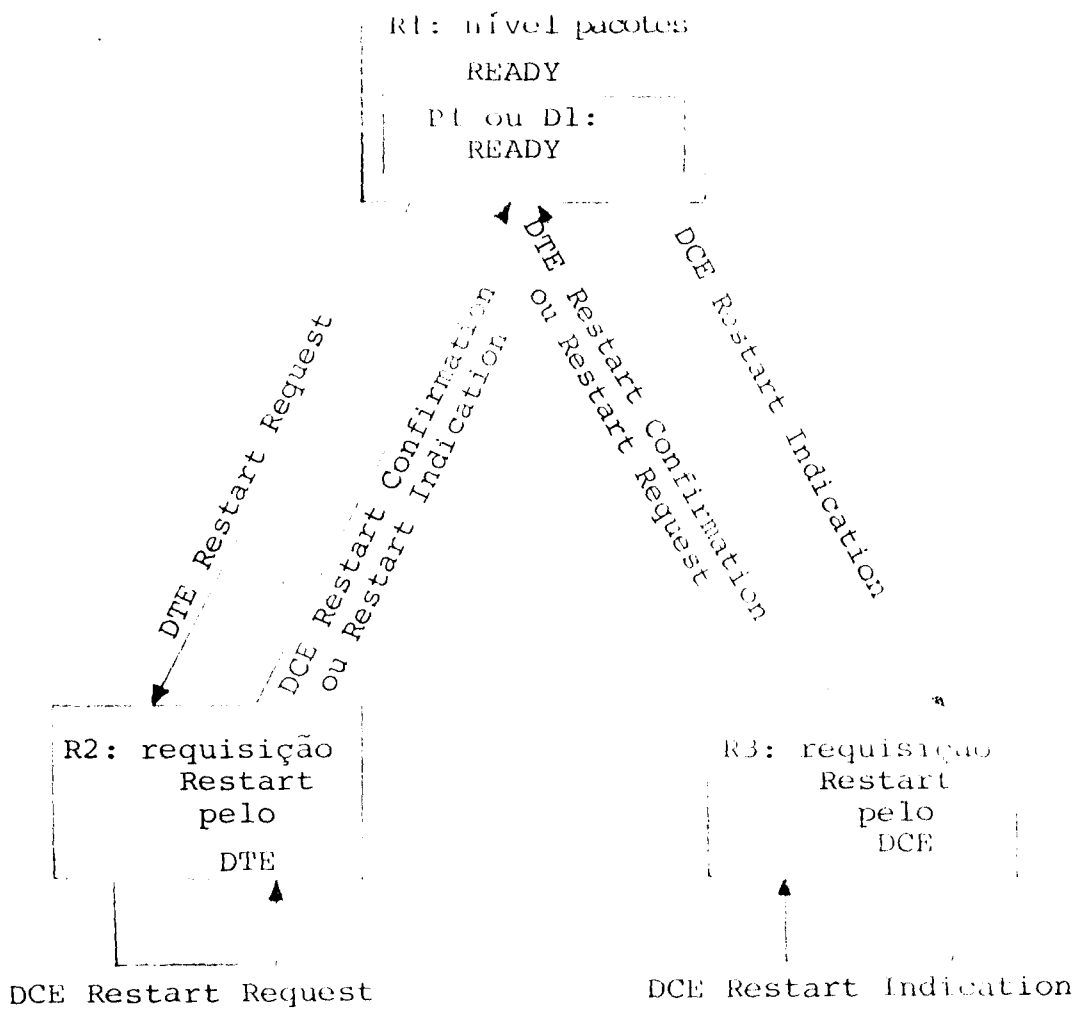


Figura 1.12a

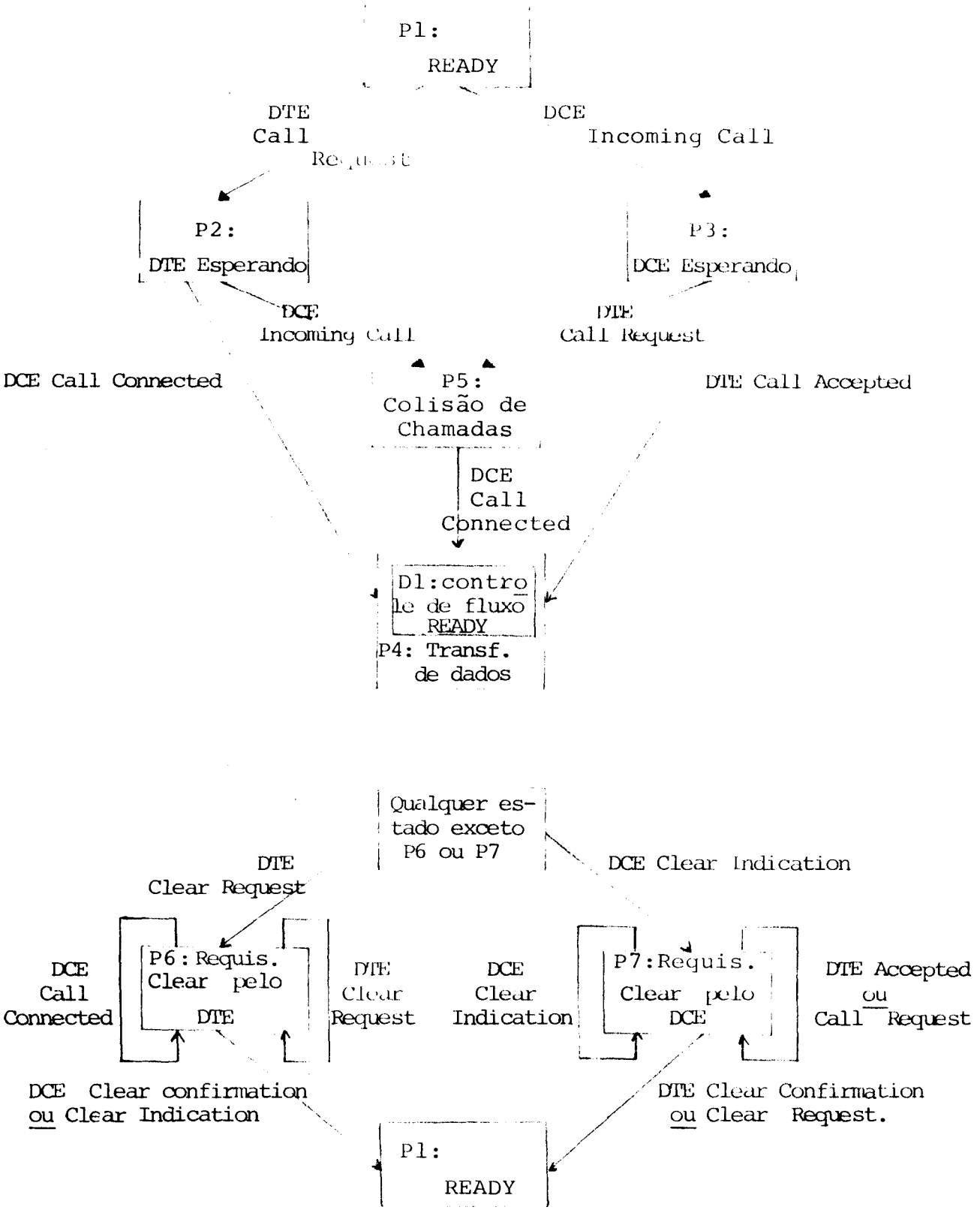

P1:
READY
DTE
Call
Request
DCE
Incoming Call
P2:
DTE Esperando
P3:
DCE Esperando
DCE
Incoming Call
DTE
Call Request
DCE Call Connected
P5:
Colisão de
Chamadas
DTE Call Accepted
DCE
Call
Connected
D1:contro
le de fluxo
READY
P4: Transf.
de dados
Qualquer es-
tado exceto
P6 ou P7
DTE
Clear Request
DCE Clear Indication
DCE
Call
Connected
P6: Requis.
Clear pelo
DTE
DTE
Clear
Request
DCE
Clear
Indication
P7:Requis.
Clear pelo
DCE
DTE Accepted
ou
Call Request
DCE Clear confirmation
ou Clear Indication
DTE Clear Confirmation
ou Clear Request.
P1:
READY

Figura 1.12b

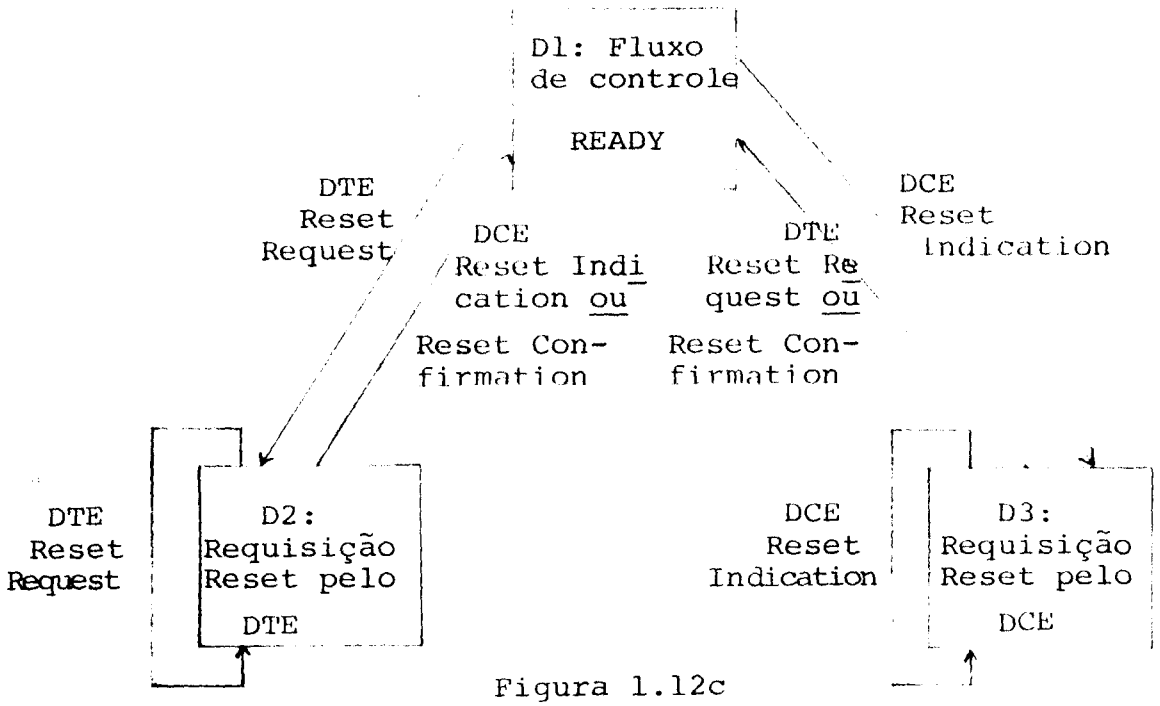


Figura 1.12c

O primeiro nível (mais alto) destes diagramas refere-se às situações de RESTART, pois, como já citamos, um RESTART significa um RESET nos circuitos virtuais permanentes e um CLEAR nas chamadas virtuais. Dentro do estado READY do nível de transferência de pacotes RESTART temos o segundo nível destes diagramas, que se refere às situações de CALL-SET-UP e CLEAR (apenas existentes para as chamadas virtuais). Dentro do estado READY deste nível de transferência de pacotes de CALL-SET-UP e CLEAR temos o último nível destes diagramas, que se refere às situações de RESET definindo as trocas de pacotes deste tipo. Este terceiro nível apresenta um estado no qual se realiza a transferência de pacotes de informação.

A Recomendação X-25 também define a ação que o DCE deve

tomar ao receber pacotes em qualquer um dos estados da interface DTE/DCE definidos pelos diagramas, em uma tabela anexa. Embora não defina as ações para o DTE, por analogia podemos facilmente descobrí-las. Em nosso sistema essas ações foram verificadas e implementadas.

1.2.3.3. NÍVEL 4.

Este nível, na implementação, é o único de controle fim-a--fim. Portanto, este nível realiza a função do nível de transporte no que concerne por exemplo à sequencialização das mensagens, abertura e fechamento de conexões à nível de DTE, etc, mas também realiza a função de níveis superiores, como por exemplo, o fornecimento do endereço do usuário remoto: DTE remoto e processo remoto (podendo ser o número do terminal conectado ao DTE remoto).

As mensagens utilizadas e suas respectivas funções são:

CONEXÃO (CON) - pedido de conexão à um DTE remoto;

CONEXÃO EFETUADA (CEF) - confirmação do pedido de conexão;

CONEXÃO REJEITADA (CRJ) - rejeição do pedido de conexão;

DADO (DAD) - transferência de informação;

- confirma a recepção de mensagens;

CONFIRMAÇÃO DADO (CONF) - confirma a recepção de mensagens;

DESCONEXÃO (DESC) - desconexão no sentido do DTE que enviou a mensagem para o outro DTE. No outro sentido, a transferência de dados continua até se enviar

um DESC neste sentido.

Para diferenciar as diversas mensagens utilizamos um byte do campo de informação dos pacotes. A mensagem CON necessita de mais um byte para informar o número do processo (ou terminal) a quem é pedido a conexão. Portanto, os pacotes de informação só poderão ter no máximo 127 bytes de dados propriamente dito pois um byte será utilizado para avisar que esta mensagem é de dado.

Os formatos das mensagens estão ilustrados na figura 1.13.

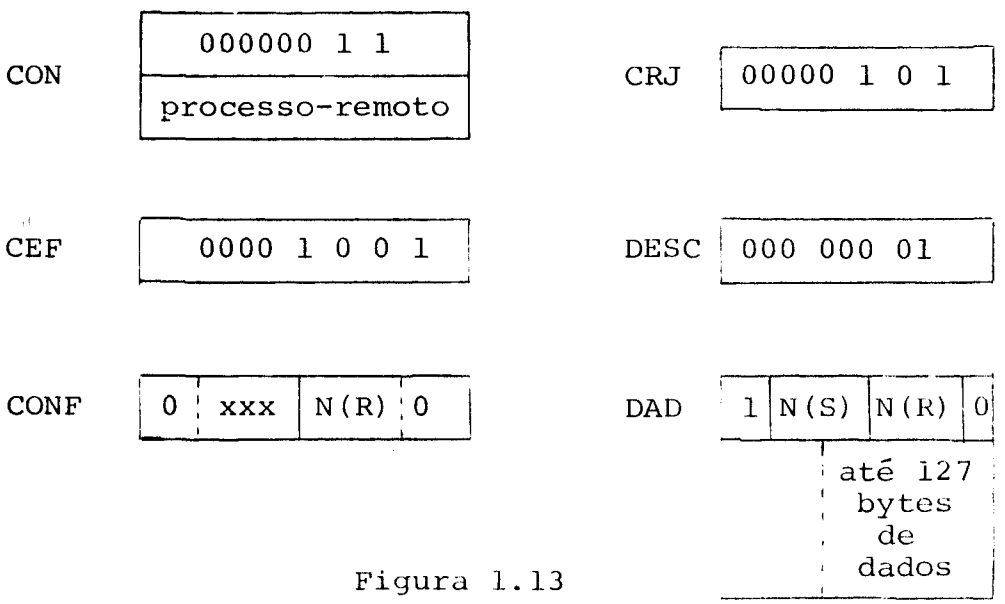


Figura 1.13

Estamos supondo na implementação que o nível 4 não possui time-out e portanto não possui a capacidade de retransmitir mensagens.

Esta hipótese é viável pois em condições de conexão do nível 2 e RESET e RESTART do nível 3 estamos retransmitindo os

quadros e pacotes não confirmados (programação defensiva). Este caso não pode ser evitado, pois, por exemplo, se uma das partes pede desconexão do nível 2, ela não sabe se os quadros transmitidos foram recebidos ou não, pois a confirmação pode vir depois do DESC (a qual será ignorada) ou pode não vir se os quadros forem perdidos.

Esta política defensiva nos leva a ter em alguns casos duplicatas de mensagens, porém nunca a perda de mensagens. Desta maneira, se o nível 4 recebeu uma mensagem fora da sequência esta pode ser jogada fora, porque necessariamente é uma duplicata, já que as mensagens seguem um mesmo circuito virtual e portanto uma mensagem transmitida posteriormente a uma primeira não pode "passar na frente" desta primeira.

A hipótese falha se qualquer um dos nós ou computadores hospedeiros cair, pois neste caso há perda de mensagens. Porém, não quisemos repetir todo o controle de time-out e retransmissão de mensagens neste nível, já que é idêntico aos dos níveis 2 e 3 e que o objetivo ao introduzir o nível 4 é dar um controle primitivo fim-a-fim e servir como interface entre os outros níveis e os terminais do concentrador.

O diagrama proposto do DTE local para este nível está ilustrado na figura 1.14.

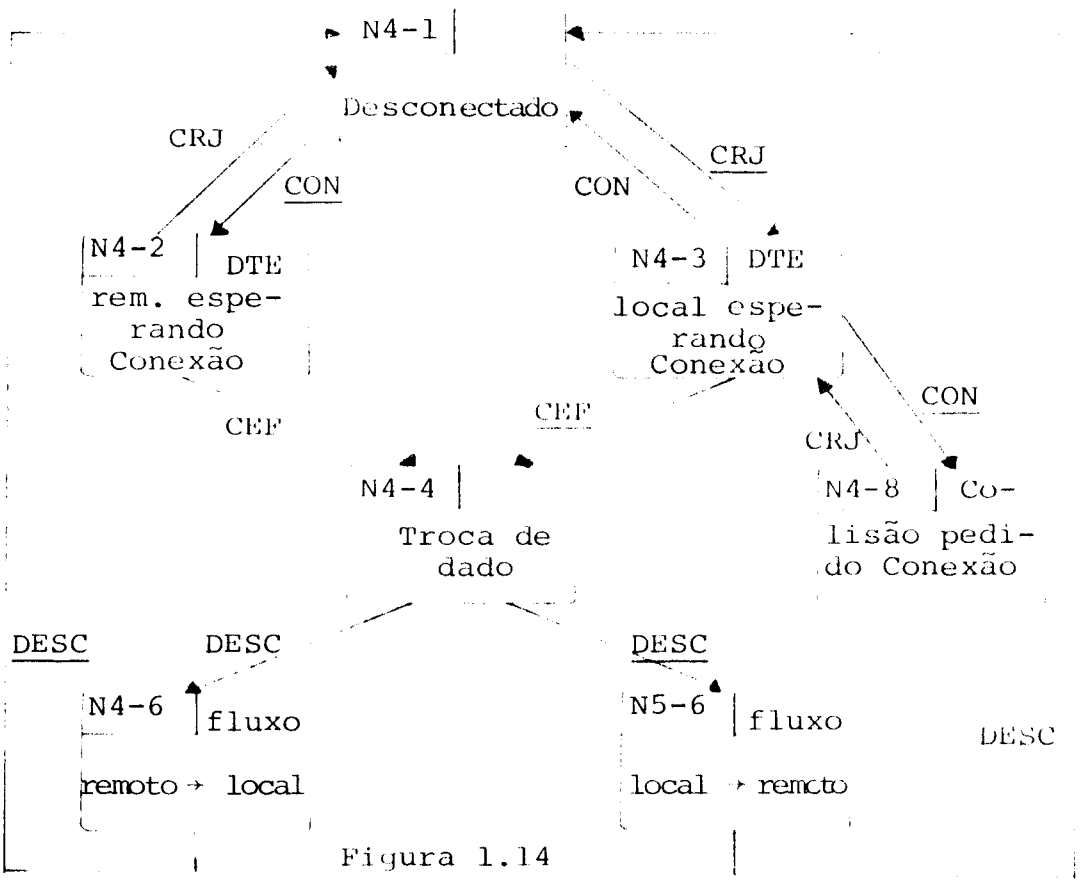


Figura 1.14

onde: MEN: transmissão de MEN
MEN: recepção de MEN.

Neste diagrama, o estado N4-1 representa o estado desconectado de uma dada porta lógica. Porta lógica é o ponto de acesso de um processo no nível 4. Cada terminal do concentrador está associado *sempre* a uma porta lógica. A cada porta lógica conectada do nível 4 existe associado um canal lógico do nível 3. (O número da porta lógica não é necessariamente igual ao do canal lógico correspondente).

No estado N4-1, ao receber um CON do DTE remoto, a conexão

associada a esta porta lógica passa para o estado N4-2. Se o terminal ao qual o DTE remoto quer se conectar já estiver conectado através de uma outra porta lógica, com possivelmente outro DTE, o DTE local envia CRJ e a porta lógica passa para o estado N4-1. Isto é feito, porque na implementação um terminal pode ter em andamento apenas uma conexão (com um terminal remoto ou com um processo remoto). Se o terminal ao qual o DTE remoto quer se conectar não estiver conectado, o DTE local envia CEF e a conexão associada a esta porta lógica passa para e estado N4-4, que é o estado de troca de dados pelas duas partes. Neste estado os dois DTE's enviam DAD (mensagem que contém informação) e CONF (mensagem que confirma mensagens de dados).

