UNIVERSIDADE FEDERAL DA PARAIBA
CENTRO DE CIENCIAS E TECNOLOGIA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇAO

UMA METODOLOGIA PARA VALIDAÇAO, ATRAVES DE SIMULAÇAO, DE ESPECIFICAÇOES FORMAIS DE PROTOCOLOS DE COMUNICAÇAO

José Neuman de Souza

CAMPINA GRANDE

MARÇO - 1990

JOSE NEUMAN DE SOUZA

UMA METODOLOGIA PARA VALIDAÇAO, ATRAVES DE SIMULAÇAO, DE ESPECIFICAÇOES FORMAIS DE PROTOCOLOS DE COMUNICAÇAO

Dissertação apresentada ao Curso de MESTRADO EM INFORMATICA da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do grau de Mestre.

AREA DE CONCENTRAÇAO - Redes de Computadores

WANDERLEY LOPES DE SOUZA

orientador

CAMPINA GRANDE

MARÇO - 1990

UMA METODOLOGIA PARA VALIDAÇAO, ATRAVES DE SIMULAÇAO DE ESPECIFICAÇOES FORMAIS DE PROTOCOLO DE COMUNICAÇAO

JOSE NEUMAN DE SOUZA

DISSERTAÇAO APROVADA EM 26.03.1990

Wanderley Lopes de Souza
Orientador

Solon Benayon da Silva
Componente da Banca

Joberto Sérgio Barbosa Martins
Componente da Banca

Campina Grande - Pb.
Março - 1990

AGRADECIMENTOS

Gostaria de agradecer ao Prof. Wanderley Lopes de Souza, meu orientador, pelo estímulo e apoio indispensáveis à realização deste trabalho.

Agradeço ao Banco do Nordeste do Brasil S/A (BNB), à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo suporte financeiro a mim concedido. Em especial quero registrar o apoio incondicional que recebi do Chefe do Departamento de Informática do BNB, Sr. José Bezerra de Morais.

Agradeço também a Solon Benayon e a Peter Lonergan, pesquisadores do Centro Científico da IBM Brasil no Rio de Janeiro, pela acolhida agradável e suporte técnico que me dispensaram por ocasião da validação deste trabalho naquele Centro.

Aos funcionários do Núcleo de Processamento de Dados da UFPb, principalmente a Elisa e Leônidas, quero prestar meus agradecimentos, como também a todos os colegas e professores do Departamento de Sistemas e Computação que, direta ou indiretamente, me ajudaram a realizar este trabalho.

Por último, quero agradecer a meus pais, José Eduardo de Souza e Maria da Silva Souza, dedicando-lhes este trabalho.