Se um terminal desconectado quer se conectar com outro terminal, ele se associa a uma porta lógica (e deve haver pelo menos uma porta lógica desconectada, pois o número de portas lógicas é igual ao número de terminais) e o DTE local envia um CON, e a ligação passa para o estado N4-3. Neste estado três recepções são possíveis:

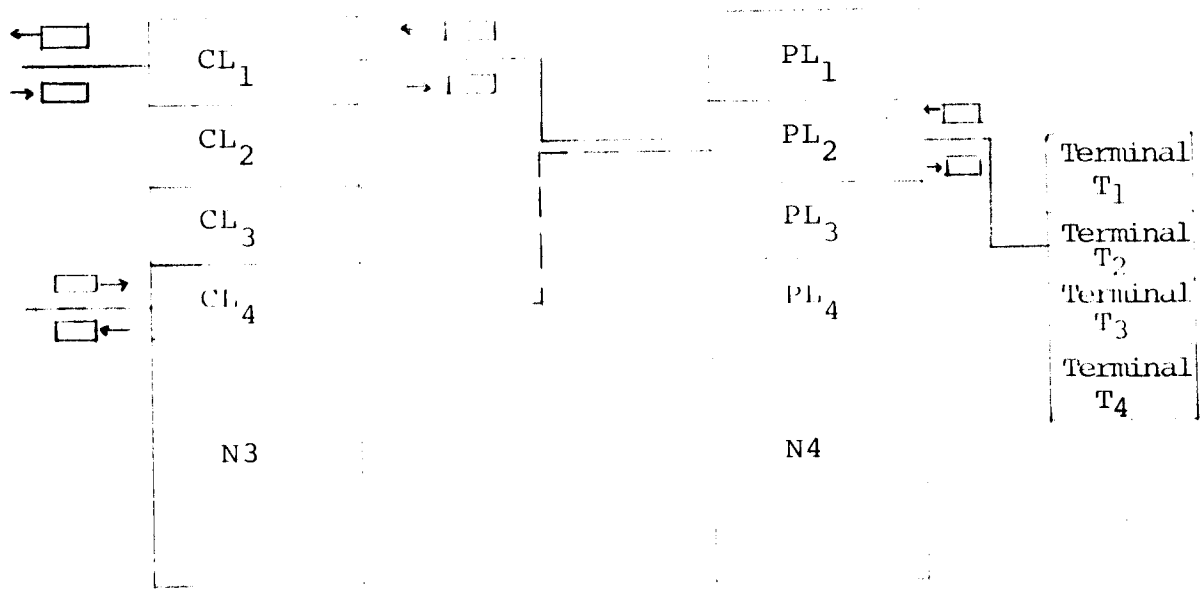
1) CEF: Esta recepção significa que o terminal remoto estava desconectado e portanto o DTE remoto completou a conexão. A ligação passa para o estado N4-4.

2) CRJ : Esta recepção significa que o terminal remoto estava conectado, possivelmente com um terceiro DTE. O DTE remoto não completou a ligação. A ligação volta para o estado N4-1.

OBSERVAÇÃO. O tempo entre recebermos a rejeição de uma conexão e o momento de tentarmos novamente a conexão deve ser aleatório, para que se os pedidos de conexão conflitantes forem dos mesmos dois terminais envolvidos, não tenhamos um impasse perpétuo dos dois DTE's sempre rejeitarem a conexão, pois pediram uma conexão anteriormente.

3) CON : Esta recepção significa que enquanto esta porta lógica foi usada por um terminal local para pedir uma conexão, possivelmente um terceiro DTE pediu também uma conexão para este terminal. A ligação passa para o estado N4-8. Este pedido de conexão é então rejeitado enviando-se um CRJ e a ligação volta para o estado N4-3.

OBSERVAÇÃO: A mensagem CRJ é enviada pelo circuito virtual do nível 3 pelo qual chegou o CON , que é um circuito virtual diferente do que foi enviado o CON inicial. Vide figura 1.15.



Comunicação em andamento: T_2 através de PL_2 e CL_1

Ao chegar um pedido de conexão para T_2 através de CL_4 : é enviado um CRJ através de CL_4

Figura 1.15

Se no estado N4-4 recebermos um DESC, a ligação passa para o estado N4-5 e significa que o DTE remoto não tem mais dados para transmitir, porém a ligação não foi encerrada pois o DTE local ainda pode enviar DADO's e recebeu CONF's. Isto encerra a comunicação de informação em apenas um sentido, do DTE remoto para o DTE local. O DTE local quando não tiver mais dado para transmitir, envia um DESC, a ligação passa para o estado N4-1 e a comunicação de informação está encerrada nos dois sentidos.

Se o DTE local para uma determinada ligação está no estado N4-4 e não tem mais informação para transmitir, envia um DESC. A ligação passa para o estado N4-6, significando que o DTE remoto ainda pode enviar mensagens do tipo DADO e que o DTE local ainda pode enviar CONF. Isto encerra a comunicação de informação em apenas um sentido, do DTE local para o DTE remoto. Em N4-6, a recepção de um DESC encerra a comunicação de informação no outro sentido. A ligação volta para o estado N4-1.

CAPÍTULO II

ANÁLISE DO PROTOCOLO

2.1. ÁRVORE DE PERTURBAÇÃO

Nessa seção analisamos o protocolo utilizado, previnindo - o contra deadlocks, recepções não especificadas, etc.

Alguns trabalhos [39, 34, 30] comentam que os protocolos existentes (CCITT X-21 e X-25, SNA da IBM) são corretos à nível funcional e de semântica, cumprindo objetivamente os propósitos aos quais foram criados. Porém, estes protocolos não tem previsto todas as possibilidades sintáticas possíveis, podendo ser incompletos ou logicamente inconsistentes.

Um exemplo deste fato é a colisão de um pacote DCE _ CLEAR _ INDICATION proveniente da rede com o pacote DTE_CALL_RESQUEST proveniente de um DTE no nível 3 do protocolo X-25. De acordo com a especificação, esta colisão seria identificada pela rede como um "local procedure error", apesar dessa colisão ser permitida pela especificação. Portanto, "local procedure error" torna-se ambíguo, pois está sendo utilizado para identificar colisões naturais e também para identificar violações reais do protocolo. Este exemplo é analisado também em [25] por P. Zafiropulo.

Isto nos leva a especificar e analisar formalmente o protocolo. Para especificar o protocolo usamos os diagramas de estado vistos no capítulo anterior e criamos um conjunto de regras que definem esses mesmos diagramas. Para analisar o protocolo construímos uma "árvore de perturbação". A árvore de perturbação simula a execução do protocolo, analisando todas as transições possíveis. Essa árvore é sugerida por P. Zafiropulo em [25] e detecta situações de erros no protocolo.

Para comunicação entre duas máquinas ditas computador 1 e computador 2, os nós da árvore possuem o seguinte formato:

EST1	CAN1
CAN2	EST2

onde: EST1: é o estado do computador 1.

EST2: é o estado do computador 2.

CAN1: é o canal que contém a sequência de mensagens enviadas pelo computador 1 e ainda não recebidas pelo computador 2.

CAN2: é o canal que contém a sequência de mensagens enviadas pelo computador 2 e ainda não recebidas pelo computador 1.

A árvore de perturbação é contruída da seguinte maneira.

O nó inicial (ou estado inicial do sistema) possui os dois canais vazios e os estados iniciais de cada computador.

Dado um nó, analisamos as próximas transições possíveis, ou seja, as transmissões a partir dos estados das duas partes e as recepções a partir do conteúdo dos dois canais. Quantas transições forem possíveis, quantos filhos este nó terá. Para cada filho, os estados e os canais serão obtidos em função do pai e da transição.

Se a transição é de recepção, a mensagem é retirada do canal e o estado do receptor é alterado (em função do diagrama de estados que define o protocolo).

Se a transição é de transmissão, a nova mensagem é inserida no canal correspondente (em função de qual computador enviou a mensagem) e o estado do transmissor também é alterado (de acordo com o diagrama de estados).

A árvore de perturbação vai sendo construída até se obter em suas folhas estados do sistema já analisados.

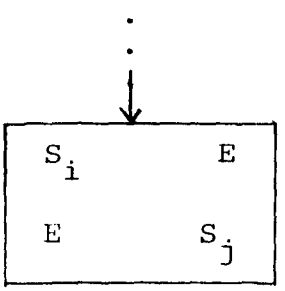
Construímos a árvore de perturbação para os níveis 2 e 4 da nossa implementação. Não construímos esta árvore para o nível 3, pois este possui diagramas de estado definidos pelo próprio X-25.

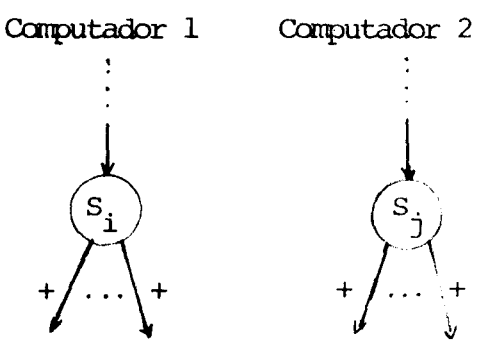
Esta técnica para a análise do protocolo pode ser aplicada para outras áreas de processos cooperantes. O argumento utilizado para demonstrar este fato é de Merlin em [37]: "Dado um sistema de processos cooperantes tal que a cooperação é feita através de troca de mensagens. Um protocolo é um conjunto de regras que governa esta mudança".

A partir da construção dessa árvore de perturbação podemos detectar quatro erros em potencial de projeto: deadlock, recepção não especificada, interações não executáveis e ambiguidade de estados.

Deadlock ocorre quando os dois computadores permanecem indefinidamente no mesmo estado em virtude de não terem o que computar. Na árvore de perturbação isto pode ser identificado pela existência de um nó que possui ambos os canais vazios (nenhuma transição de recepção a realizar) e nenhuma transição de transmissão possível a realizar.

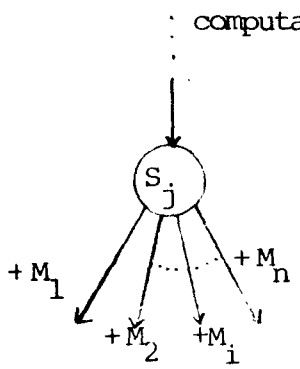
EX.: O computador 1 em S_i e o computador 2 em S_j não podem enviar mensagens, ou seja no diagrama de estados:

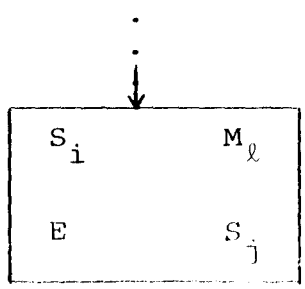




Recepção não especificada ocorre quando um dos computadores recebe uma mensagem M, num determinado estado S_i, e que no diagrama de blocos não possui transição capaz de absorver esta mensagem M. Na árvore de perturbação isto pode ser identificado pela existência de um nó que não possua transições (eventos) para absorver mensagens que estavam em um dos dois canais.

EX.: no diagrama de estados:





onde $\forall i$, $i \neq e$

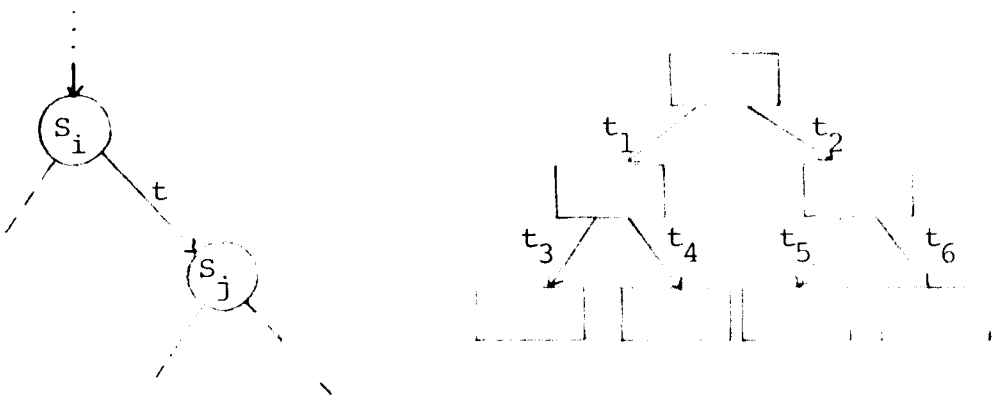
Interação não executável ocorre quando um evento de transmissão ou recepção nunca acontece em função de como foi projetado o protocolo (diagrama de estados). É análogo ao código não atingível em programas.

Este fato não é um erro, se o projetista colocou este evento apenas para ser executado em condições anormais (por exemplo,para

se recuperar de uma condição de erro).

Identificamos uma interação não executável, pela não existência de uma transição (transmissão ou recepção) do diagrama de estados na árvore de perturbação. A árvore de perturbação "cobre" todas as possibilidades de eventos).

EX.: No diagrama de estados:

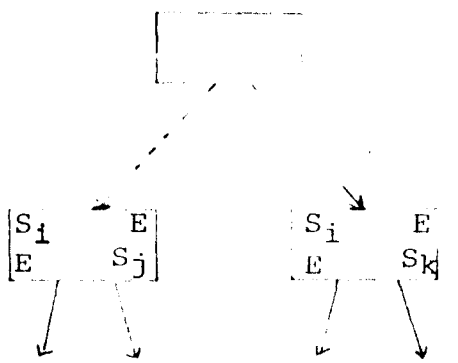


onde $t \neq t_i$ $\quad i = 1,\ldots,5$

Ambiguidade de estados ocorre quando um estado de um computador coexiste em dois ou mais estados do outro computador. Isto não significa uma situação de erro, mas devemos analisá-la com cuidado (erro em potencial). A semântica do protocolo é que nos vai informar se a existência dessa ambiguidade de estados é válida como um recurso a mais no protocolo ou não é válida por se tratar de um erro. Na árvore de perturbação essa ambiguidade é identificada pela existência de dois ou mais nós do sistema que possuam os dois canais vazios e um estado comum na diagonal.

4. Ambiguidade de estados:

EX.:



onde $j \neq k$

Construímos a seguir a árvore de perturbação para o protocolo do bit alternado como um exemplo. (figura 2.2).

O diagrama de estados deste protocolo á apresentado na figura 2.1, onde - significa transmissão, + significa recepção, D é uma mensagem de dados (D_o c/bit zero e D_1 com bit um) e A é uma mensagem de confirmação (A_o da mensagem de bit zero e A_1 da mensagem de bit 1).

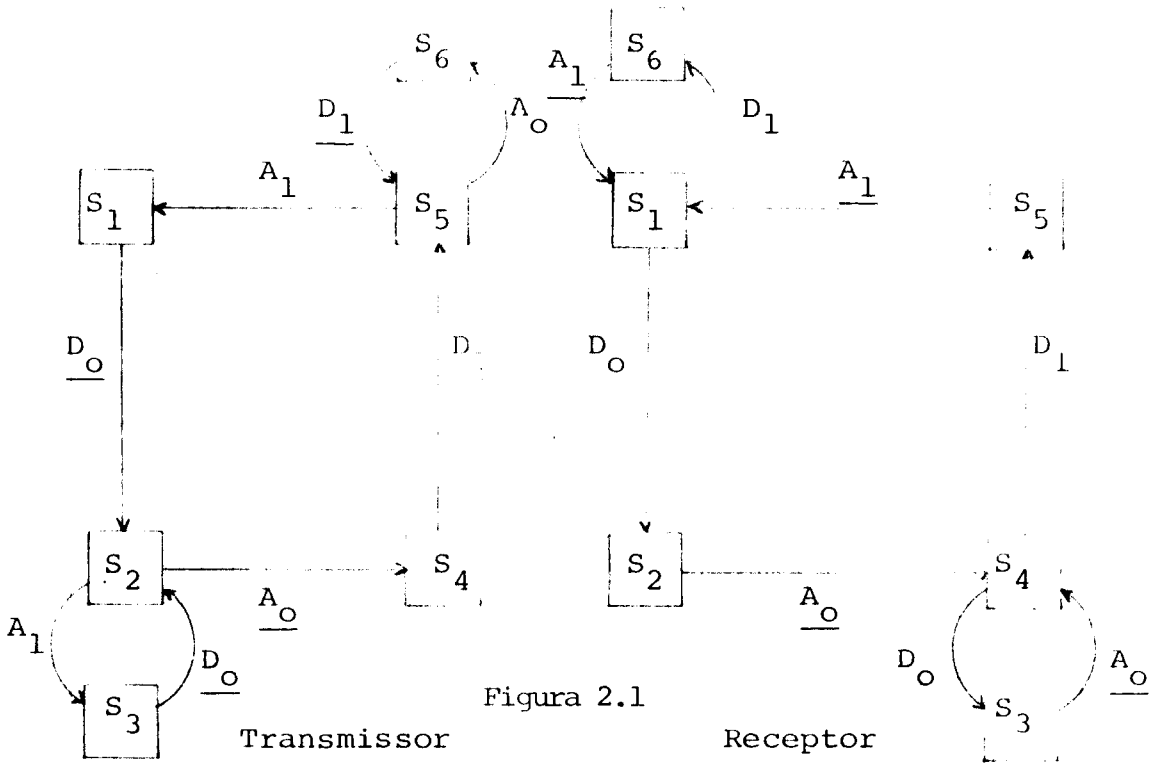


Figura 2.1

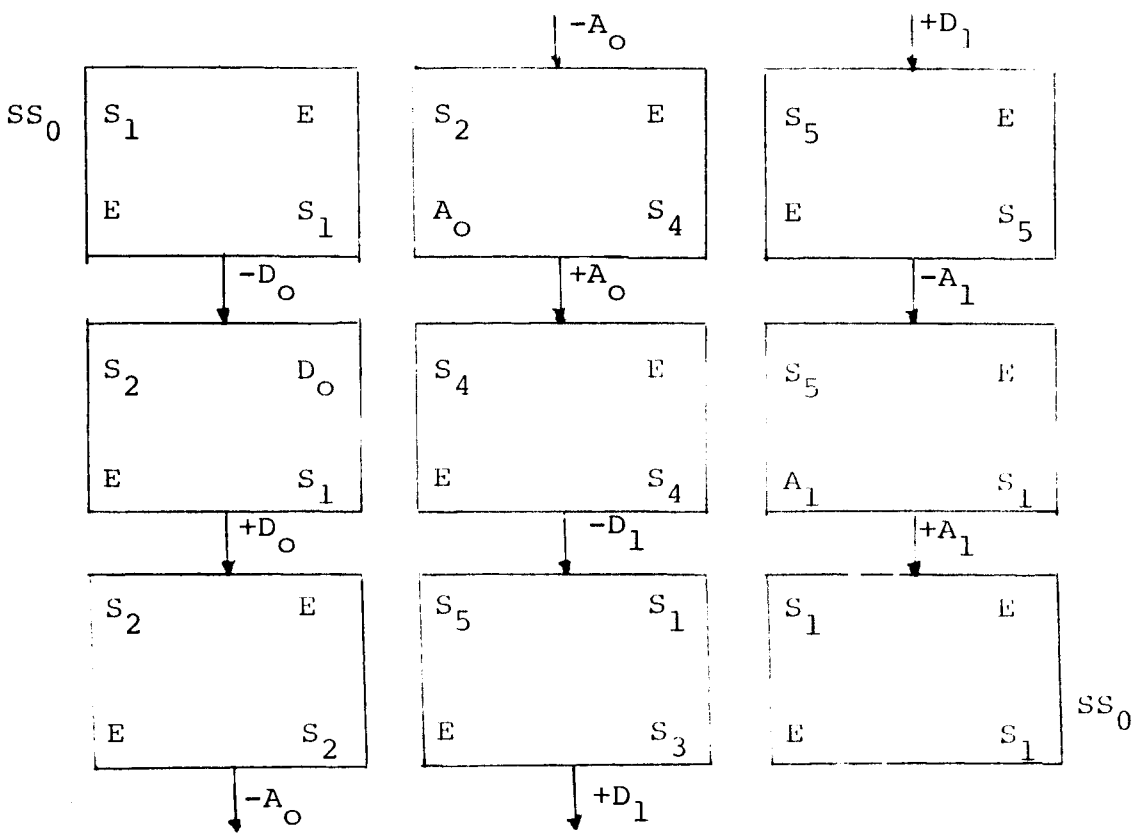


Figura 2.2.

Devido a simplicidade deste protocolo, cada nó da árvore de perturbação tem apenas um filho. Isto significa que para cada estado da comunicação transmissor-receptor existe apenas uma transição possível, seja esta transmissão ou de recepção. No caso geral temos n filhos, significando que temos n transições possíveis para a comunicação.

Analisando a figura 2.2, observamos que o protocolo do bit alternado não apresenta deadlocks, recepções não previstas e ambiguidade de estados. Porém, apresenta algumas interações não executáveis: transição de S_2 para S_3 no transmissor devido à recepção de A_1, transição de S_1 para S_6 no receptor devido à recepção de D_1, etc. Isto não é um erro, pois estas transições foram projetadas para serem realizadas apenas em condições de erro, e além disso, essas interações são "ferramentas para se recuperar dessas condições de erro".

2.2. NÍVEL 2.

O diagrama de estado proposto para este nível é o seguinte: (Ver seção 1.2.3.1).

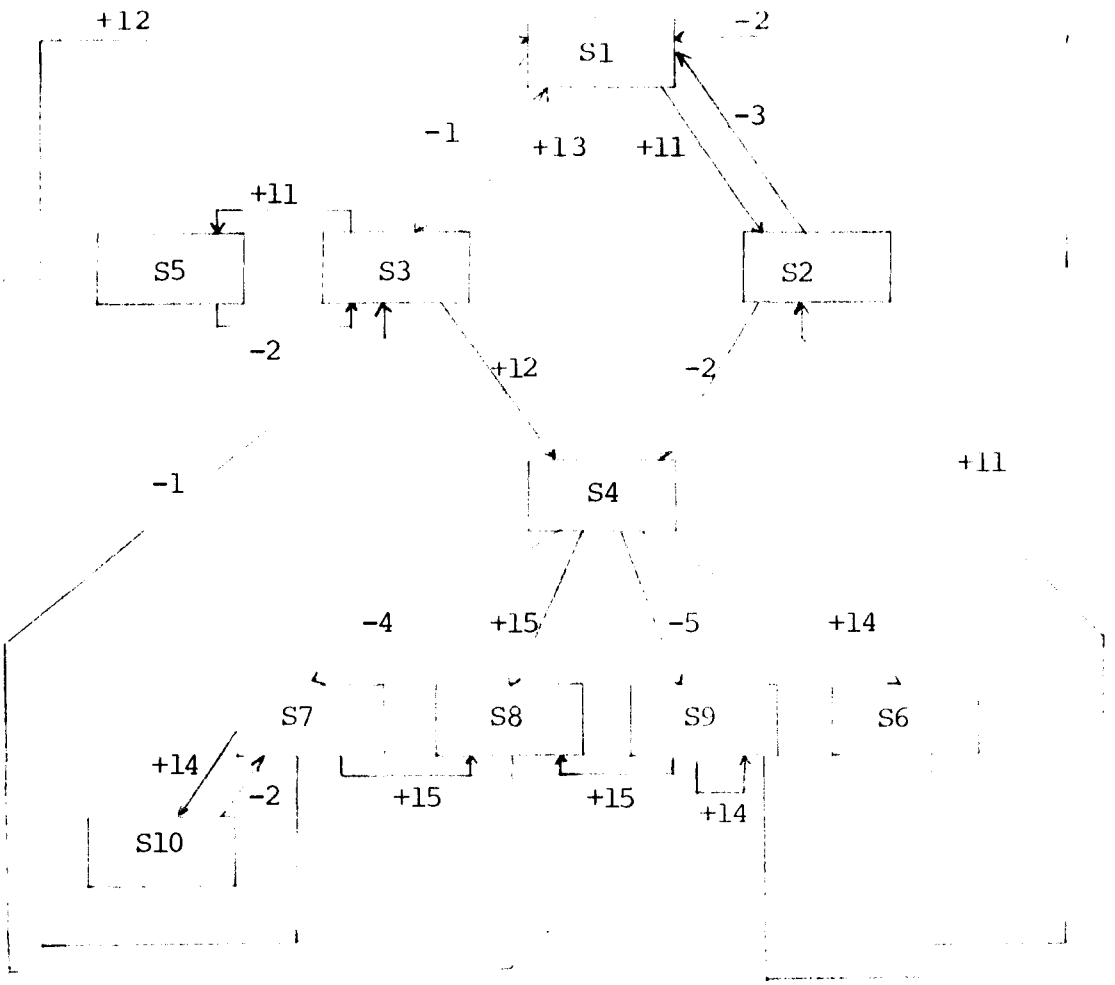


Figura 2.3

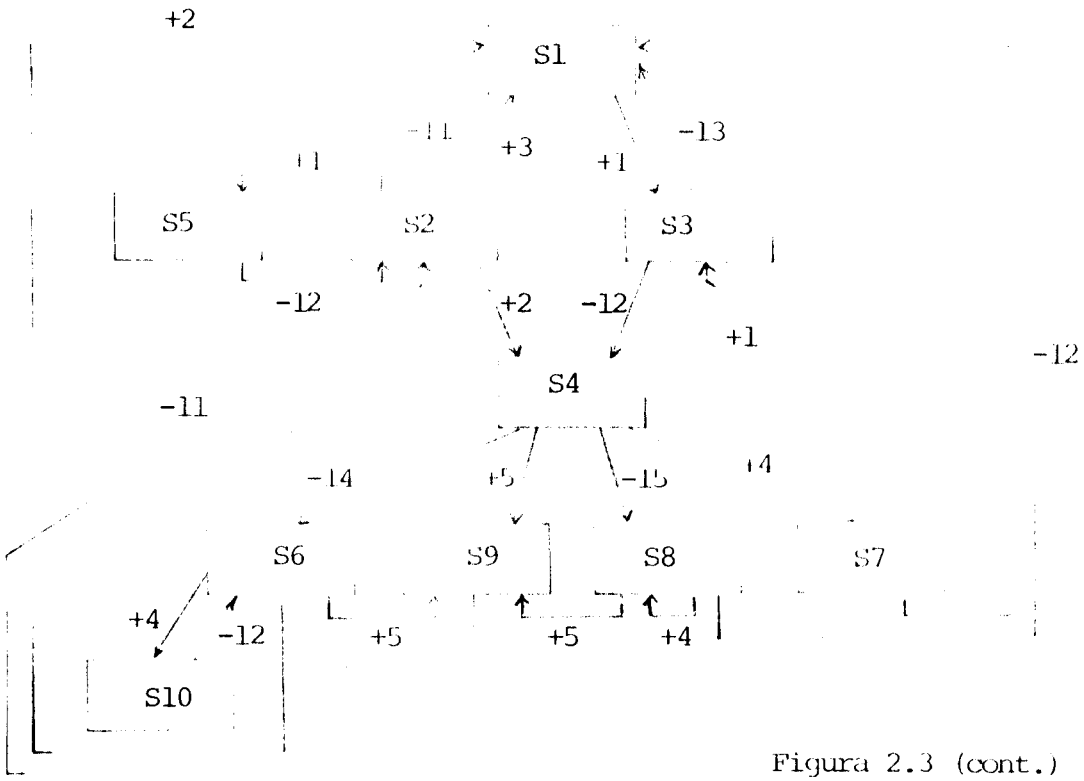


Figura 2.3 (cont.)

onde os quadros tem um número correspondente de acordo com a seguinte tabela:

	DTE → DCE	DCE → DTE
SABM	1	11
UA	2	12
DM	3	13
DISC	4	14
FRMR	5	15

e + significa recepção e - significa transmissão.

Poderíamos também representar estes diagramas através de um conjunto de regras. Para o DTE teríamos:

```
DTE = {( S1, rec-SA  , S2 );      ( S4, rec-DISC, S6 );
       ( S2, env-UA,   S4 );      ( S6, env-UA,   S1 );
       ( S2, env-DM,   S1 );      ( S4, env-I,    S4 );
       ( S1, env-SABM, S3 );      ( S4, env-RR,   S4 );
       ( S3, rec-DM,   S1 );      ( S4, env-RNR,  S4 );
       ( S3, rec-UA,   S4 );      ( S4, env-REJ,  S4 );
       ( S3, rec-SABM, S5 );      ( S4, rec-I,    S4 );
       ( S5, env-UA,   S3 );      ( S4, rec-RR,   S4 );
       ( S4, env-DISC, S7 );      ( S4, rec-RNR,  S4 );
       ( S7, rec-UA,   S1 );      ( S4, rec-REJ,  S4 );
       ( S7, rec-DISC, S10);
       ( S7, rec-FRMR, S8 );       Estado inicial = S1}
       ( S10,env-UA,   S7 );
       ( S4, rec-FRMR, S8 );
       ( S8, env-SABM, S3 );
       ( S4, env-FRMR, S9 );
       ( S9, rec-FRMR, S8 );
       ( S9, rec-DISC, S9 );
       ( S9, rec-SABM, S2 );
```

onde a tripla (S_a, evento 1, S_b) significa que o DTE estava no estado S_a e que passou para o estado S_b ao ocorrer o evento 1. Definimos como evento apenas a recepção e a transmissão de quadros (rec-MEN e env-MEN respectivamente).

O estado inicial também é definido. Para o DCE temos um conjunto de regras análogo.

As últimas oito regras não aparecem como transições no diagrama da figura 2.3, pois estas se relacionam com o envio e a recepção destes quadros só tem sentido no estado S4 e não causam a mudança para um novo estado.

A figura 2.4 mostra a árvore de perturbação construída para a análise do protocolo.

Não temos deadlocks neste nível, pois não obtivemos nenhum nó da árvore de perturbação que tenha seus dois canais vazios e que não possua nenhum evento (transição) a realizar.

Não temos recepções não especificadas, pois não obtivemos nenhum estado do sistema sem transições (eventos) para absorver elementos que estavam em um dos dois canais.

Temos estados ambíguos, ou seja, dois estados do sistema, ambos com os dois canais vazios e um estado comum na diagonal. São eles

S8	E
E	S8

S8	E
E	S9

e

S8	E
E	S9

S9	E
E	S9

Porém isto não significa erro. A existência destes estados ambíguos é decorrente do mecanismo que utilizamos para resolver o problema da colisão do envio de um quadro FRMR por um computador e o envio de um DISC pelo outro computador. O sistema permanece temporariamente neste estado. A recepção dos quadros pelos dois computadores tira o sistema desse estado de ambiguidade.

Não temos interações não executadas, pois todas as transições do diagrama de estado aparecem na árvore de perturbação construída.

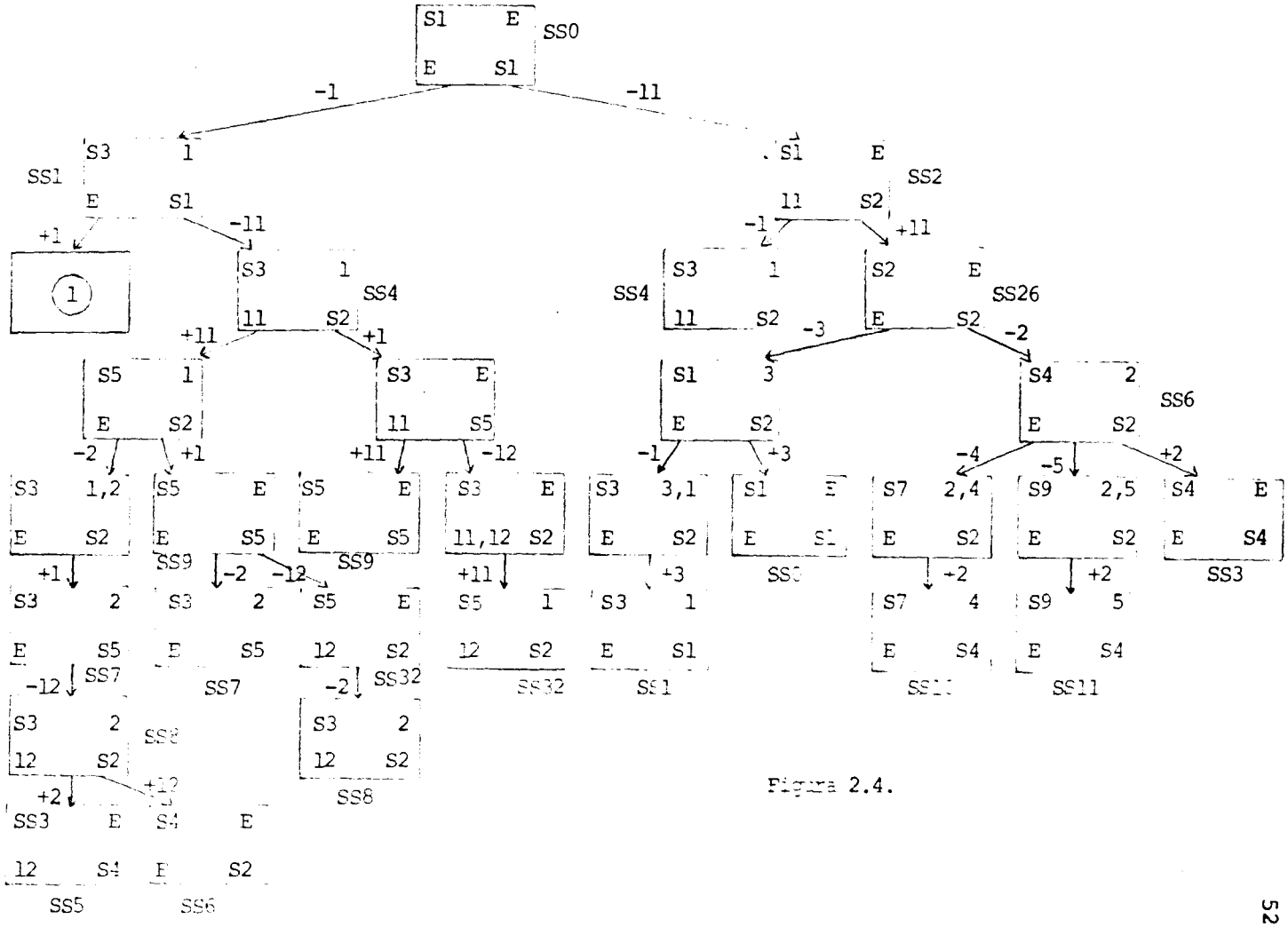

S1 E
E S1
SS0
-1
-11
S3 1
SS1
E S1
+1
-11
1
S3 1
11 S2
SS4
+11
+1
S5 1
E S2
S3 E
11 S5
-2
+1
+11
-12
S3 1,2
E S2
S5 E
E S5
SS9
S5 E
E S5
SS9
S3 E
11,12 S2
+1
-2
-12
+11
S3 2
E S5
S3 2
E S5
S5 E
12 S2
S5 1
12 S2
-12
SS7
SS7
-2
SS32
SS32
S3 2
12 S2
SS8
S3 2
12 S2
SS8
+2
+12
SS3 E
12 S4
S4 E
E S2
SS5
SS6
S1 E
11 S2
SS2
-1
+11
S3 1
11 S2
SS4
S2 E
E S2
SS26
-3
-2
S1 3
E S2
S4 2
E S2
SS6
-1
+3
-4
-5
+2
S3 3,1
E S2
S1 E
E S1
SS8
S7 2,4
E S2
S9 2,5
E S2
S4 E
E S4
SS3
+3
+2
+2
S3 1
E S1
SS1
S7 4
E S4
S9 5
E S4
SS11
SS11

Figura 2.4.

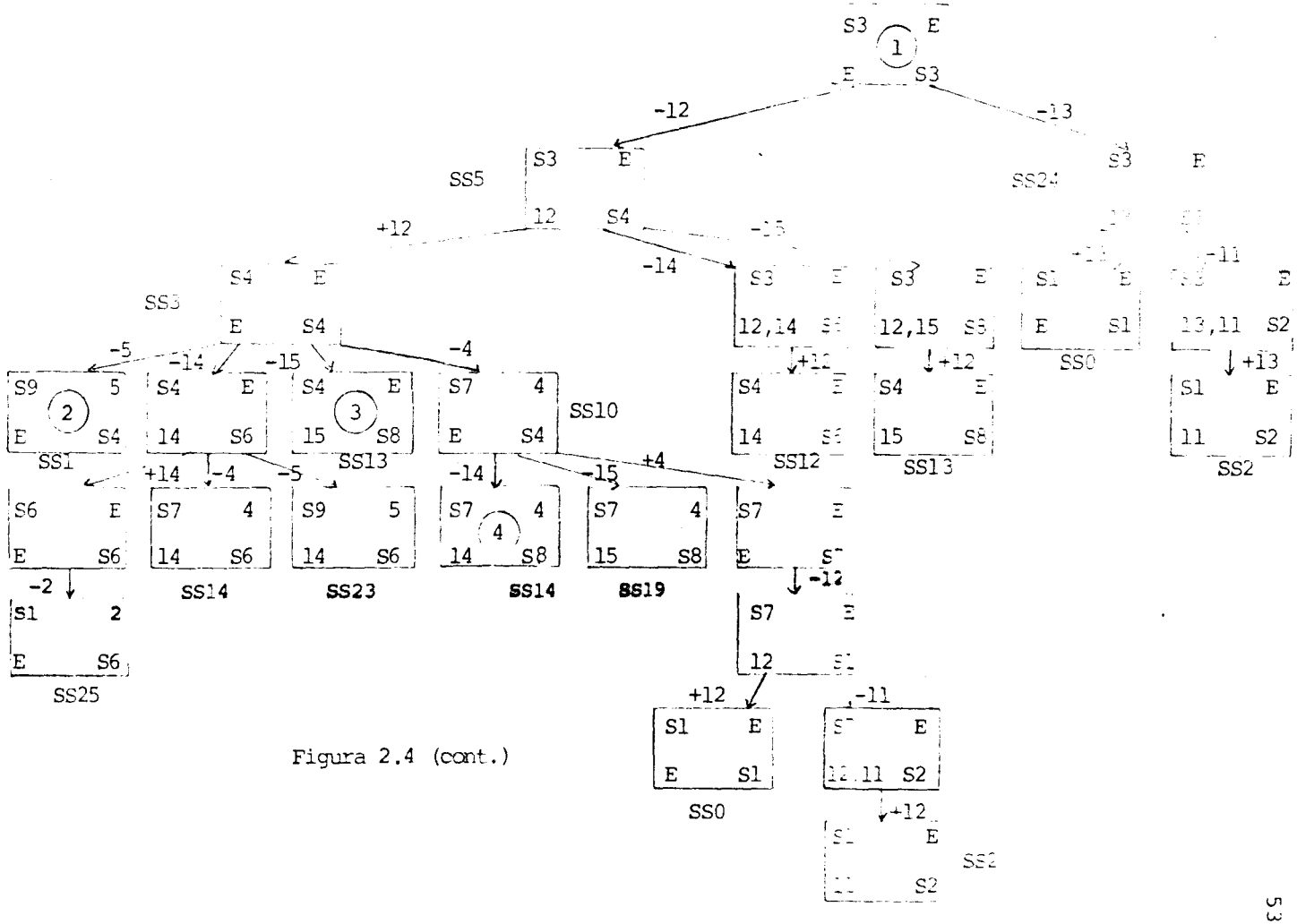


Figura 2.4 (cont.)

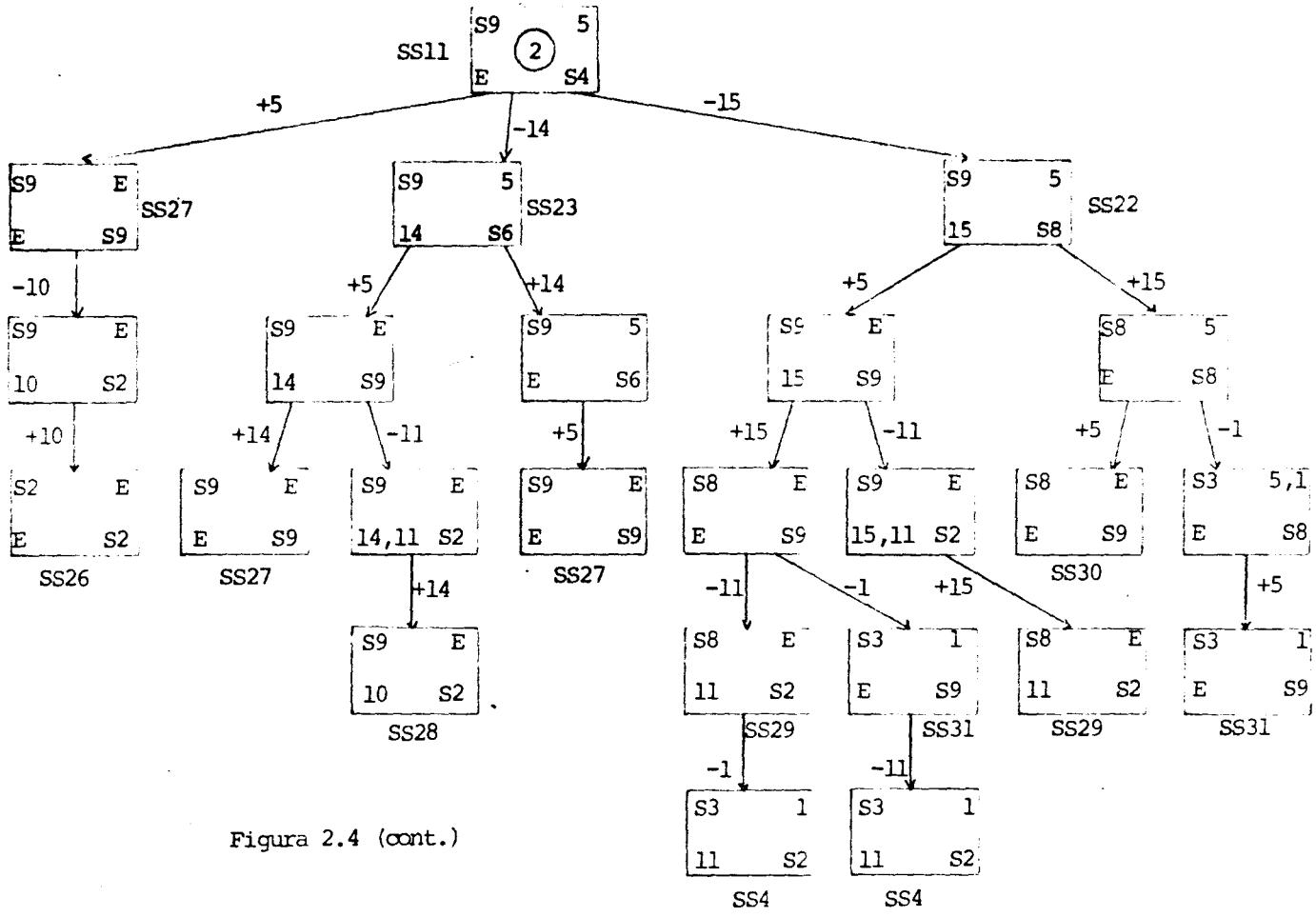


Figura 2.4 (cont.)

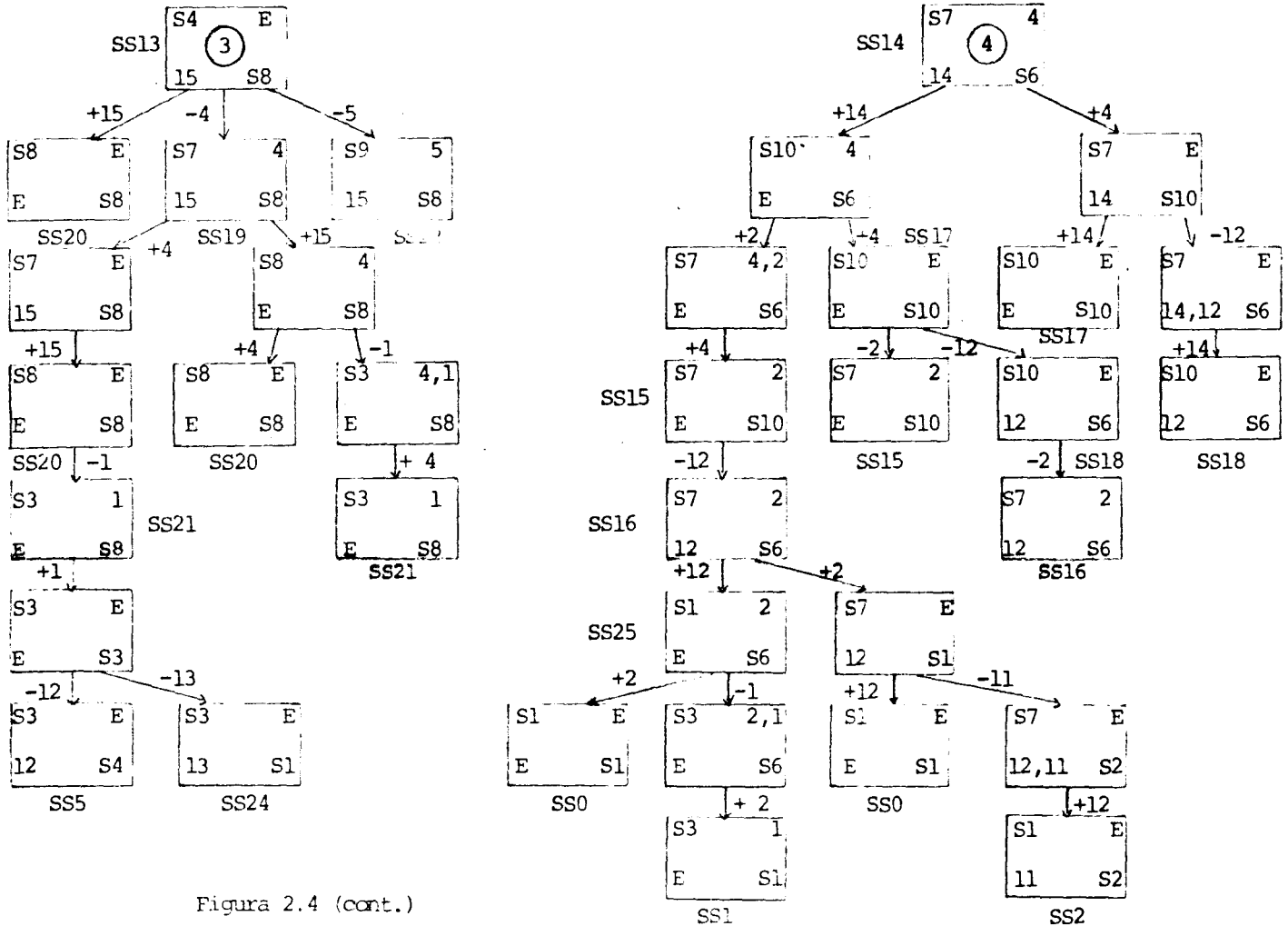


Figura 2.4 (cont.)

2.3. NÍVEL 4.

O diagrama de estado proposto para este nível é o seguinte: (ver seção 1.2.3.3).

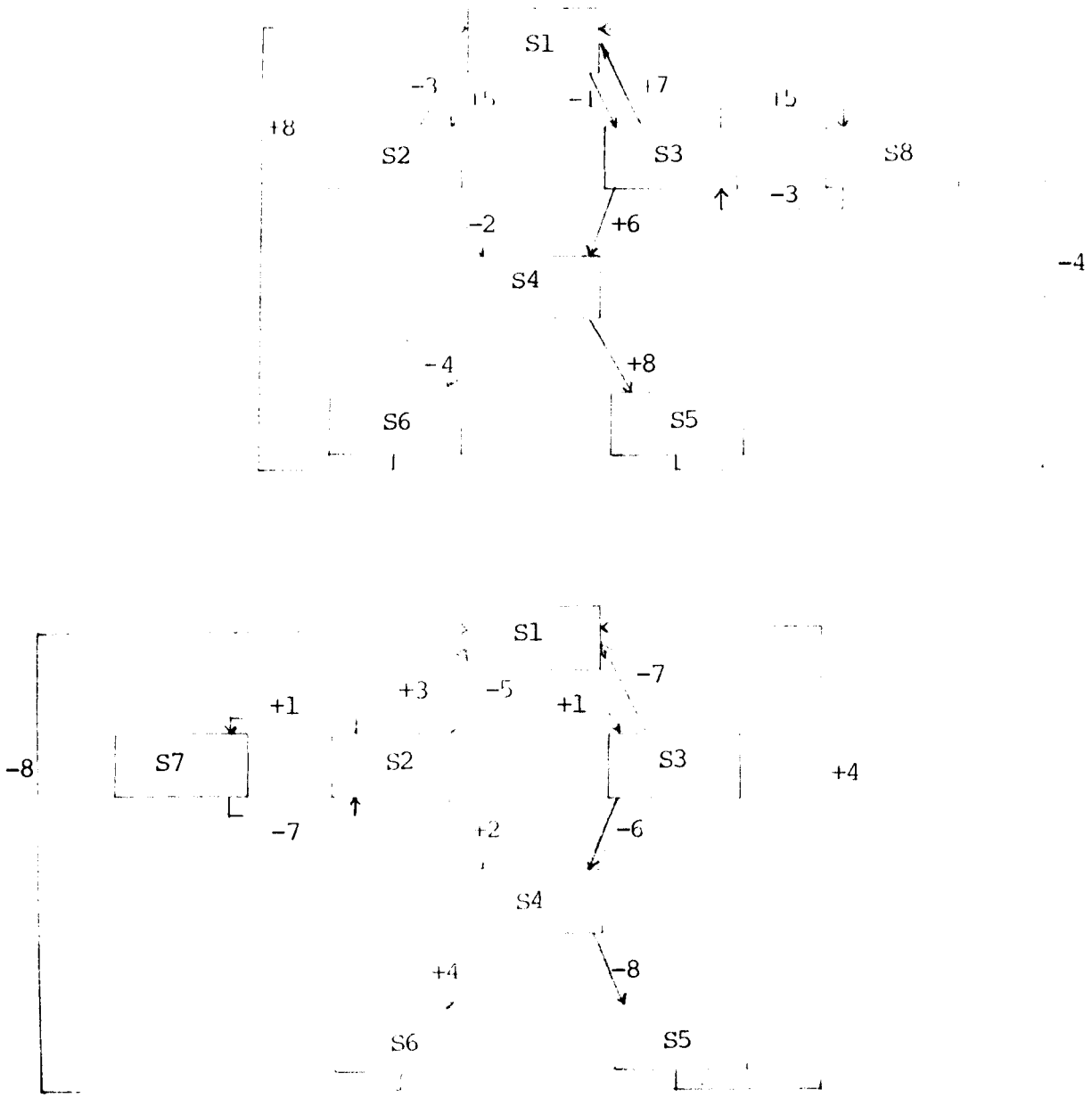


Figura 2.5

onde as mensagens tem um número correspondente de acordo com a seguinte tabela:

	LOC → REM	REM → LOC
CONEX	1	5
CONEFET	2	6
CONREJ	3	7
DESCONEX	4	8

e + significa recepção e - significa transmissão.

Poderíamos também representar estes diagramas através de um conjunto de regras. Para o DTE local teríamos:

```
DTE local = { ⟨ S1, rec-CONEX,    S2 ⟩ ;
              ⟨ S2, env-CONEFET, S4 ⟩ ;
              ⟨ S2, env-CONREJ,  S1 ⟩ ;
              ⟨ S1, env-CONEX,   S3 ⟩ ;
              ⟨ S3, rec-CONREJ,  S1 ⟩ ;
              ⟨ S3, rec-CONEX,   S8 ⟩ ;
              ⟨ S8, env-CONREJ,  S3 ⟩ ;
              ⟨ S3, rec-CONEFET, S4 ⟩ ;
              ⟨ S4, rec-DESCONEX,S6 ⟩ ;
              ⟨ S6, rec-DESCONEX,S1 ⟩ ;
              ⟨ S4, rec-DESCONEX,S5 ⟩ ;
              ⟨ S5, env-DESCONEX,S1 ⟩ ;
```

```
( S4, rec-DADO,      S4 );
( S4, env-DADO,      S4 );
( S6, rec-DADO,      S6 );
( S5, env-DADO,      S5 );
( S4, rec-CONFDADO,  S4 );
( S4, env-CONFDADO,  S4 );
( S6, env-CONFDADO,  S6 );
( S5, rec-CONFDADO,  S5 );

 Estado inicial = S1}
```

onde a tripla $(S_a$, evento 1, $S_b)$ significa que o DTE local estava no estado S_a e que passou para o estado S_b ao ocorrer o evento 1. Definimos como evento apenas a recepção e a transmissão de mensagens (rec-MEN e env-MEN respectivamente). O estado inicial também é definido.

Para o DTE remoto temos um conjunto de regras análogo.

As últimas oito regras não aparecem como transições no diagrama da figura 2.5. Estas regras estão relacionadas com o envio e a recepção de pacotes DAD e CONF.

O envio da mensagem DAD é válido apenas nos estados S4 e S5 e da mensagem CONF nos estados S4 e S6.

A recepção da mensagem DAD é válida apenas dos estados S4 e S6 e da mensagem CONF nos estados S4 e S5.

Estas recepções e transmissões não alteram o estado do receptor e transmissor respectivamente.

A árvore de perturbação construída para a análise do protocolo está mostrada na figura 2.6.

Não temos deadlocks neste nível, pois não obtivemos nenhum nó da árvore de perturbação que tenha seus dois canais vazios e que não possua nenhum evento (transição) a realizar.

Não temos recepções não especificadas, pois não obtivemos nenhum estado do sistema sem transições (eventos) para absorver elementos que estavam em um dos canais.

Não temos estados ambíguos, pois não obtivemos dois estados so sistema, ambos com os dois canais vazios e um estado comum na diagonal.

Não temos interações não executadas, pois todas as transições do diagrama do estado aparecem na árvore construída.

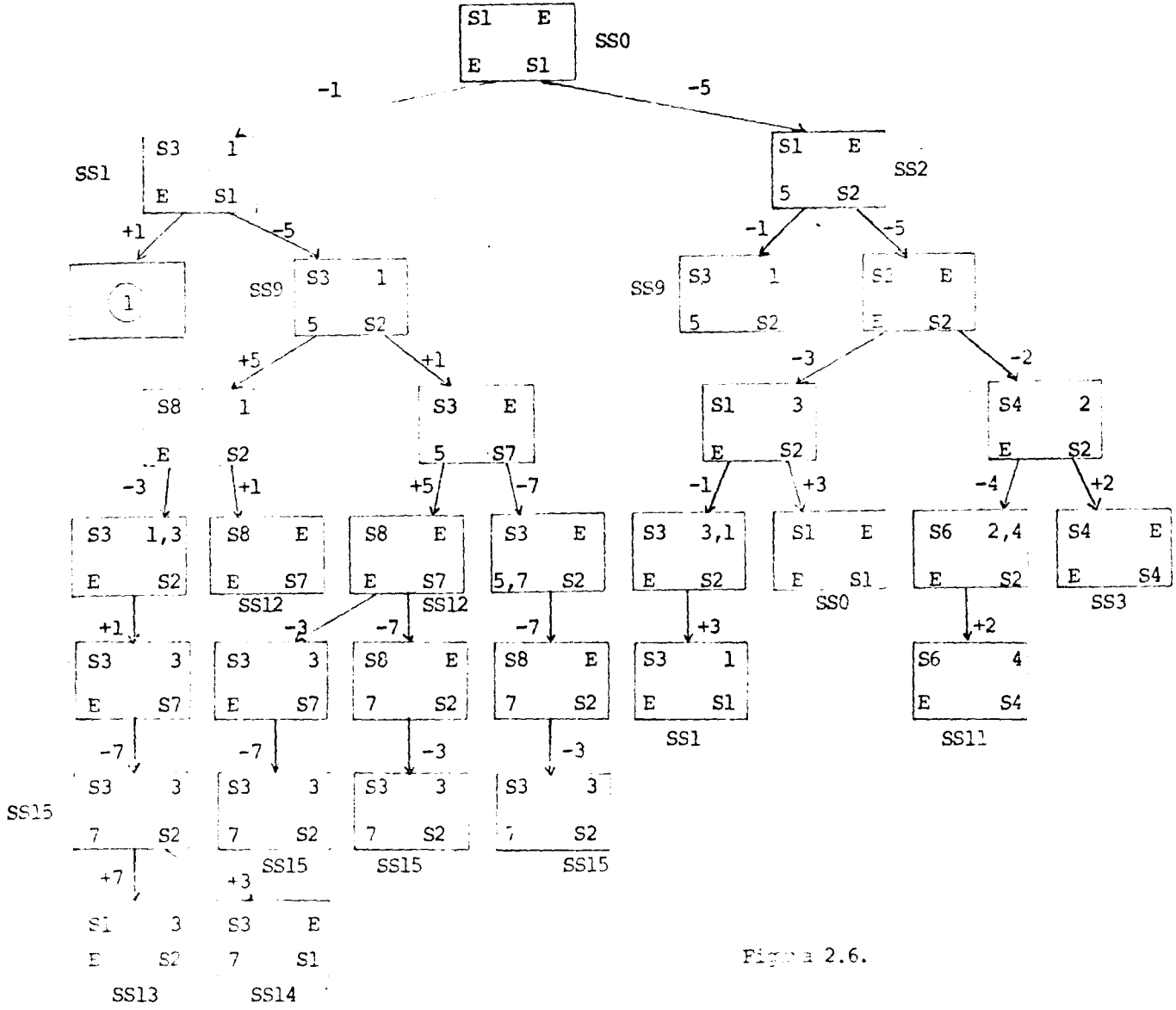

S1 E
E S1
SS0
-1
-5
SS1
S3 1
E S1
+1
-5
1
SS9
S3 1
5 S2
+5
+1
S8 1
E S2
-3
+1
S3 1,3
E S2
+1
S3 3
E S7
-7
S3 3
7 S2
+7
S1 3
E S2
SS13
SS15
S8 E
E S7
SS12
-3
-7
S3 3
E S7
-7
S3 3
7 S2
+3
S3 E
7 S1
SS14
SS15
S3 E
5 S7
+5
-7
S8 E
E S7
SS12
-7
S8 E
7 S2
-3
S3 3
7 S2
SS15
S3 E
5,7 S2
-7
S8 E
7 S2
-3
S3 3
7 S2
SS15
SS2
S1 E
5 S2
-1
-5
SS9
S3 1
5 S2
S1 E
E S2
-3
-2
S1 3
E S2
-1
+3
S3 3,1
E S2
+3
S3 1
E S1
SS1
S1 E
E S1
SS0
S4 2
E S2
-4
+2
S6 2,4
E S2
+2
S6 4
E S4
SS11
S4 E
E S4
SS3

Figure 2.6.

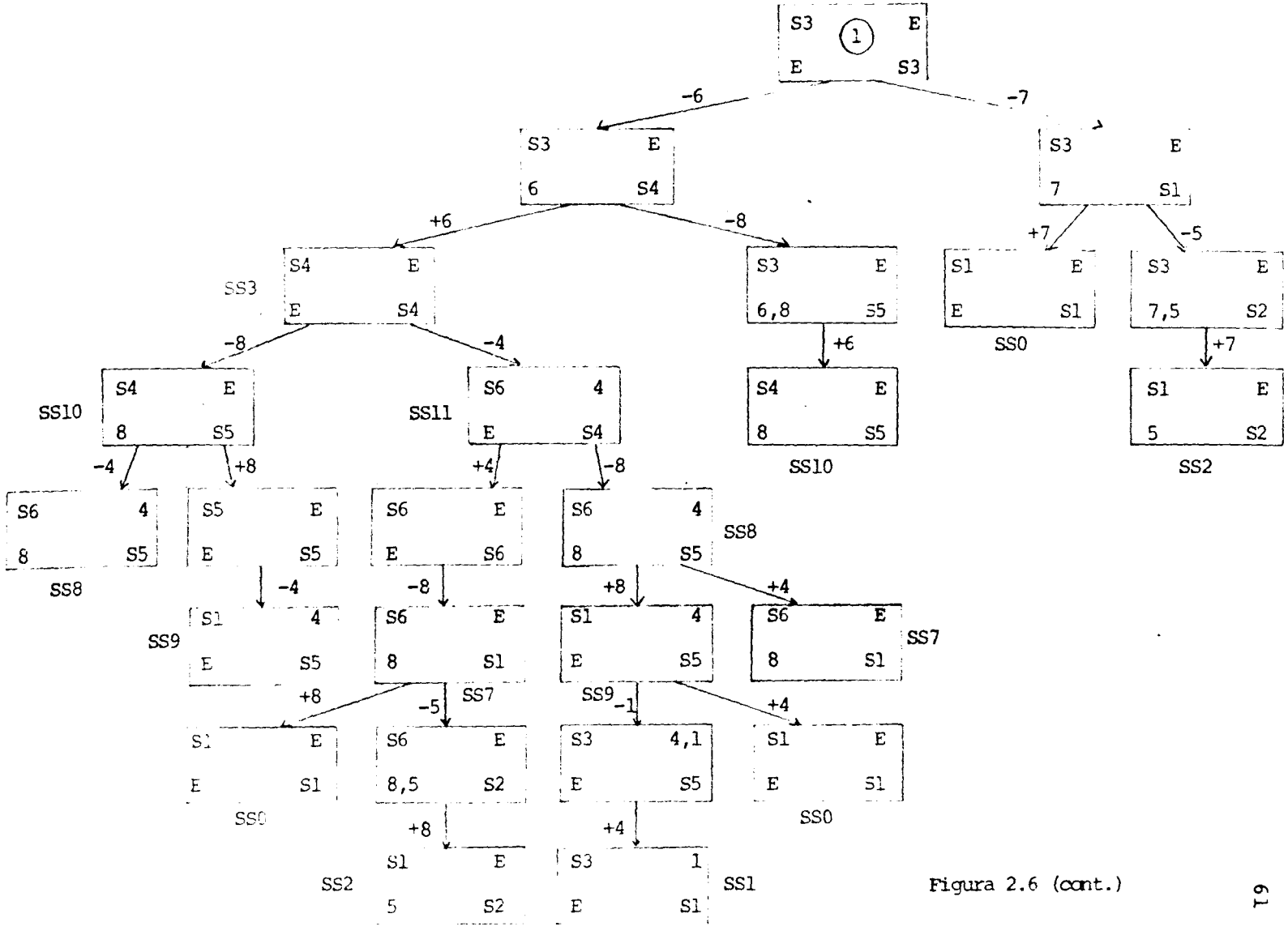


Figura 2.6 (cont.)

CAPÍTULO III

COMUNICAÇÃO E SINCRONIZAÇÃO DE PROCESSOS

3.1. INTRODUÇÃO

No capítulo I descrevemos o conceito de níveis para um protocolo segundo a padronização da ISO. Um ponto importante deste esquema, que deve ser percebido, é o fato de que a execução de um nível pode ser feito independente dos outros níveis, a menos da mensagem recebida no início desta execução e da mensagem enviada ao final desta execução, se necessária. Um nível recebe e envia mensagens para outros níveis, mas não influi no processamento destas mensagens nos outros níveis. Portanto cada nível do protocolo X-25 poderá ter "vida própria", ou seja, ser um processo. Esses diversos níveis, que estamos tratando como processos, precisam comunicar entre si. Por exemplo, o nível 2 se comunica com os níveis 1 e 3 e o nível 3 se comunica com os níveis 2 e 4. Portanto estes processos podem ser executados paralelamente, mas não de maneira incomunicável. Haverá comunicação e consequentemente uma sincronização entre os processos.

3.2. LINGUAGENS PARA PROGRAMAÇÃO CONCORRENTE

A primeira característica necessária para a linguagem de

UNICAMP
BIBLIOTECA CENTRAL

alto nível que devemos escolher para implementar o protocolo é ter suporte para processos concorrentes.

Muito se tem pesquisado na área de linguagens com suporte para processos concorrentes para:

1) garantir a integridade (consistência) das variáveis compartilhadas;

2) ter um esquema eficiente de escalonamento de processos (sincronização eficiente).

As soluções encontradas podem ser classificadas em três grandes categorias:

1) Sincronização através de semáforos, eventos, condições e monitores (Dijkstra [28] e [29], Brinch Hansen [26], Wirth [40], Hoare [32], etc).

Por exemplo, na proposta de Brinch Hansen, utilizando eventos, temos os comandos wait (e) e signal (e).

a) wait (e) - o processo fica bloqueado a espera do evento e;

b) signal (e) - o processo notifica outro processo da ocorrência do evento e

A existência destes comandos implica na existência de uma fila de processos que estavam esperando por e.

2) Sincronização através de troca de mensagens (Brinch Hansen [27], Harland [31], etc).

Um exemplo de solução baseado em troca de mansagens é o sistema RC-4000 da Dinamarca (1970). Neste sistema temos primitivos do tipo send-message (receiver, message, buffer) wait-answer (result, answer, buffer), wait-message (sender, message, buffer) , wait-answer (result, answer, buffer), wait-event e outros.

Implementações baseadas em mailbox possuem primitivos wait-msg (mailbox, pointer) send-msg (mailbox, point) e também possuem filas de processos associados ao mailbox.

3) Sincronização através de "randez-vous" (Hoare [33] no CSP; ADA [35] e [38], etc).

Estas soluções implicam no envio de mensagens entre processos e na espera de ocorrência do "randez-vous" (encontro) entre esses processos. Um processo A em determinado ponto ao necessitar de outro processo B, envia a este segundo um pedido de randez-vous e aguarda a resposta. O processo B quando tiver disponível a mensagem necessária para a continuidade do processo A confirma o randez-vous e envia esta mensagem. Neste ponto, os dois processos se "encontram", ou seja, realizam o "randez-vous". Depois deste encontro (sincronização), os dois processos podem continuar suas respectivas computações.

3.3. MONITOR × TROCA DE MENSAGENS

Monitor é um conjunto de procedimentos que operam sobre

variáveis comuns a vários processos. Utiliza o conceito de tipo de dados abstratos.

Os tipos de dados abstratos possuem dois componentes: objetos (estruturas de dados) e operações. As únicas operações que podem ser efetuadas sobre o objeto são definidas pelo tipo de dados abstratos. Uma encapsulação nos dois sentidos é requerida: o acesso externo a um objeto tem que ser validado (permitido) e os procedimentos do tipo de dados não podem acessar objetos externos não declarados explicitamente.

Portanto o estado de um objeto em um dado momento da computação pode ser verificado a partir do seu estado inicial e do conjunto de operações permitidas efetuadas até este momento.

Algumas linguagens que implementam tipos de dados abstratos são CLU, MESA, ALPHARD, GYPSY, MODULA 2, ADA, PASCAL CONCORRENTE.

Um monitor é um caso particular de tipo de dados abstratos, onde objetos são variáveis comuns aos vários processos e as operações são as Regiões Críticas exigidas pela aplicação. Portanto, um monitor é um conjunto de Regiões Críticas operando sobre variáveis comuns. Um processo, para ter acesso a uma região crítica, chama um procedimento do monitor.

É necessário, em geral, exclusão mútua no tempo para a execução de procedimentos do monitor (Regiões Críticas) por processos concorrentes. A integridade (consistência) das variáveis comuns do monitor é garantido por esta exclusão mútua no tempo e

pela encapsulação descrita anteriormente.

Sistemas baseados em troca de mensagens estabelecem canais específicos de comunicação, através dos quais mensagens são trocadas entre processos. Segundo Lauer e Needham em [12] os processos são associados aos recursos do sistema e aplicações são codificadas sob a forma de dados passados entre processos como mensagens.

Lauer e Needham mostram a dualidade entre os dois modelos , ou seja, se o estilo é preservado, uma aplicação pode ser transformada de um modelo para outro através de uma simples substituição de construções (sintática), sem nenhuma alteração na semântica, nas estruturas de dados e nos algoritmos usados.

A dualidade é obtida relacionando-se os seguintes conceitos:

TROCA DE MENSAGENS	PROCEDIMENTOS (MONITORES)
processo	monitor
canais de mensagens	procedimentos externos (importados)
portos	procedimentos exportados
send-message, await replay	chamada de procedimento
send-message,...,await message	FORK ... JOIN
send-replay	Retorno de procedimento
wait-for-message selective	wait(e), signal (e).

Ao nosso ver, dois fatores devem ser analisados para escolha do modelo a ser baseado o sistema:

a) Tempo para executar os primitivos, o qual depende da arquitetura da máquina (número de instruções pode ser diferente para máquinas distintas).

2) Características da aplicação.

3.4. IMPLEMENTAÇÕES DO PROTOCOLO X-25

Para motivarmos a nossa proposta, nesta seção descrevemos duas idéias de implementações do protocolo X-25, as quais aproveitam os recursos da programação concorrente. A primeira é de Michael Stanton [16]. A segunda é uma implementação realizada por Gregor V. Bochmann e Tankoano Joachim no Canadá [3].

No primeiro trabalho aplica-se a dualidade entre o uso de monitores e o uso de troca de mensagens, como ferramenta na construção de programas concorrentes, no sentido de que um sistema pode ser idealizado num modelo e implementado em outro. Para isso desenvolve-se uma representação gráfica única para os dois modelos, e conforme a interpretação dessa representação, ela pode representar um modelo ou outro.

Como primeira versão, e de acordo com a idéia anterior, apresenta-se a representação gráfica para um canal duplex de comunicação de pacotes da figura 3.1.

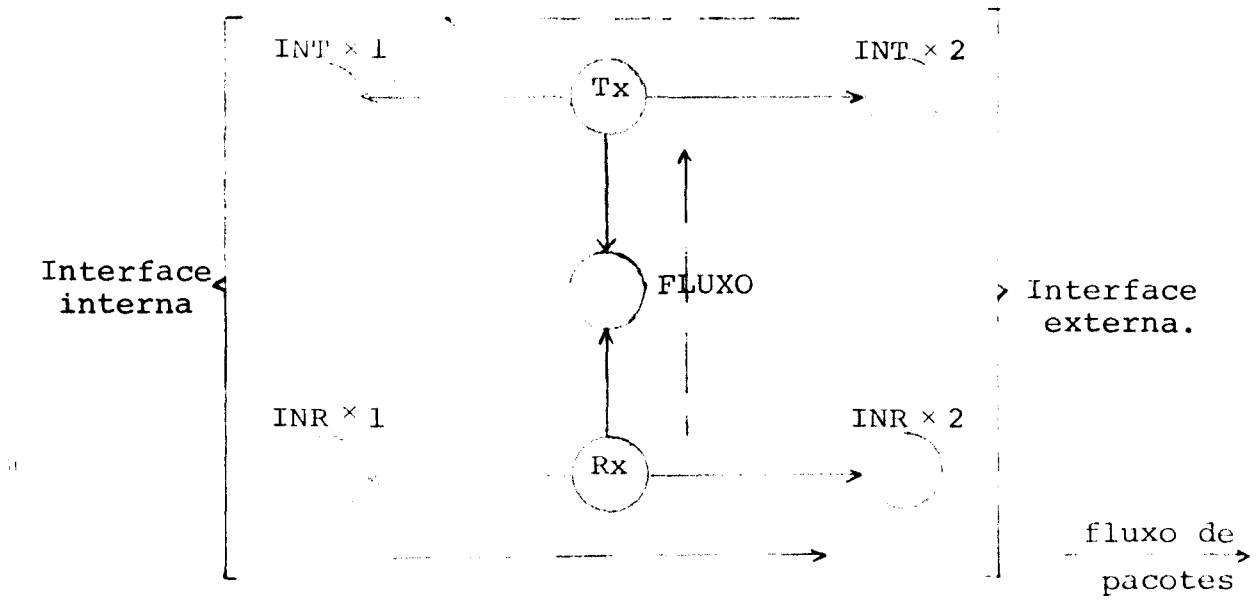


Figura 3.1
(Michael Stanton)

OBSERVAÇÃO: A representação gráfica única deve ser interpretada da seguinte maneira:

MONITORES: a) Cada círculo representa um processo ou um monitor.

b) Cada seta representa o direito de chamar entradas do monitor.

TROCA DE MENSAGENS: a) Cada círculo representa um processo

b) Cada seta representa uma comunicação, com a flecha apontando o recipiente das mensagens, que deverá respondê-las.

INT × 1, INR × 1, INT × 2, INR × 2 e FLUXO são monitores, sendo que o último é usado para controlar os pacotes que transitam pelo canal. Rx e Tx são processos que tratam da recepção e

transmissão de pacotes. A deficiência encontrada foi a seguinte: Tx procura um pacote para computar em INT × 1 e FLUXO, e se não houver, espera neste monitor. Porém pode existir um outro pacote recebido para ser tratado em outro monitor. Perde-se paralelismo devido a esta espera.

A versão final, constituída baseando-se em troca de mensagens e que pode ser implementado baseando-se em monitores, está apresentada na figura 3.2.

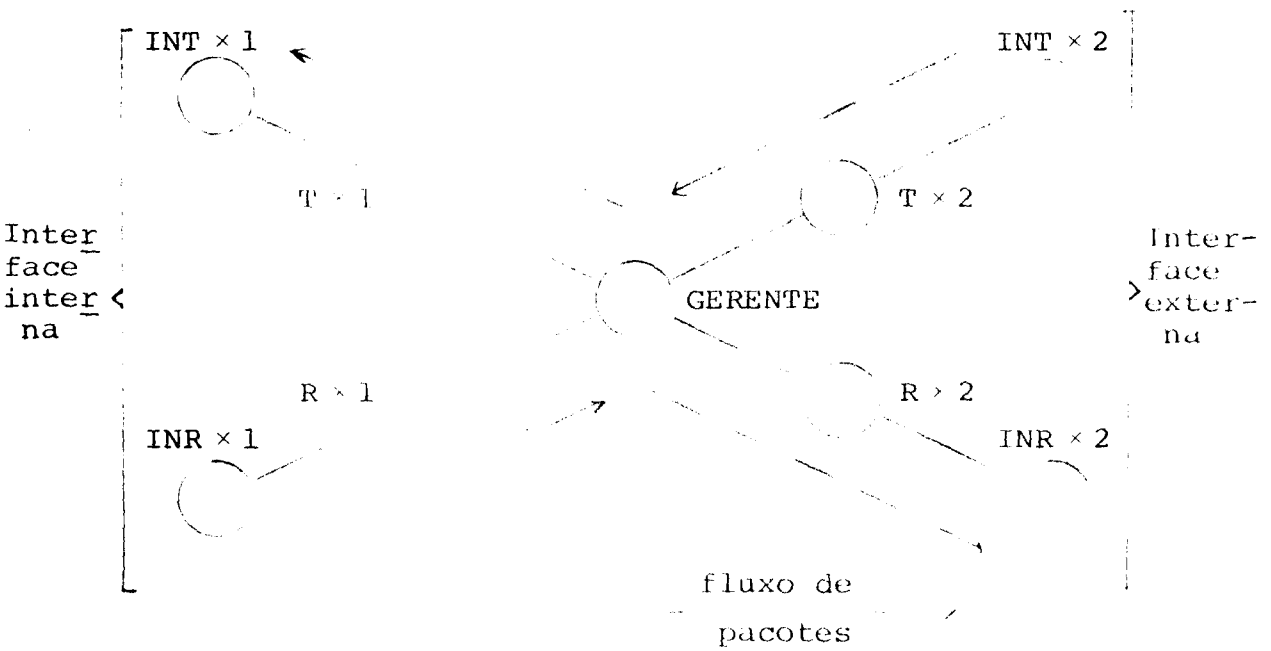


Figura 3.2 (Michael Stanton)

A diferença básica é que R×1, T×1 e FLUXO foram substituídos por um gerente e quatro correios (R×1, R×2, T×1, T×2). Todos os pacotes são processados no gerente.

No segundo trabalho, Bochmann e Joachim desenvolveram uma

especificação do protocolo no qual todo o trabalho de implementação é baseado. Para nossa implementação nos foi útil a especificação do nível 2, a qual comentamos na seção 1.2.3.1.

O Pascal concorrente foi utilizado como linguagem de programação. Utilizou-se, como veremos a seguir, os conceitos de processos paralelos, monitores e classes. Os seguintes pontos foram considerados: a) garantir compatibilidade de implementação com a parte remota; b) implementar várias atividades paralelas.

O estudo foi realizado analisando-se duas maneiras de se obter sincronização e comunicação entre processos: troca de mensagens e monitores:

A figura 3.3 mostra a decomposição da implementação em módulos (processos) interagindo através de troca de mensagens.

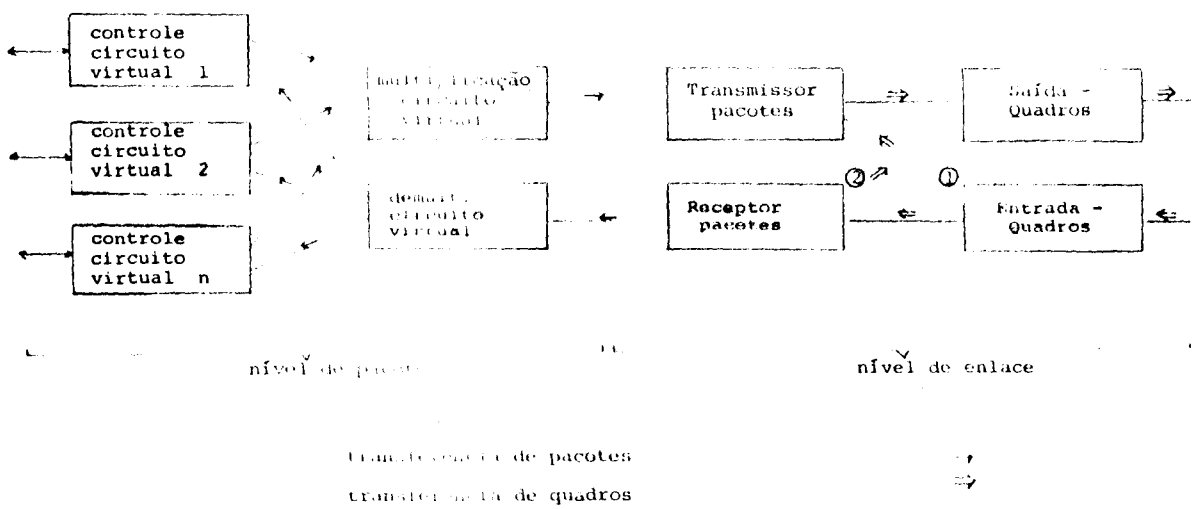


Figura 3.3 (Bochmann e Joachim)

OBSERVAÇÕES: a) Utiliza-se para um mesmo nível módulos distintos para recepção e transmissão, buscando um maior paralelismo.

b) A recepção de um quadro pode causar o envio de outro quadro através de 1 ou 2 .

c) Existem módulos no nível 3 para multiplexar e demultiplexar os diversos circuitos virtuais.

A figura 3.4 mostra a estrutura de implementação em termos de monitores e processos.

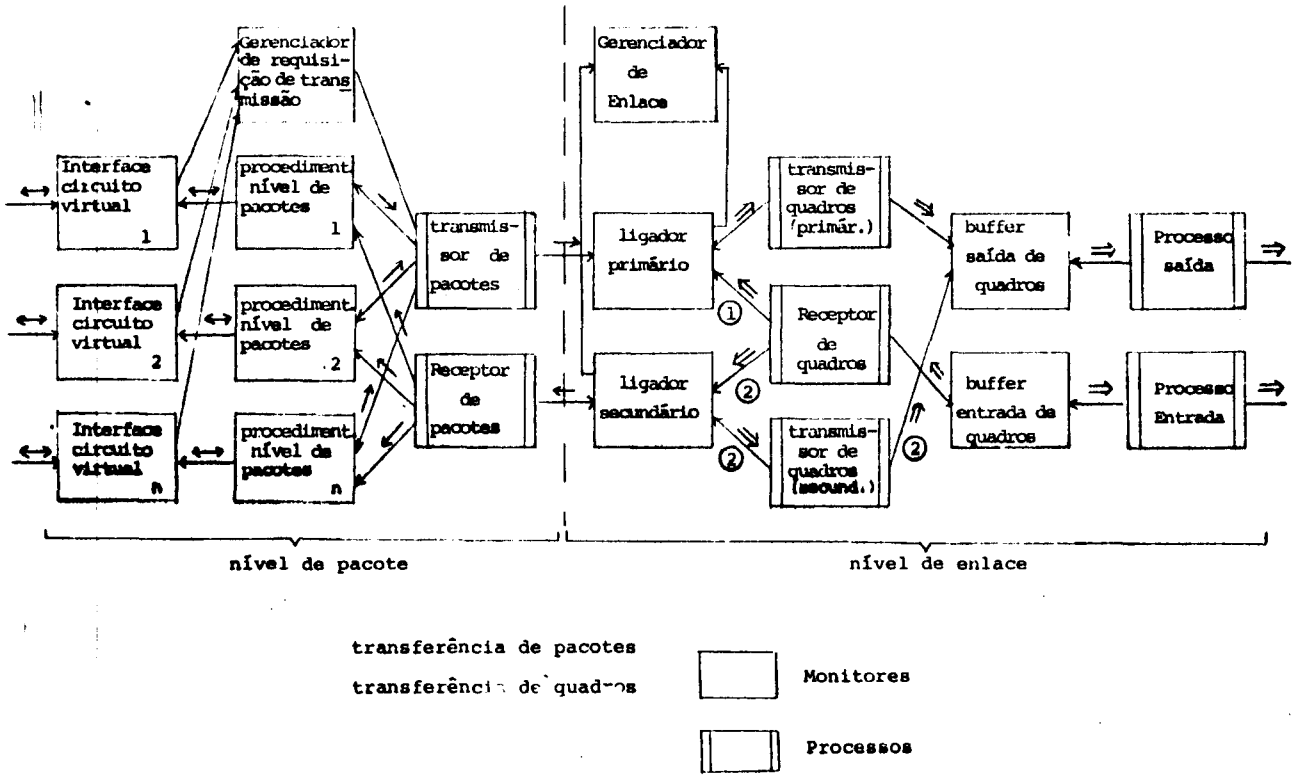


Figura 3.4 (Bochmann e Joachim)

OBSERVAÇÕES: a) Utiliza-se para um mesmo nível processos e monitores distintos para recepção e transmissão, buscando um maior paralelismo.

b) A recepção de um quadro pode causar o envio de outro quadro através de 1 e 2 .

c) Para tratar os procedimentos a nível de pacote de cada um dos circuitos virtuais existem monitores distintos.

d) O monitor gerenciador de requisições de transmissão do nível 3 é o lugar onde diferentes prioridades podem ser introduzidas para diferentes circuitos virtuais.

3.5. LINGUAGEM DE IMPLEMENTAÇÃO-MODULA 2

As linguagens que suportam processos concorrentes, têm geralmente primitivos que nem sempre são adequados a uma aplicação específica, além de não serem de implementação elementar.

A linguagem escolhida foi Módula 2, pois além de possuirmos um compilador desta linguagem que gera código para o microprocessador 8088, é uma linguagem que permite estruturas de dados complexas e que possui um mecanismo de transferência entre processos muito simples e eficiente. Este mecanismo leva a um bom aproveitamento do paralelismo entre os níveis (processos) do protocolo.

Em Módula 2 os processos são executados como corrotinas . Transfere-se explicitamente o controle de um processo para outro.

Esta é a única sincronização envolvida.

A transferência entre processos é feita através dos comandos NEWPROCESS (p, e, t, pl), TRANSFER (pl,p2) e IOTRANSFER (pl, p2, a).

O comando NEWPROCESS é utilizado para criar um novo processo. p é o procedimento de nível zero e sem parâmetros do programa que queremos transformar em um processo; e é o endereço da base da área de trabalho deste novo processo; t é o tamanho desta área de trabalho e pl é uma variável do tipo processo que será associada a este processo.

O comando TRANSFER (pl, p2) transfere o controle do processo corrente (atribuído então à variável pl) para o processo p2 que necessariamente deve ter sido definido antes por um NEWPROCESS.

O comando IOTRANSFER (pl, p2, a) é utilizado em "drivers" para tratar de interrupções de dispositivos periféricos. Este comando transfere o controle do processo corrente ("driver"),atribuindo-o à variável pl, para o processo p2 interrompido anteriormente à execução do driver. Quando uma interrupção do tipo a ocorre, o processo em execução é suspenso e atribuído à variável p2, e o controle é transferido para o driver correspondente e a primeira instrução executada é a seguinte do IOTRANSFER.

Para executarmos processos concorrentes basta inicialmente definirmos alguns processos através de NEWPROCESS e rotinas de interrupção com IOTRANSFER. A partir daí podemos transferir controle

de um processo para outro através de TRANSFER's.

Se compararmos este mecanismo de suporte para processos concorrentes com os anteriormente vistos, podemos dizer que Módula 2 ataca o problema de transferência entre processos de uma maneira mais primitiva.

Uma observação importante é que esse mecanismo de transferência de processos através dos comandos TRANSFER e IOTRANSFER permite a um nível (processo) depois de executado o que era necessário, transferir o controle para um outro processo conveniente, para que este dê continuidade ao processamento da informação. Com isto evitamos laços de espera desnecessários.

Em [18], Wirth argumenta que as ferramentas fornecidas por Módula 2 são suficientes para a implementação eficiente de mecanismos de comunicação/sincronização mais elaborados como monitores e troca de mensagens. Dois exemplos, em [18] a implementação de monitores em Modula 1 com sincronização através de wait e signal sobre eventos e em [7] Jiri Hoppe apresenta um núcleo (kernel) escrito em Modula 2. A implementação é feita usando-se Mailbox e os primitivos sendmsg (mailbox, msg), waitmsg (mailbox, msg) e waitIO (intvec) são "atômicos". Um processo ao executar o primitivo sendmsg envia uma mensagem apontada por msg para o mailbox mailbox. Um processo ao executar waitmsg espera uma mensagem apontada por msg do mailbox mailbox. Um processo "driver" ao executar waitIO espera por uma interrupção do tipo intvec. A atomicidade destes primitivos é obtida escrevendo-se estes procedimentos em

módulos não interrompíveis. TRANSFER é usado em waitmsg e sendmsg e IOTRANSFER em waitIO. O escalonamento proposto é preemptivo,ou seja, quando uma interrupção ocorre, depois de executada a rotina de interrupção, não necessariamente o controle é devolvido ao processo interrompido. O processo interrompido pode ser continuado depois de vários processos terem sido executados.

Estas são as considerações sobre Módula 2 necessárias até este ponto. Novos conceitos sobre esta linguagem serão introduzidos à medida que necessários. Para uma compreensão total da linguagem recomendamos a leitura de [18] e [11].

CAPÍTULO IV

IMPLEMENTAÇÃO DO PROTOCOLO X-25 NO MICROPROCESSADOR 8088

4.1. SISTEMA PROPOSTO.

Na seção 3.1 definimos cada nível do protocolo como um processo. Um processo adicional deve ser introduzido para o gerenciamento do protocolo. Duas rotinas de interrupção devem cuidar da recepção e transmissão de quadros de / para o nó da rede.

O sistema proposto possuirá portanto, *seis processos* concorrentes: níveis 2, 3 e 4, gerenciador e os dois processos associadoa às rotinas de interrupção.

A transferência entre os processos envolvidos é feito segundo o esquema da figura 1.

O gerenciador pode transferir o controle para os níveis 2, 3 e 4. O nível 2 pode transferir o controle para o gerenciador e para o nível 3. O nível 3 para o gerenciador e para os níveis 2 e 4 e finalmente o nível 4 para o gerenciador e para o nível 3.

Vamos estudar agora como este esquema funciona.

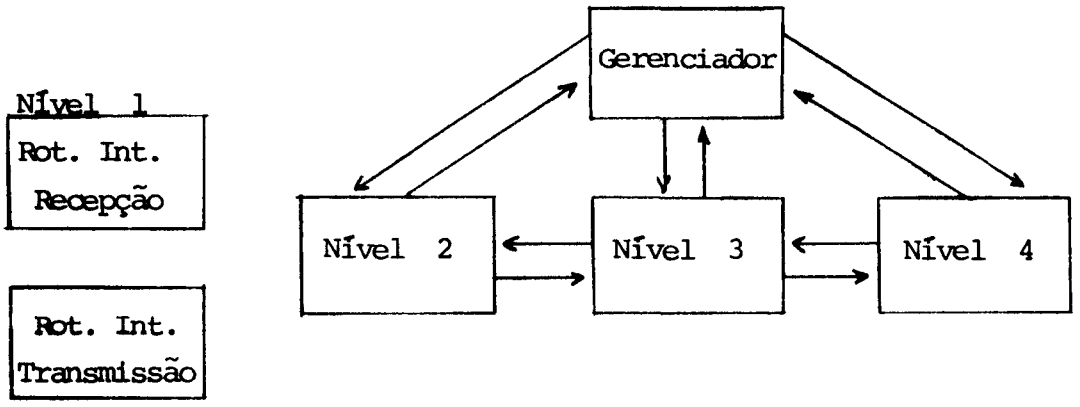


Figura 4.1

A interface HDLC 8273 ao receber um quadro do nó da rede gera uma interrupção no microprocessador e a Rotina de Interrupção de Recepção passa a ser executada. Dois casos são possíveis:

a) Se o processo interrompido foi o nível 2, a Rotina de interrupção simplesmente insere este quadro numa fila de recepção e retorna.

b) Em caso contrário (processo interrompido é o nível 3 ou nível 4 ou gerenciador) esta rotina transfere o controle para o nível 2 se o quadro implicar em uma mudança de comportamento (estado) do nível 2. É o caso dos quadros de controle e de supervisão. Exemplos típicos são as recepções dos quadros de controle e dos quadros REJ e FRMR que influem na fila de transmissão. Neste caso a fila de

transmissão é atualizada suprimindo-se os quadros de informação que seriam rejeitados pelo nó da rede e inserindo-se os quadros de informação que precisam ser retransmitidos. Isto é feito para que se aumente a velocidade de transferência de informação correta à rede. O nível 2 após terminar o procedimento acima transfere o controle para o processo interrompido anteriormente. Se o quadro não for de controle ou supervisão (portanto é de informação), a rotina simplesmente insere o quadro na fila de recepção e devolve o controle ao processo interrompido.

A existência dos casos a e b nos permite ter no máximo dois processos interrompidos, e ainda mais, se houver dois processos interrompidos, o segundo necessariamente será o nível 2. Se era o nível 2 que estava em execução quando ocorreu uma interrupção, a rotina correspondente executada de maneira não interrompível não transfere controle para outro processo e devolve o controle para o nível 2 no final de sua execução. Temos portanto, apenas um processo interrompido, ou seja, o nível 2. Se não era o nível 2 que estava em execução quando ocorreu a interrupção, a rotina correspodente transfere o controle para o nível 2. Se acontecer agora uma outra interrupção caímos no caso anterior. Temos portanto, até dois processos interrompidos: inicialmente o nível 3 ou nível 4 ou gerenciador e posteriormente o nível 2.

A identidade do processo interrompido pode ser guardado

numa variável do tipo processo. Desta maneira sabemos a qual processo devemos devolver o controle posteriormente.

O gerenciador permanece num laço infinito verificando:

(1) Se a fila de recepção não está vazia, transferindo então o controle para o nível 2.

(2) Se existe alguma mensagem a ser transmitida por algum terminal do concentrador, transferindo então o controle para o nível 4.

(3) Se ocorreu algum evento em um dos níveis, (temporização que expirou, DTE tornou-se ocupado, etc), transferindo o controle para este nível.

O nível 2 sendo acionado pelo gerenciador para tratar um quadro da fila de recepção, analisa este quadro e se ele for de supervisão ou de controle, após processá-lo (podendo implicar no envio de outro quadro de supervisão ou de controle para o nó da rede) devolve o controle para o gerenciador. Em caso contrário (o quadro era de informação), o nível 2 envia o conteúdo deste quadro, ou seja, um pacote para o nível 3 e o controle também é passado para o nível 3. Se este pacote era de dado, o nível 3 envia o conteúdo deste pacote, ou seja, uma mensagem para o nível 4 que também recebe o controle do microprocessador. O nível 4 insere esta mensagem na fila associada ao terminal correspondente e passa o controle para o gerenciador. Se o pacote que o nível 3 recebeu

não era de dado, o nível 3 processa este pacote de controle e dois casos são possíveis:

a) Se for necessário enviar outro pacote, este pacote é transmitido e o controle transferido para o nível 2.

b) Se não for necessário enviar outro pacote, o controle é devolvido ao gerenciador.

O nível 4 ao ser acionado pelo gerenciador para tratar uma mensagem de algum terminal, após terminada a execução passa o controle para o nível 3 e este ao processar a mensagem envia um pacote para o nível 2 e passa o controle para este processo e finalmente o nível 2 insere este quadro na fila de transmissão e devolve o controle para o gerenciador. Portanto estamos buscando também o aproveitamento do paralelismo no sentido dos terminais para a linha X-25.

O esquema de mudança de processos desejável a maior parte do tempo é:

Na recepção:

Gerenciador → Nível 2 → Nível 3 → Nível 4 → Gerenciador

Na transmissão:

Gerenciador → Nível 4 → Nível 3 → Nível 2 → Gerenciador

A interface HDLC 8273 ao terminar um envio de quadro para o nó da rede gera uma interrupção no microprocessador e a Rotina de Interrupção de transmissão passa a ser executada. Se a fila de

transmissão não está vazia, a rotina ativa a interface dando o primeiro quadro desta fila e retorna o controle ao processo interrompido anteriormente. Isto é feito para que o hardware da interface pcssa funcionar a plena velocidade.

Uma característica importante do sistema é que a transferência de controle de um processo para outro é feito *ao final do código do processo*. Sabemos que os processos em Módula 2 são implementados como corrotinas, e portanto, toda vez que um processo é chamado, ele começa a ser executado a partir do início pois os processos ficam num laço infinito.

Existem duas exceções:

(1) Quando uma interrupção ocorre, o controle é passado a uma das Rotinas de Interrupção e o controle é retornado posteriormente ao processo interrompido, não ao seu final, mas obviamente na instrução seguinte à última executada antes da interrupção.

(2) O processo gerenciador transfere o controle a vários processos em diversos pontos do seu código. Obviamente, quando o controle é retornado a ele, não é ao seu final, mas sim ao ponto imediatamente após ao procedimento de transferência executado anteriormente.

Outra observação importante é sobre o gerenciador: Toda vez que o controle é retornado ao gerenciador, este deve verificar se algum time-out expirou ou se alguma condição de ocupado se tornou

realidade, para transferir o controle para o nível apropriado. Estas situações são de exceção, e portanto, devem ser tratadas o mais rapidamente possível.

Os níveis tem diferentes atividades para executar, dependendo de qual processo recebeu o controle.

O nível 2 possui quatro atividades:

a) Tratar um quadro da fila de recepção, tendo recebido controle do gerenciador;

b) Transmitir um pacote recebido do nível 3;

c) Tratar de eventos que ocorreram neste nível: temporização que expirou, DTE tornou-se ocupado, etc.;

d) Tratar uma "recepção urgente", tendo recebido o controle da rotina de interrupção de recepção quando o quadro é de supervisão ou controle.

Existe prioridade para a recepção de quadros: O nível 2 após terminar qualquer uma das atividades a), b) e c) descritas acima, se for passar o controle para o gerenciador, deve antes tratar os quadros da fila de recepção até que encontre um quadro de informação. Isto é feito para que a fila de recepção fique vazia, sempre que possível, para que quadros de supervisão e controle recebidos, e que podem influenciar na fila de transmissão, sejam tratados o mais rapidamente possível ("recepção urgente").

As atividades do nível 2 são esquematizadas no procedimento

seguinte:

Utilizaremos uma variável "NÍVEL 2" de um tipo enumerável. De acordo com o seu conteúdo, o nível 2 pode decidir qual atividade dentre essas quatro deverá executar.

```
tipo MODO2  = (GR 2, GT 2, GE 2, GRU 2 );
Var  Nivel2 : MODO2 ;
Procedimento  P_NÍVEL_2;
     :
   início
   laço
     caso nível 2 seja
       GR2: RECEPÇÃO_2;
       GT2: TRANSMISSÃO_2;
       GE2: EVENTO_2;
       GRU2: RECEPÇÃO_URGENTE_2;
     fim; (* caso *)
   fim;    (* laço *)
   fim; P_ NÍVEL_2;
```

O nível 3 possui 5 atividades:

a) Transmitir uma mensagem do nível 4 para o nível 2;

b) Tratar de algum evento que ocorreu neste nível: time-out, DTE ocupado, etc.;

c) Tratar um pacote recebido do nível 2;

d) Abrir um canal lógico à pedido do nível 4;

e) Fechar um canal lógico à pedido do nível 4.

O nível 4 possui três atividades:

a) Recepção de uma mensagem de um dos terminais;

b) Recepção de uma mensagem do nível 3;

c) Tratar de algum evento que ocorreu neste nível.

A implementação do procedimento principal do nível 3 e do nível 4 é equivalente ao do nível 2 utilizando uma variável "NÍVEL 3" e uma variável "NÍVEL 4", que descrevem qual atividade deve ser executada. Os algoritmos correspondentes são análogos aos do nível 2 acima.

4.2. ESTRUTURA DE DADOS - FILAS.

Como existe troca de mensagens entre os processos envolvidos necessitaremos de filas para ordenar as mensagens recebidas. Em quatro casos estas filas são necessárias:

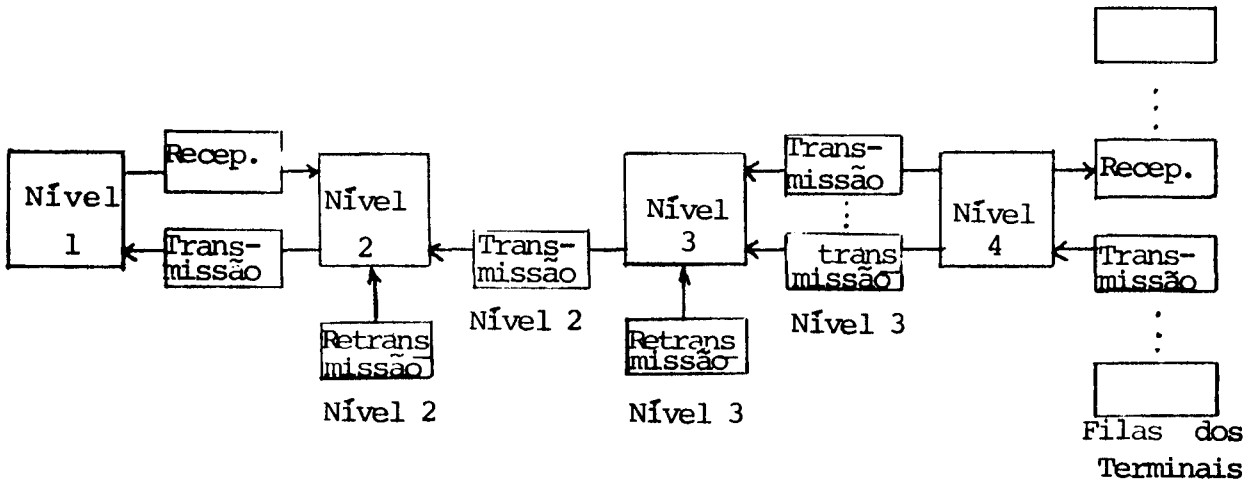


Figura 4.2

1) Filas de recepção e transmissão como interface entre os níveis 1 e 2. A rotina de interrupção de recepção insere quadros recebidos que serão tratados posteriormente pelo nível 2 na fila de recepção. A rotina de interrupção de transmissão retira quadros colocados anteriormente pelo nível 2 da fila de transmissão.

2) Filas de recepção e transmissão associadas a cada terminal do concentrador.

3. Filas de retransmissão para os níveis 2 e 3. Segundo o protocolo X-25, o nível 2 deve ter condições de retransmitir quadros perdidos na linha X-25, pois esta é uma das ferramentas utilizadas para a recuperação de erros. Portanto, o nível 2 ao

inserir na fila de transmissão um quadro ao ser enviado, deve também inserir uma cópia (ou mesmo quadro porém apontado por outro "pointer") na fila de retransmissão. Deve retirar posteriormente desta fila os quadros confirmados. Segundo o mesmo protocolo não existe o conceito de retransmissão de pacotes pelo nível 3. Porém o nível 4 deve ter acesso aos pacotes não confirmados em situações de RESTART e RESET do nível 3. Portanto, o nível 3 ao inserir na fila de transmissão do nível 2 um pacote, deve também inserir uma cópia (ou o mesmo pacote porém apontado por outro "pointer") na sua fila de retransmissão. Deve retirar posteriormente desta fila os pacotes confirmados. O nível 4 em situações de RESTART e RESET do nível 3 retransmitiu os pacotes não confirmados: O nível 4 retira todos os pacotes da fila de retransmissão do nível 3 e os insere na fila de transmissão do nível 3, para que o nível 3 possa tratá-los novamente.

4) Filas de transmissão para os níveis 2 e 3. Da maneira como estamos propondo o sistema, ou seja, ao se receber um quadro de informação o nível 2 imediatamente passa um pacote para o nível 3 e este envia a mensagem para o nível 4 que a insere numa fila associada ao terminal e vice-versa, ou seja, ao receber uma mensagem de um terminal, o nível 4 passa essa mensagem para o nível 3 que então envia um pacote para o nível 2 que insere este quadro na fila de transmissão, pode parecer a princípio que não precisamos de filas intermediárias entre os diversos

níveis.Isto é verdade no sentido do nível 2 para o nível 4. No sentido contrário, do nível 4 para o nível 2, isto não é verdade por causa de apenas um fato: as janelas envolvidas. O nível 4 pode dar uma mensagem para o nível 3 enviar, mas a janela do circuito virtual envolvida pode barrar esta transmissão. É necessário portanto filas que sirvam de interface entre os níveis 3 e 4 no sentido de transmissão, uma para cada circuito virtual. O mesmo ocorre entre os níveis 2 e 3, podendo a janela do nível 2 barrar pacotes enviados pelo nível 3. Portanto, é necessário uma fila de transmissão para o nível 2. Este fato faz com que o Gerenciador , em seu laço infinito, tenha também que verificar se estas filas não estão vazias para transferir o controle para o processo conveniente, para que este possa verificar a "abertura" da janela e transmitir os quadros/pacotes permitidos.

As filas de transmissão e retransmissão dos níveis 2 e 3 não necessitam de regiões críticas para serem manipuladas. A exclusão mútua no tempo já acontece devido às características intrínsecas do sistema:

a) temos apenas um processador
b) para as filas de transmissão apenas um processo insere novas mensagens e apenas um processo retira mensagens;
c) da maneira como o sistema foi projetado, o controle nunca é passado do processo que insere (ou retira) uma mensagem para o segundo processo retirar (inserir) uma

mensagem nesta mesma fila devido à ocorrência de interrupções;

d) para as filas de retransmissão é o mesmo processo que insere e retira mensagens, exceto para a fila de Retransmissão do nível 3 em cada caso de RESET e RESTART. Neste caso particular ocorre a condição c.

Para as filas de recepção e transmissão da interface nível 1/nível 2 e para as filas de recepção e transmissão associadas aos terminais é necessário exclusão mútua. Por exemplo: o nível 2 está retirando um quadro da fila de recepção, quando ocorre uma nova recepção. O controle é passado para a rotina de interrupção de recepção que insere um novo quadro nesta fila. Se a fila tiver dois ou mais quadros, embora os dois processos estejam alterando a fila, a consistência é mantida porque cada um dos dois processos atualiza um apontador diferente (nível 2 → header(H1); nível 1 → fim-da-fila (H2)). Porém se a fila tem apenas um quadro, o nível 2 vai alterar H1 e H2. A exlusão mútua é então requerida. O mesmo fato ocorre entre as filas que servem de interface entre o nível 4 e os terminais.

Com essas regiões críticas são muito pequenas e temos apenas um processador para obter a exclusão mútua basta executar estas regiões críticas de maneira não interrompível.

4.3. CARACTERÍSTICAS DA IMPLEMENTAÇÃO.

Em uma variável EXEC sempre teremos o processo em execução, a menos que estejamos transferindo o controle para outro processo. Neste caso, em EXEC já teremos o novo processo. A variável EXECANT é utilizada pela rotina de interrupção de recepção para guardar a indetificação do processo corrente antes da interrupção.

Utilizaremos módulos não interrompíveis para:

a) transferir controle de um processo para outro;

b) acessar algumas variáveis compartilhadas por mais de um processo.

O caso a é necessário para que possamos atualizar as variáveis EXEC e NÍVEL 2 ou NÍVEL 3 ou NÍVEL 4 (vide seção 4.1) se necessário, antes de transferir o controle para o outro processo.

O caso b é necessário apenas para as filas de recepção e transmissão e filas associadas aos terminais.

As variáveis tipo processo (apontadores para processos) N2, N3, N4, Gerenc estão *sempre* associadas aos processos P_nível 2 , P_nível 3, P_nível 4 e Gerenciador respectivamente.

Portanto, a transferência de um nível para outro será realizado em procedimentos de Módulos não interrompíveis, como por exemplo nesta transferência de controle do gerenciador para o nível 2 para tratar um evento:

```
Módulo  INTG-DESATIV [1] ;
  do sistema importe  transfer;
  importe EXEC, GERENC, N2, NÍVEL_2, GE2;
  exporte  GERENC_N2
     :
  procedimento GERENC N2;
     início
     Exec:= N2;
     nível 2:= GE2;
     transfer (GERENC, N2)
     fim GERENC_N2
       :
  fim INTG_DESATIV;
```

onde [1] ao lado do nome do módulo significa que este será executado de maneira não interrompível.

Trechos de procedimentos que apresentam transferência de controle entre processos podem ser vistos no Apêndice 1.

Em Módula 2 um processo só pode ser um procedimento de nível 0 sem parâmetros. Uma primeira idéia sobre o esquema básico do sistema poderia ser:

```
Módulo  PROTOCOLO X-25;
   Módulo DRIVERS [1];
      procedimento DR_RECEPÇÃO;
      procedimento DR_TRANSMISSÃO;
   Procedimento P_NÍVEL 2;
   Procedimento P_NÍVEL 3;
   Procedimento P_NÍVEL 4;
   Procedimento GERENCIADOR;
```

onde todos os procedimentos seriam processos.

As variáveis do nível 2 seriam locais ao nível 2, assim como para os demais procedimentos (processos). Logicamente este esquema é desejável.

Porém o acesso a uma variável local custa mais do que o acesso a uma variável global em nosso compilador. Portanto o esquema básico foi melhorado para:

```
Módulo PROTOCOLO-X25;
   Módulo DRIVER [1];
      Procedimento DR_RECEPÇÃO;
      Procedimento DR_TRANSMISSÃO;
   Módulo PROCESSO 2;
      Procedimento P_NÍVEL 2;
   Módulo PROCESSO 3;
      Procedimento P_NÍVEL 3;
   Módulo PROCESSO 4;
      Procedimento P_NÍVEL 4;
   Procedimento GERENCIADOR;
```

onde as variáveis particulares do nível 2 seriam locais ao Módulo Processo 2, assim como para as variáveis dos níveis 3 e 4.

Isto resolve o problema pois:

1 As variáveis particulares do nível 2 continuariam sendo privativas do nível 2 (a menos que as exportássemos do Módulo processo-2). O mesmo ocorre com as variáveis dos níveis 3 e 4.

2 Como a existência do Módulo Processo 2 não introduz aumento no nível de encaixamento dos procedimentos, as variáveis locais a este Módulo seriam globais no programa.

O esquema geral, mais detalhado, pode ser visto no Apêndice 1.

O esquema geral do nível 1 é o seguinte:

```
Módulo DRIVER [1];

   Procedimento DR_RECEPÇÃO
         .
         .
         .

   Procedimento DR_TRANSMISSÃO;
         .
         .
         .

   início

      INICIAL_RECEPÇÃO_INTERFACE;

      INICIAL_TRANSMISSÃO_INTERFACE;

   fim driver;
```

onde o corpo do módulo é executado uma única vez no início da execução do protocolo para inicializar a interface HDL8273 nos dois sentidos de tráfego de informação.

Os drivers utilizam a instrução IOTRANSFER para transferir o controle para o processo apropriado após a sua execução.

O algoritmo, um pouco mais detalhado, para o driver de recepção é o seguinte:

Se o processo em execução não é o nível 2 e o quadro é de controle ou de supervisão e a fila está vazia

então o controle é passado para o nível 2 para tratar "recepção urgente" e o processo interrompido é guardado na variável EXECANT (informação esta que vai ser utilizada posteriormente pelo nível 2 para devolver o controle a este processo)

Senão inserimos este quadro recebido na fila de recepção do nível 1. (Esta fila será verificada que não está vazia pelo gerenciador ou pelo próprio nível 2 em outra oportunidade, e então será feito o tratamento deste quadro). O procedimento é o seguinte:

```
procedimento DR_RECEPÇÃO;
    início
    laço
      IOTRANSFER (p1, p0, 1);
      {forneço a interface o endereço  END2}
      se (EXEC <> N2) e (H1 RECN1) e (8 em ALOC MEM [END1 + 7]
         então  início
                EXECANT ← EXEC;
                EXEC ← N2;
                p0 ← N2, nível 2:= CRU 2;
                END_INTERFACE12  ← END1;
         senão  INSERE-FILA (H1_REC N1, H2_REC N1, END1);
      END1 ← END2;
      ALOCA_BLOCO (END2, CTE);
    fim (* laço *)
    fim DR_RECEPÇÃO;
```

O algoritmo mais detalhado para o driver de transmissão é o seguinte:

Se a fila de transmissão não está vazia.

então a rotina entrega à interface o primeiro quadro desta fila, devolvendo o controle ao processo interrompido.

senão a rotina somente devolve o controle ao processo interrompido e a interface aguarda novos quadros para serem transmitidos.

A seguir apresentamos este procedimento:

```
Procedimento DR_TRANSMISSÃO;
    início
    laço
      IOTRANSFER (p1, p2, 2);
      DESALOCA_BLOCO (END1);
      se (H1_TRANSM1 <> 0)
          então início
                primeira-fila (H1_TRANS1, H2_TRANS1, END1);
                {forneço à interface o endereço END1};
                fim
    fim (* laço *)
    fim DR_Transmissão;
```

4.4. ALOCAÇÃO DINÂMICA DE MEMÓRIA.

O comprimento das mensagens recebidas dos terminais pelo concentrador é variável. São estas mensagens que irão constituir os pacotes do protocolo. Portanto, é uma política razoável considerar o tamanho do pacote igual ao tamanho da mensagem digitada no terminal até um CR. Se for maior do que 128 bytes devemos quebrar esta mensagem em mais de um pacote.

Se o serviço requerido pelo terminal é o de transferência de um arquivo, nesta política, o tamanho do campo de informação

é o comprimento máximo que o protocolo X-25 permite, ou seja, 128 bytes.

Com esta consideração passamos a ter pacotes de informação de tamanho variável. Possuímos também pacotes e quadros de controle e supervisão que tem tamanho pequeno em relação aos pacotes de informação. Estes dois fatos nos levam a introduzir um mecanismo de alocação de memória no sistema para a alocação e liberação de mensagens, pacotes e quadros.

O esquema de alocação dinâmica proposto é o seguinte: as mensagens são alocadas num vetor de bytes. Quando uma mensagem oriunda de um terminal chega no concentrador, este aloca dinamicamente um bloco com cinco bytes a mais do que o tamanho da mensagem. Estes cinco bytes serão utilizados para conterem os cabeçalhos do nível 2 (2 bytes) e nível 3 (3 bytes). O nível 4 passa para o nível 3 apenas o endereço desta mensagem. O nível 3 coloca o seu cabeçalho nos bytes 3, 4 e 5 do bloco e passa o endereço deste pacote para o nível 2. Este coloca seu cabeçalho nos bytes 1 e 2 do bloco. No sentido de recepção, procedimento inverso é realizado.

Desta maneira, a informação para percorrer todos os níveis é alocada uma única vez.

Uma observação importante é a de que um quadro ou pacote pode pertencer a mais de uma fila ao mesmo tempo. Como por exemplo, um quadro pode estar na fila de transmissão e na fila de

retransmissão do nível 2. Não queremos ter o mesmo quadro armazenado duas vezes na memória.

Um método para resolver este problema é o seguinte: o primeiro elemento de cada fila de recepção e transmissão é apontado por um"header". As filas de retransmissão, que contém as cópias, seriam tratadas externamente ao vetor de alocação, pois elas possuem tamanho limitado devido às janelas envolvidas.

Um esquema de alocação dinâmica para ser eficiente deve ter compactação de memória livre, ou seja, ao se liberar um espaço de memória deve-se juntar este espaço livre aos espaços livres adjacentes se existirem. Os espaços livres são ligados através de apontadores, formando uma lista geralmente duplamente ligada. Os espaços reservados serão ligados através de apontadores formando listas que serão as filas necessárias para esta implementação e que estão descritas na seção 4.2. Os blocos devem ter um ou dois "tags" que servirão para distinguir os blocos livres dos ocupados. Knuth em [10] propõe esquemas e algoritmos que utilizam dois "tags" por bloco (um no início e outro no fim) e em [5] temos blocos com um "tag" e uma avaliação de que este esquema também é eficiente.

Adotamos apenas um "tag" por bloco e a lista de blocos livres é duplamente ligada. No Apêndice 2 descrevemos os campos dos blocos utilizados e propomos uma otimização na alocação dinâmica de memória.

Para a implementação do protocolo X-25 num concentrador pode

parecer muita sofisticação (e talvez até perda de eficiência) ter um esquema de alocação dinâmica, ao invés de ter blocos de tamanho fixo (no caso o tamanho máximo de 133 bytes), em função do barateamento das memórias. Porém, estamos adotando mensagens de tamanho muito variado (mensagens que se originam nos terminais) e existem também pacotes e quadros de supervisão, os quais são de tamanho pequeno. Talvez os resultados não diriam muito para um concentrador pequeno, porém, para um nó da rede, o qual possui duas ou mais linhas X-25 que utilizam altas velocidades de transmissão, algum mecanismo de alocação dinâmica deve ser usado (não podemos armazenar as mensagens que transitam em memória secundária, como por exemplo em discos, pois sera muito demorado).

4.5. TEMPORIZAÇÕES.

A necessidade de temporização (time-out) para implementar o protocolo X-25 e exemplos simples foram analisados nas seções 1.2.1 e 1.2.2. Nesta seção comentamos algumas características das temporizações dos diversos níveis.

No nível 2 estamos trabalhando com um time-out. Possuímos dois fluxos para controlar: o de transmissão e o de recepção. O primeiro é óbvio pois quando enviamos pedidos de conexão, quadros de informação, pedidos de desconexão, etc, precisamos do time-out para verificar se a confirmação correspondente chega em um tempo razoável, ou se é necessário a retransmissão de algum quadro. O

segundo é necessário apenas para controlar a retransmissão de REJ (controla o fluxo recebido). Este controle requer um time-out,que pode "correr" simultaneamente com o outro time-out de controle de transmissão de quadro de informação. Sincronizamos estes dois controles para ter apenas um time-out.

No nível 3 temos um time-out para cada canal lógico, pois, cada canal lógico controla um tráfego de pacotes diferentes. Estes time-out controla o envio dos pacotes Restart Request, Call Request, Reset Request e Clear Request. Se o time-out expirar, ocorre a retransmissão do pacote no canal lógico correspondente. Um só time-out pode controlar a transmissão de todos esses quatro pacotes em determinado canal lógico, porque em um determinado instante, o estado desse canal lógico é de espera de confirmação de apenas um desses pacotes. Transmissões de pacotes de dados não necessitam de time-out (vide seção 1.2.2). Implementamos o nível 4 sem utilizar time-out. O argumento para justificar este fato pode ser visto na seção 1.2.3.3.

4.6. DESCRIÇÃO DO SISTEMA.

O protótipo desta tese é baseado no microprocessador 8088 da Intel, e possui 32K de memória RAM estática além de memória EPROM onde está armazenado o programa monitor.

A carga dos programas no microprocessador é feita via DEC-10 do Centro de Computação. Os programas escritos em Módula 2 são compilados e montados no DEC-10 através de um compilador cruzado para Módula 2 e um montador cruzado que gera código para o 8088. Um programa carregador transmite o código objeto do DEC-10 para a memória do micro.

O protótipo possui duas interfaces HDLC 8273 da Intel para transmissão e recepção de quadros e um controlador de interrupções 8259 da Intel. A utilização dessas interfaces e controladores é comentada na seção seguinte.

O sistema possui ainda três USART's. Uma está conectada a um terminal (console), a outra a uma impressora e a outra não está sendo utilizada. O console também funciona como um "terminal" do DEC-10. Do console podemos carregar na memória do micro um programa existente no DEC-10, mandar executá-lo e os resultados podem ser escritos na impressora ou no próprio console.

4.7. INTERFACE HDLC 8273 INTEL.

Como interface entre o concentrador e a linha X-25 foi utilizado o controlador programável de protocolo HDLC/SDLC 8273 da Intel.

O controlador opera em modo full duplex e tem as facilidades de cálculo e verificação automática de FCS, de inserção e

retirada automática do bit zero para resolver o problema da transparência, de compatibilidade de tecnologia TTL com outros controladores SDLC e de estrutura de comando a nível de quadros ou seja, o controlador 8273 interrompe a CPU apenas ao final da recepção de um quadro ou ao final de transmissão de um quadro. Não utilizamos esta última característica devido ao fato de não termos DMA (direct memory access) no protótipo da implementação.

O lay-out usado está mostrado na figura 3.11 e o modo de comunicação é assíncrono sem a utilização de modens.

O controlador 8259 é o controlador de interrupção do sistema. São utilizadas duas entradas para interrupções do 8273: uma para recepção e outra para transmissão.

Nesta configuração de hardware não estamos utilizando DMA. Portanto o controlador 8273 gera uma interrupção a cada byte recebido e a cada byte transmitido (dois sinais diferentes).

As interrupções de final de byte transmitido e de final de quadro transmitido são diferenciados por um bit no registrador de estado do controlador 8273. Portanto, quando a CPU aceita um pedido de interrupção de transmissão, deve ler o registrador de estado para identificar se é um término de quadro ou simplesmente o término de um envio de um byte do quadro. Neste caso coloca o byte no buffer de montagem do quadro e devolve o controle ao processo interrompido. Procedimento análogo deve ser feito para diferenciarmos o término da recepção de um quadro e o término da recepção de um byte de um quadro.

Normalmente T×D e R×D são pinos ligados ao Mcdem. Devido ao fato de não possuirmos dois microprocessadores para comunica — rem entre si, os pinos T×D e R×D são ligados a um conector. Temos dois controladores 8273, e os testes de funcionamento do sistema são feitos "ligando" a saída de um controlador com a entrada do outro ou com a entrada do mesmo. Por software podemos alterar um pouco a entrada, para simular o meio físico de comunicação que é sujeito a erro.

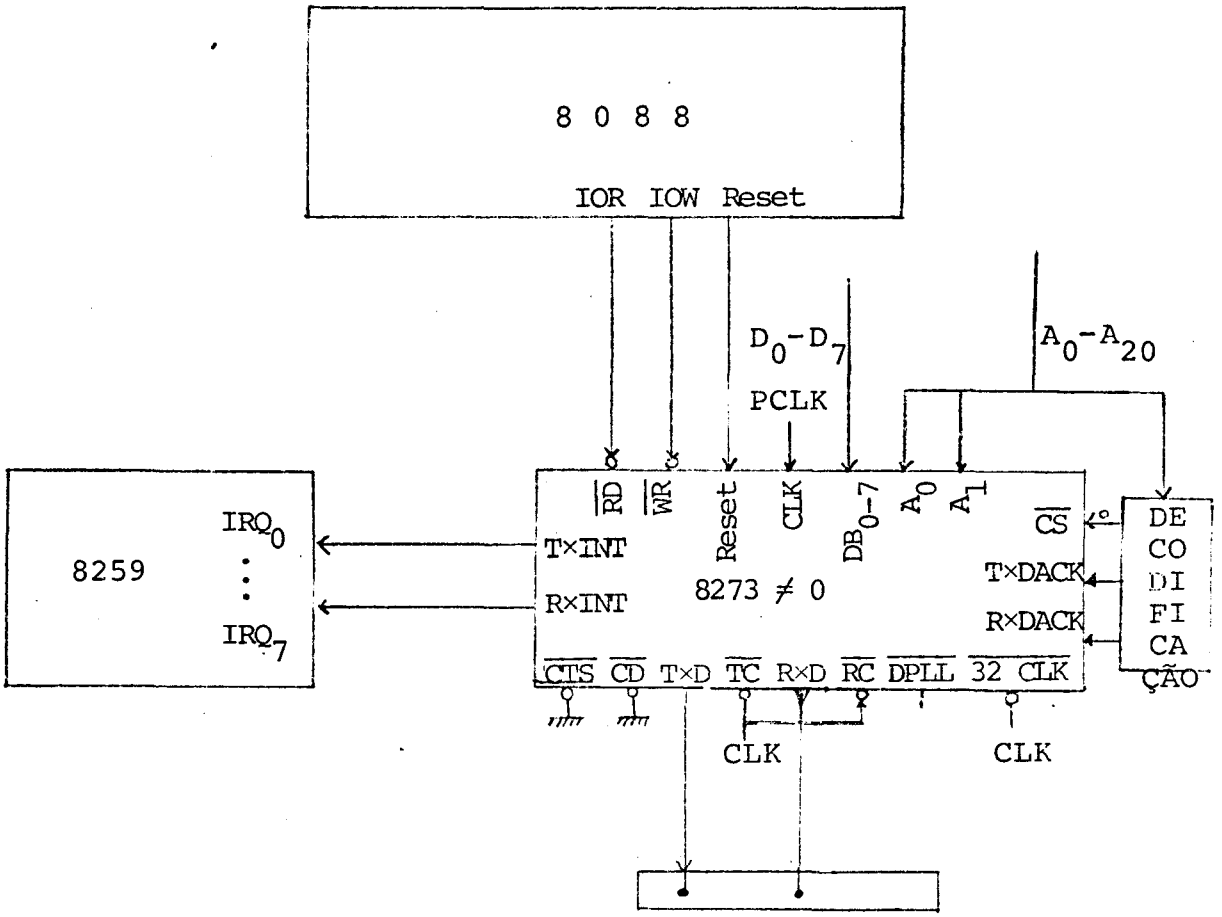


Figura 3.11

CAPÍTULO V

TESTE DO SISTEMA

5.1. TESTES.

Inicialmente o programa foi testado em módulos separados: nível 2, nível 3, nível 4 e estrutura de dados (alocação dinâmica de memória e filas) (1ª etapa da depuração).

A 2ª etapa consistiu no teste integrado do sistema da seguinte forma:

O gerenciador permanece num laço infinito verificando se algum nível tem alguma atividade a realizar: filas não vazias com mensagens a serem tratadas, temporizações que expiraram, situações que tornaram o concentrador ocupado que devem ser tratadas pelos diversos níveis (vide seção 4.1). A essas diversas verificações foi acrescentada uma leitura (do console) de uma sequência de caracteres, a qual é utilizada para o teste do programa.

Esta sequência de caracteres pode ser de seis tipos, em função do primeiro caracter:

1º TIPO: Simula a recepção de um quadro pelo nível 2. É lido o conteúdo do quadro (incluindo o byte de endereço e controle), e este é inserido na fila de recepçao do nível 2.

2º TIPO: Simula a recepção de uma mensagem pelo nível 4. É lido o número do terminal que enviou a mensagem e o conteúdo da mesma. A mensagem é inserida na fila de transmissão do nível 4 associada a este terminal.

3º TIPO: Simula um time-out expirado. É lido qual time-out expirou: nível 2, nível 3 - CL = 1, nível 3 - CL = 2, nível 3 - CL = 3 ou nível 3 - CL = 4. As variáveis de time-out do programa são alteradas convenientemente para mostrar a ocorrência deste evento.

4º TIPO: Simula a transmissão de um quadro pelo nível 2. O primeiro quadro da fila de transmissão do nível 2 é retirado.

5º TIPO: Simula a retirada de uma mensagem por um dos terminais do concentrador. É lido o número do terminal que retira a mensagem e então é retirado a primeira mensagem da fila de recepção do nível 4 associada a este terminal.

Após a execução do procedimento de qualquer um desses tipos, mais um ciclo do laço infinito do gerenciador é executado. O 6º tipo não simula nada, indicando apenas que mais um ciclo do laço deve ser iniciado.

Com este recurso simulamos a chegada assíncrona de quadros na linha X-25 e de mensagens dos diversos terminais. Verificamos a recuperação do sistema quando um time-out ocorre, ou quando um

um quadro(ou pacote) incorreto é recebido. Verificamos também a "passagem" de uma mensagem de dados de um terminal para o nível 4, e deste para o nível 3, e deste para o nível 2 e então ser transmitida, ou um quadro (contendo uma mensagem para um terminal) "atravessar" todos os níveis (2,3 e 4) a ser entregue ao terminal.

OBSERVAÇÃO: As mensagens enviadas são escritas numa impressora. É impresso também por qual fila esta mensagem foi remetida: fila de transmissão de algum terminal ou fila de transmissão associada ao nível 1. Nos testes preliminares também foram impressos as mensagens enviadas de qualquer nível para outro. Mudanças no vetor de alocação foram também registradas.

A terceira etapa inclui a inserção das interfaces HDLC 8273 no sistema.

Inicialmente foram realizados testes para verificar o funcionamento das interfaces em "busy wait" e com interrupções.

Foram realizados também testes para a implementação da temporização (time-out).

Se tivéssemos dois microcomputadores para realizar os testes ligaríamos os dois, que então se comunicariam de acordo com o protocolo X-25. Devido ao fato de termos apenas um microcomputador disponível, ligamos a saída da linha X-25 à entrada da segunda interface e a entrada da linha X-25 à saída da segunda interface. Esta segunda interface 8273 se comunica com um programa definido apenas para propósito de teste que responde aos quadros que

chegam e pede conexão quando quer transmitir informação. O esquedo teste é o seguinte:

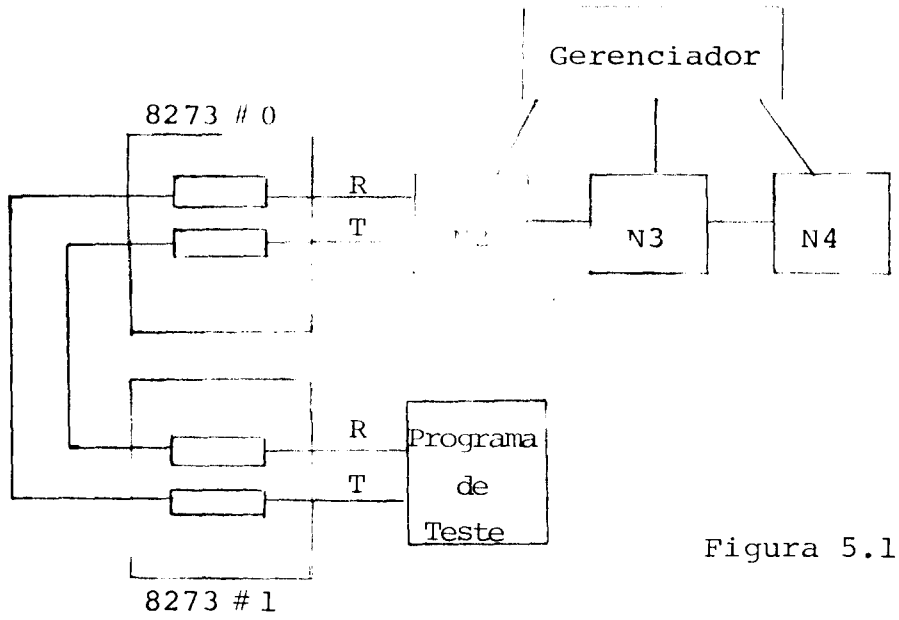
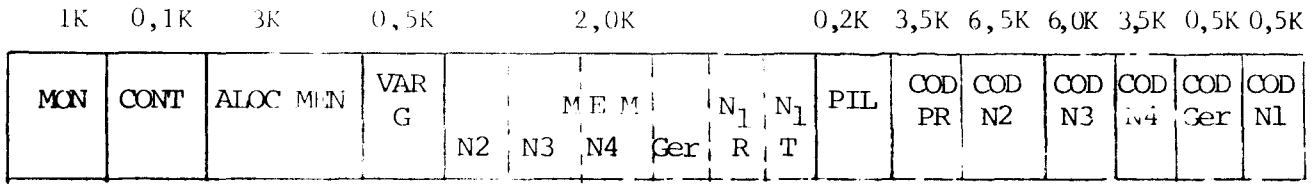


Figura 5.1

5.2. MAPA DE MEMÓRIA.

Em Módula 2 cada processo, inclusive o processo principal (Módulo principal), está associado a uma área de memória que contém o contexto do processo, as variáveis locais ao processo e a pilha de trabalho.

O mapa de memória do sistema está mostrado na figura abaixo:



Código: 20K

Total: 26,8K

Figura 5.2

onde:

MON - área de trabalho do monitor;

CONT - área de contexto do processo correspondente ao módulo principal;

ALOC_MEM - área reservada para a alocação dinâmica: vetor de bytes;

VAR.G - área que contém as variáveis globais do sistema;

MEM - área de trabalho dos processos: nível 2, nível 3, nível 4, Gerenciador e Rotinas de interrupção. Para cada um dos processos, contém o contexto, as variáveis locais ao processo e a pilha de trabalho;

PIL - área que contém a pilha de trabalho do processo correspondente ao módulo principal;

COD-PR - área de código dos corpos dos módulos (inclusive o módulo principal) e procedimentos de manipulação de filas e de alocação e liberação de blocos e inicializações;

COD_N2 - área de código do nível 2;

COD_N3 - área de código do nível 3;

COD_N4 - área de código do nível 4;

COD_GER - área de código do Gerenciador;

COD_N1 - área de código do nível 1;

OBS: Os procedimentos de alocação e liberação de blocos de memória podem ser melhorados se escritos em Assembler (código menor). Escritos em Módula 2, estes procedimentos possuem muitas chamadas de procedimentos global com parametros. Isto custa caro em Módula 2, pois em cada uma destas chamadas, o contexto da CPU é guardado e recuperado posteriormente. Escrito em Assembler, o contexto para guardar e recuperar é bem menor (devido às características desta aplicação), além de que as variáveis passadas como parâmetros tem posições na memória bem definidas (embora difiram em tipo com os parâmetros formais) e poderiam ser acessadas facilmente em Assembler.

CAPÍTULO VI

CONCLUSÕES E SUGESTÕES

O sistema descrito é baseado em troca de mensagens, pois existe movimento explícito de dados entre os processos envolvidos. Porém ele possui também uma característica importante de monitores que é o de compartilhamento de memória.

Se compararmos este sistema com o modelo teórico baseado em troca de mensagens, proposto em [12] vemos que ele possui as seguintes características particulares:

1) Um processo nunca fica bloqueado à espera de uma resposta de um outro processo, ao qual enviou uma mensagem anteriormente. Isto ocorre porque como foi definido o sistema não existe o conceito de mensagem como resposta.

2) O recipiente (nível 2, nível 3 ou nível 4) sabe explicitamente quem foi o remetente da mensagem, verificando qual processo transferiu o controle para o mesmo em variáveis apropriadas.

3) Os processos trabalham com uma só mensagem de cada vez.

4) A troca de mensagens entre processos é feita manipulando se uma estrutura de dados comum, através de apontadores.

5) A integridade, ou seja, a consistência das variáveis compartilhadas deve ser garantida. Como as regiões críticas nos diversos processos são de pequena duração e como temos apenas um processador, esta consistência pode ser obtida apenas inibindo as interrupções na entrada da região crítica e desinibindo as interrupções na saída da região crítica.

6) O controle de entrada/saída é feito associando-se um processo com a interface, e uma interrupção pode gerar uma mensagem para algum outro processo do sistema.

7) Um processo ao enviar uma mensagem para outro processo transfere também o controle para este processo. Esta é uma maneira simples de escalonamento de processos, que não é o comumente usado nos sistemas baseados em troca de mensagens. Desta maneira procuramos aproveitar o paralelismo existente no protocolo.

SUGESTÕES PARA MELHORAMENTOS POSTERIORES.

Para a continuação do estudo de desempenho de microprocessadores para implementar o protocolo X-25 sugerimos a análise dos itens seguintes:

a) Introdução de DMA no sistema. Com este hardware adicional a CPU seria interrompida somente ao final da transmissão e da recepção de quadros, e não mais byte a byte.

b) Estudo da utilização de mais de um microprocessador. Devido ao fato dos níveis poderem ser executados paralelamente, poderíamos ter três CPU's, uma para cada nível (2, 3 e 4) com memória compartilhada. Neste caso não teríamos o Gerenciador. Quão vantajoso seria este sistema é um estudo interessante a se fazer.

O nível 2 é o mais sobrecarregado do sistema, pois todos os pacotes do nível 3 e as mensagens do nível 4 transitam por esse nível. Se tivéssemos duas CPU's, um esquema que deveria ser analisado é o seguinte: uma CPU seria dedicada ao nível 2 e às rotinas de interrupção do nível 1 e a outra CPU seria dedicada ao níveis 3 e 4, tendo também um Gerenciador. A memória seria compartilhada.

c) Implementação de um concentrador de terminais de verdade. Para tanto, ao sistema escrito teria que ser ligado os diversos terminais e implementado um pequeno gerenciador de tratamento de interrupções dos terminais. O protocolo para comunicação entre o

nível 4 e os terminais é descrito na seção 1.2.3.3.

d) Implementação de um nó de rede. Este caso é bem mais complexo do que a implementação do concentrador, pois um nó de rede tipicamente possui várias linhas X-25 de alta velocidade que o liga a outros nós da rede e a DTE's. Filas para atender aos diversos fluxos de informação devem ser implementadas. Esquemas de roteamento devem ser analisados para o envio eficiente de mensagens para a rede em função da distância a ser percorrida pela mensagem e do nível de ocupação das linhas, inclusive se algum nó ou linha está fora da rede temporariamente.

BIBLIOGRAFIA

[1] BOCHMANN, Gregor V.

Finite State Description of communicati n protocols

Computer Networks ^ (19?8), 361-372.

[2] BOCHMANN, Gregor V.

Architecture of distributed computer systems

Lecture Notes in Computer Science Nº 77, 192-238, Springer Verlag, 1979.

[3] BOCHMANN, Gregor V. e JOACHIM, Tankoano

Development and Structure of an X-25 Implementation

IEE Transactions on software Engineering, vol. SE-5, Nº 5, September 1979.

[4] DAVIES, BARBER e SOLOMONIDES

Computer Networks and their protocols.

[5] GUIMARÃES, Célio C.

Princípios de sistemas operacionais

Ed. Campus - 1981.

[6] GUIMARÃES, Célio C.
Two efficient dynamic storage allocation algorithm for the first-fit one-sided boundary tag method, Relatório Interno Coppe, Universidade Federal do Rio de Janeiro, 1974.

[7] HOPPE, Jiri.
A simple nucleus written in Modula-2 - A case Study
Software Practice and Experience, Sept. 1980, 697-706.

[8] JOACHIM, Tankoano.
Implatation du protocole standard X-25 a partir d'un modele de formalisation et de mecanismes abstraits de programmation-Document de travail # 103
Tese de doutorado - Universite de Montreal, December 1977.

[9] JORRAND, Philippe
Specification and analysis of Communication Protocols
Research Report - IBM - RJ2853 - 7/2/80.

[10] KNUTH, Donald E.
The art of computer programming
Vol. 1 - Fundamental Algorithms - 1973
Seção 2.5 - 435-455
Addison-Wesley Publishing Company.

[11] KOWALTOWSKI, Tomasz.
Tradução do relatório sobre Modula 2
IMECC-UNICAMP, 1983.

[12] LAUER e NEEDHAM
On the duality of operating systems structure
System Review 13(2) págs. 3-19, 1979.

[13] MADEIRA, Edmundo R.M.
Implementação do protocolo X-25 num concentrador de comunicações baseado no 8088
Anais do IIIº Congresso da Sociedade Brasileira de Computação
págs. 523-527, Julho 1983 - Campinas.

[14] MADEIRA, Edmundo R.M. e GUIMARÃES, Célio C.
Implementação do protocolo X-25 num concentrador de comunicações baseado no 8088
Anais do 2º Simpósio sobre desenvolvimento de software básico para micros
págs. 137-140
Dezembro 1982 - São Paulo.

[15] MENASCÉ, Daniel e SCHWABE, Daniel
Redes de Computadores: aspectos técnicos e operacionais
3ª Escola de Computação, 1982.

[16] STANTON, Michael.
Dualidade e a construção de programas concorrentes
Pontifícia Universidade Católica do Rio de Janeiro, 1981.

[17] TANEMBAUN, Andrew
Computer Networks
Pr. Hall, 1981.

[18] WIRTH, Niklaus
Modula 2
Relatório nº 36, Institut fur Informatic, 1980.

[19] ZAFIROPULO, P., WEST, C., RUDIN, H. e COWAN, D.D.
Towards analysing and Synthesising Protocols
Research Report - IBM - RZ 963, 7/9/79.

[20] Recomendação X-25
CCITT - 1980.

[21] Especificação REXPAC - versão 01
CPqD - Telebrás, 19[illegible].

[22] 8273 - Programmable HDLC/SDLC Protocol Controller
Application - Intel.

[23] ZIMMERMANN, Hubert
OSI Reference Model - The ISO Model of architecture for open systems interconnection
IEEE Transactions on Communications, Vol. Nº 4, Com - 28 , April 1980.

[24] LEE, R.
Ciclyc Code Redundancy
Digital Design - Jul. 81, pg. 77-85.

[25] ZAFIROPULO, P., WEST, C., RUDIN, H. e CORVAN, D.D.
Towards analyzing and synthesizing protocols
Research Report - IBM - Com. RZ963 (#33588), 1979.

[26] BRINCH HANSEN, P.
Structured multiprogramming
Comm. ACM, 15, 7, pp. 574-578, julho 1972.

[27] BRINCH HANSEN, P.
Operating System Principles
Prentice-Hall, 1973.

[28] DIJKSTRA; E.W.
Cooperating sequential process, THE, The Neterlands, 1965.

[29] DIJKSTRA, E.W.
Hierarchical ordering of sequential processes
Acta Inf. 1,2, pp. 115-138, 1971.

[30] GRAY, J.P.; ROSE, D.B.; SCHULTZ, G.D. e WEST, C.H.
Executable Description and Validation of SNA
IEEE Transaction on Communication on Computer Network Architectures and Protocols, 1980.

[31] HARLAND, D.M.
On Facilities for Interprocess Communication
Information Processing Letters, Vol.12, Nº 5, Out. 1981, págs. 221-226.

[32] HOARE, C.A.R.
Towards a theory of parallel programming
Operating System Techniques. Academic Press, N. Y. 1972.

[33] HOARE, C.A.R.
Communicating Sequential Processes
C.A.C.M. - Vol. 21, nº 8, Agosto 78, pp. 666-677.

[34] IBM Europa.
Technical Improvements to CCITT Recommendation X-25
Group IV - Outubro 1978.

[35] ICHBIAH, Jean D.

Preliminary ADA Reference Manual

Sigplan Notices - vol. 14, nº 6, junho 1979.

[36] ISO,

High level data link control-elements of procedure

IS4335, 1977.

[37] MERLIN, P.M.

Specification and Validation of protocols

E.E. Publication Nº 343, Faculty of Electrical Engineering

Technion, Israel, Jan. 1979.

[38] WEGNER, Peter.

Programming with ADA: An introduction by means of graduated examples

Sigplan Notices - Vol. 14, nº 12, Dezembro 79.

[39] WEST, C.H.; ZAFIROPULO, P.

Automated Validadtion of a Communication Protocol: the CCITT X-21 Recomendation

IBM Journal of Research and Development, Vol. 22, nº 1, Janeiro 1978.

[40] WIRTH, N.

Modula: a language for modular programming

Software Practice and Experience, 7, 1, pp. 3-35, Jan. 1977.

Uma explanação sobre redes e a descrição de diversos protocolos podem ser vistos em [17]. As sete camadas que compõe a arquitetura da ISO podem ser vistas em [23]. [36] descreve o protocolo HDLC da ISO. O protocolo X-25 é definido em [20]. Estes assuntos também podem ser vistos em [15]. [8] apresenta uma implementação na Universidade de Montreal do protocolo X-25 a partir de um modelo de formalização. Algumas implementações ou propostas são descritas em [3] e [16]. [4] é um texto que comenta os procedimentos do HDLC - High-level Data Link Control e a interface X-25. Tem um capítulo que trata de terminais inteligentes.

A linguagem Modula 2 é discutida em [18] e [11] é uma tradução. [7] apresenta um "kernel" escrito em Módula 2. Em [5], que introduz conceitos de sistemas operacionais, um capítulo trata especificamente de processos paralelos e seus mecanismos de comunicação e sincronização. [12] descreve modelos de troca de mensagens e monitores e mostra a dualidade entre eles. Os conceitos de semáforos, eventos, condições e monitores podem ser vistos em [28] e [29], [26], [40] e [32]; o conceito de troca de mensagens em [27] e [31] e o conjunto de Randez-Vous em [33] (CSP do Hoare). A linguagem ADA é definida e comentada em [35] e [38].

[19] descreve uma análise de correção de protocolos. Alguns artigos sobre especificação e análise de protocolos de comunicação são [9], [1] e [2]. [39], [34], [30] e [37] apresentam especificações e validações de protocolo. [25] descreve um método automático de análise e síntese de protocolos. [21] é a especificação da Rede Nacional de Pacotes da Telebrás. Os problemas envolvendo alocação dinâmica são analisados em [10] e [6]. O modo de operação da interface IEEE 802.3, sua pinagem, comandos e aplicações podem ser vistos em [7]. O cálculo de CRC por software utilizando tabelas é analisado em [24]. [13] e [14] são resumos desta tese.

APÊNDICE 1

ESQUEMA GERAL:

```
Modulo PROTOCOLO X-25;
    procedimento LIBERA_BLOCO;
    procedimento ALOCA_BLOCO;
    procedimento INSERE_FILA;
    procedimento PRIMEIRO_FILA;
    Modulo DRIVER[1];
        procedimento DR_RECEPÇÃO;
        procedimento DR_TRANSMISSÃO;
    Modulo PROCESSO 2;
        Modulo INT2_DESATIVADA[1];
        procedimento P_NÍVEL2;
            procedimento CONTROLE_SUPERVISÃO
            procedimento RECEPÇÃO2;
            procedimento RECEPÇÃO_URGENTE;
            procedimento TRANSMISSÃO2;
            procedimento [illegible]2;
    Modulo PROCESSO  ;
        Modulo INT3_DESATIVADA[1];
        procedimento P_NÍVEL3
            procedimento RECEPÇÃO3;
```

```
            procedimento TRANSMISSÃO3;
            procedimento EVENTO3;
            procedimento ABRE_CANAL_LOGICO;
            procedimento FECHA_CANAL_LOGICO;
        Modulo PROCESSO4;
            Modulo INT4_DESATIVADA|1|;
            procedimento P_NÍVEL4;
                procedimento RECEPÇÃO4;
                procedimento TRANSMISSÃO4;
                procedimento EVENTO4;
            Procedimento Gerenciador;
                Modulo INT GERENC_DESATIVADA|1|;
                Procedimento TIME-OUT;


Modulo PROTOCOLO-X25;
    do sistema importe byte, transfer, process, newprocess, adr,
                       address,  short_address;
    constante Memory_Size = 2000;
              Ger_Size    =  320;
              N2 Size     =  320;
              N3 Size     =  320;
              N4 Size     =  320;
              N5 Size     =  320;
              Transm Size =  320;
```

```
variável Ender: address;
    Rec_add, Transm_add, Ger_add, N2_add, N3_add, N4_add:
                short address;
                Mem: array[1.. memory_Size] de byte;
                P0, Gerenc, N2, N3, N4, N1R, N1T: process;
  :
início
ender:= adr (Mem);
Ger_add:= Ender.base+(Ender.displacement+15) div 16;
Rec_add:= Ger_add + (Ger_Size div 16);
Transm_add:= Rec_add + (Rec_Size div 16);
N2_add:= Transm_add + (Transm_Size div 16);
N3_add:= N2_add + (N2_Size div 16);
N4_add:= N3_add + (N3_Size div 16);
Newprocess(Dr_Recepção, Rec_add, Rec_Size, N1R);
Newprocess (Dr_Transmissão, Transm_add, Transm_Size, N1T);
Newprocess (Gerenciador, Ger_add, Ger_Size, Gerenc);
Newprocess (P_nível 2, N2_add, N2_Size, N2);
Newprocess (P_nível 3, N3_add, N3_Size, N3);
Newprocess (P_nível 4, N4_add, N4_Size, N4);
Transfer (P0, Gerenc)
fim Protocolo X_25;
```

```
tipo  modos 2 = (GR2, GT2, GE2, GRU2);
      modos 4 = (GR4, GT4, GE4);
      modos 3 = (GR3, GT3, GE3, GACN3, GFCN3);
variável nível 2: modos 2;
         nível 3: modos 3;
         nível 4: modos 4;
         P0, N2, N3, N4, Gerenc, Exec, Execant: process;

Módulo  INT3_DESATIVADA[1];
     do sistema importe  transfer;
     importe Exec, Gerenc, N2, N3, N4, nível 2, nivel 4, GT2, GR4;
     exporte N3_Gerenc, N3_N2, N3_N4;
        procedimento N3_Gerenc;
           início
           Exec:= Gerenc;
           transfer (N3, Gerenc)
           fim  N3_Gerenc;
        procedimento N3_N2;
           início
           Exec:= N2;
           nível 2:= GT2;
           transfer (N3, N2)
           fim N3_N2;
```

```
        procedimento N3_N4;
            início
            Exec:= N4;
            nível 4:= GR4;
            transfer (N3, N4)
            fim N3_N4;
fim INT3_DESATIVADA;

Procedimento  EVENTO 2;
     início
           :
     se Timeout N2 = 1 então TRATA_TIMEOUT_N2;
     senão se ocupado então TRATA_OCUPADO_N2;
     senão se abrir_ligação então WAIT_CONEXÃO;
     senão se fechar_ligação  então WAIT_DESCONEXÃO;
     N2_Gerenc;
     fim EVENTO 2;
```

onde N2_Gerenc é o procedimento do módulo não interrompível que transfere controle de N2 para o Gerenc.

```
Procedimento  CONTROLE_OU_SUPERVISÃO;
      :
Procedimento  RECEPÇÃO_2;
      início
        :
      INT2_PRIMEIRO_FILA (aux 12);
      DECODIFICA;
      Se (quadro = I) então TRATA_I
                      senão CONTROLE_OU_SUPERVISÃO;
      Se    B    então LIBERA_BLOCO (aux 12);
        .   :
      N2 _N3 Gerenc
      fim RECEPÇÃO_2;
```

onde N2_N3 Gerenc é o procedimento do módulo não interrompível que transfere o controle do N2 para o N3 ou para o Gerenc.

```
Procedimento RECEPÇÃO_URGENTE;
      início
        :
      DECODIFICA;
      CONTROLE_OU_SUPERVISÃO;
      LIBERA_BLOCO (aux 12);
            :
      N2_EXECANT
      fim RECEPÇÃO_URGENTE;
```

onde N2_EXECANT é o procedimento do módulo não interrompível que transfere o controle do N2 para o processo interrompido anteriormente.

```
Procedimento GERENCIADOR;
    Modulo INTGERENC_DESATIVADA[1];
       do sistema importe  transfer;
       importe Exec, Gerenc, N2, N3, nível 2, nível 3,...,GE2,
               GT2, GR2, GE3;
       exporte N2_Evento, N2_Transm, N2_Rec, N3_EVENTO,...;
       procedimento N2_EVENTO;
          início
          Exec:= N2;
          nível 2:= GE2;
          transfer (Gerenc, N2);
          fim N2_EVENTO;
       procedimento N2_TRANSM;
          início
          Exec:= N2;
          nível 2:= GT2;
          transfer (Gerenc, N2)
          fim N2_TRANSM;
```

```
    procedimento N2_REC;
       início
       Exec:= N2;
       nível 2:= GR2;
       transfer (Gerenc, N2);
       fim N2_REC;
    procedimento N3_EVENTO;
       início
       Exec:= N3;
       nível 3:= GE3;
       transfer (Gerenc, N3)
       fim N3_EVENTO;
      :
fim INT GERENC_DESATIVADA ;
    procedimento TIME_OUT;
       início
       se timeout2 ou ocupado então N2_Evento;
       se timeout3 ou ocupado então N3_Evento;
       fim TIME_OUT;
    início
       laço
         se (H2_REC N1 = 0) então N2_Rec;
         se Timeout ou ocupado então Time out;
```

```
        se (H2_TRANS2 <> 0) então N2_Transm;
        se Timecut ou ocupado então Time_out;
                    :
    fim (* laço *)
fim GERENCIADOR;
```

APÊNDICE 2

ALOCAÇÃO DINÂMICA DE MEMÓRIA

As informações necessárias nos blocos para alocação dinâmica são:

(1) Para bloco reservado:

(TAG): informando que este é um bloco reservado.

(SIZE): informando o tamanho do bloco.

(2) Para bloco livre:

Além dos dois campos do bloco anterior temos:

(LINK 1): próximo bloco livre

(LINK 2): bloco livre anterior.

O campo SIZE deve ter dois bytes, pois o tamanho do bloco pode ser maior do que 256 bytes (e menor do que 64k). Por exemplo, na inicialização do sistema, teremos apenas um bloco livre do tamanho da área de alocação. Os campos LINK1 e LINK2 também devem ter dois bytes, pois os endereços dos blocos anterior e posterior podem variar de 0 a M-1. A área de alocação dinâmica tem M bytes: $256 < M < 64K$ bytes.

Para implementarmos o protocolo X-25 ainda precisamos das seguintes informações para alocação (além dos cabeçalhos dos

níveis e a mensagem propriamente dita):

a) NUM - número de filas a que a mensagem pertence.

b) TAM - tamanho da mensagem, pacote ou quadro (Não é necessariamente igual ao tamanho do bloco). Pode ser apenas um byte, pois o tamanho máximo é de 133 bytes (128+5).

c) LINK - próxima mensagem, pacote ou quadro da fila correspondente.

Os campos NUM e TAG podem ser o mesmo, pois NUM só é utilizado em blocos reservados. Podemos supor que TAG = 0 quando o bloco é livre, e TAG = NUM quando o bloco é reservado.

Desta maneira, podemos propor os seguintes dois tipos de bloco, procurando ter compatibilidade entre ambos:

1) Blocos reservados.

1º byte TAG = NUM - número de filas a que o bloco pertence (para termos apenas um bloco).

2º byte: TAM - tamanho da mensagem, pacote ou quadro (não é necessariamente igual ao tamanho do bloco).

3º, 4º bytes:LINK - próxima mensagem, pacote ou quadro da fila correspondente.

5º, 6º bytes:SIZE - tamanho do bloco reservado.

2) Blocos livres:

1º byte: TAG = 0, indicando que o bloco está livre.

2º byte: contém informação irrelevante; apenas para se tornar compatível com o bloco reservado.

3º, 4º bytes: LINK1 - próximo bloco livre.

5º, 6º bytes: SIZE - tamanho do bloco livre.

7º, 8º bytes: LINK2 - bloco livre anterior.

Uma maneira de otimizarmos este esquema de alocação dinâmica é dividirmos o vetor de bytes em duas parte. A primeira parte seria reservada para mensagens, pacotes e quadros de supervisão ou controle e esta parte seria composta de um conjunto de blocos de tamanho fixo (11 bytes). A segunda parte, reservada para mensagens, pacotes e quadros de informação, teria alocação dinâmica.

Portanto, um terceiro tipo de bloco deve ser introduzido:

3) Bloco de tamanho constante (para controle e supervisão):

1º byte: TAG = NUM - número de filas a que o bloco pertence. (se for um pacote de controle, pode estar nas filas de transmissão do nível 1 e retransmissão do nível 2, pois este pacote é um quadro de informação).

2º byte: TAM - tamanho do quadro, pacote ou mensagem (não é necessariamente igual ao tamanho do bloco).

3º, 4º bytes: LINK - se reservado contém um apontador para a próxima mensagem, pacote ou quadro da fila correspondente.

se livre contém um apontador para o próximo bloco da fila de blocos livres.

bytes seguintes: mensagem, pacote ou quadro.

Esta idéia de otimização não pode ser utilizada em sua totalidade porque o nível 1 ao receber um quadro não sabe o seu tamanho antecipadamente. Desta maneira, temos que fornecer ao nível 1 um espaço livre de tamanho máximo (133 bytes) antes da recepção do quadro. Portanto este quadro, independente do seu tipo, será alocado na segunda parte do vetor. No sentido dos níveis mais altos para o nível 1 esta otimização pode ser utilizada.