A Tereza e Juliana

S U M A R I O

1. INTRODUÇAO 1

2. VALIDAÇAO DO DESIGN DE UM PROTOCOLO 12

3. O SERVIÇO E O PROTOCOLO ABRACADABRA 15

3.1 O Serviço Abracadabra 15

3.2 O Protocolo Abracadabra 17

4. A LINGUAGEM Estelle/83 22

5. O COMPILADOR Estelle/83 33

5.1 Estruturas de dados geradas pelo pré-processador Estelle/83 36

5.2 Interligação das estruturas de dados 43

5.3 Interligação das estruturas geradas para o protocolo Abracadabra 48

5.4 Tradução das transições Estelle/83 pelo compilador 57

6. METODOLOGIA PARA VALIDAÇAO, ATRAVES DE SIMULAÇAO, DE ESPECIFICAÇOES FORMAIS DE PROTOCOLOS DE COMUNICAÇAO 61

6.1 Núcleo de simulação 62

6.2 Simulação das especificações formais do serviço e protocolo Abracadabra 67

6.3 Simulação do sistema Abraservice'..... 68

6.4 Validação do design do protocolo Abracadabra 87

6.4.1 Erros encontrados na simulação 89

7. CONCLUSAO 91

REFERENCIAS BIBLIOGRAFICAS 94

ANEXO 1: Especificação formal do serviço Abracadabra em Estelle/83 96

ANEXO 2: Código Pascal do serviço 102

ANEXO 3: Código do núcleo de simulação para o serviço 121

ANEXO 4: Especificação formal do protocolo Abracadabra em Estelle/83 132

ANEXO 5: Código Pascal do protocolo 147

ANEXO 6: Código do núcleo de simulação para o protocolo ... 186

ANEXO 7: Código Pascal das rotinas de suporte 200

ANEXO 8: Alguns traços verificados na simulação 206

LISTA DE FIGURAS

1.1 Modelo Básico de Referência OSI/ISO 3

1.2 Conceitos de Serviço (N), PAS (N) e Protocolo (N) 6

1.3 Ciclo de Desenvolvimento de um Protocolo 9

2.1 Validação do Design do Protocolo (N) 14

3.1 Primitivas do Serviço Abracadabra 15

3.2 UDPs e Correspondentes Primitivas de Serviço (PSs) 17

4.1 MEFE de Estelle/83 22

4.2 Arquitetura do Serviço Abracadabra em Estelle/83 23

4.3 Arquitetura do Protocolo Abracadabra em Estelle/83 23

4.4 Sintaxe para a Definição de um Canal 24

4.5 Especificação dos Canais no Protocolo Abracadabra 25

4.6 Sintaxe para a Definição da Cabeça de um Módulo 26

4.7 Especificação dos Módulos no Protocolo Abracadabra 27

4.8 Sintaxes para a Definição do Corpo de um Módulo 28

4.9 Sintaxe para a Definição de uma Transição 29

4.10 Exemplo de Especificação para a Definição do Refinamento de um Módulo 31

5.1 Etapas e procedimentos necessários para se alcançar um código executável 34

5.2 Estrutura do Compilador Estelle/83 35

5.3 Estrutura de Dados de um Processo 37

5.4 Código Pascal para a Estrutura de um Processo 38

5.5 Código Pascal para as Estruturas dos Processos no Protocolo Abracadabra 38

5.6 Estrutura de Dados de uma Porta 39

5.7 Código Pascal para a Estrutura de uma Porta 41

5.8 Estrutura de Dados de uma Interação 41
5.9 Declaração em Pascal das Interações 42
5.10 Declaração da Estrutura de uma Interação em Pascal 42
5.11 Declaração em Pascal das Interações no Protocolo Abracadabra 43
5.12 Ações de SYSTEM_INIT 45
5.13 Ações de IOident_1o.refinamento 45
5.14 Ações de IOident_refinamento 46
5.15 Código Pascal do Procedimento IOABRA_SERVICE 49
5.16 Código Pascal do Procedimento IOABRA_BODY 50
5.17 Código Pascal do Procedimento IOTRANSCODE_PROC 50
5.18 Encadeamento das Estruturas após a Execução de IOTRANSCODE_PROC 51
5.19 Código Pascal do Procedimento IOSTATION_PROC 52
5.20 Encadeamento das Estruturas após a Execução de IOSTATION_PROC 53
5.21 Código Pascal do Procedimento IOUSER_PROC 54
5.22 Encadeamento das Estruturas após a Execução de IOUSER_PROC 56
5.23 Exemplo de Tradução de uma Transição Estelle/83 em Código Pascal 58
5.24 Exemplo de Tradução de uma Transição Espontânea Estelle/83 em Código Pascal 59
6.1 Código Pascal das Instruções executáveis do Núcleo 65
6.2 Ações do SCHEDULER 65
6.3 Ações de MOD_I 66
6.4 Processos do sistema Abraservice 67
6.5 Processos do Sistema Abraservice' 67

6.6 Tela para Escolha do Processo a ser Executado 68
6.7 Código Pascal do Procedimento MOD_1 70
6.8 Código Pascal da Transição Espontânea Inicialmente Habilitada no Procedimento USER_PROC 71
6.9 Código Pascal do Procedimento OUT 72
6.10 Código Pascal da Transição Habilitada no Procedimento STATION_PROC para Tratar a Primitiva ConReq 73
6.11 Código Pascal da Transição Habilitada no Procedimento TRANSCODE_PROC para Tratar a UDP CR 74
6.12 Código Pascal do Procedimento BuildCR 75
6.13 Código Pascal da Transição Habilitada no Procedimento CMS_PROC para Tratar a Primitiva UnitReq 76
6.14 Código Pascal da Transição Habilitada no Procedimento TRANSCODE_PROC para Tratar a Primitiva UnitInd (lado 2) . 77
6.15 Código Pascal da Transição Habilitada no Procedimento STATION_PROC para Tratar a UDP CR 78
6.16 Código Pascal da Transição Habilitada no Procedimento STATION_PROC para Tratar a Primitiva DisReq 79
6.17 Código Pascal da Transição Habilitada no Procedimento TRANSCODE_PROC para Tratar a UDP DR 80
6.18 Código Pascal da Transição Habilitada no Procedimento TRANSCODE_PROC para Tratar a Primitiva UnitInd (lado 1) . 81
6.19 Código Pascal da Transição Habilitada no Procedimento STATION_PROC para Tratar a UDP DR 81
6.20 Código Pascal da Transição Habilitada no Procedimento STATION_PROC para Tratar a Primitiva ConResp 82

6.21 Código Pascal da Transição Habilitada no Procedimento TRANSCODE_PROC para Tratar a UDP CC 83
6.22 Código Pascal da Transição Habilitada no Procedimento USER_PROC para Tratar a Primitiva ConConf 84
6.23 Código Pascal da Transição Habilitada no Procedimento STATION_PROC para Tratar a Primitiva DatReq 85
6.24 Código Pascal da Transição Habilitada no Procedimento STATION_PROC para Tratar a UDP DT 86
6.25 Código Pascal da Transição Habilitada no Procedimento STATION_PROC para Tratar a UDP AK 87

UMA METODOLOGIA PARA VALIDAÇAO, ATRAVES DE SIMULAÇAO, DE ESPECIFICAÇOES FORMAIS DE PROTOCOLOS DE COMUNICAÇAO

RESUMO

Nesse trabalho é proposta uma metodologia para validar, através de simulação, especificações formais de protocolos, realizadas com a técnica de descrição formal (TDF) "Extended State Transition Language (Estelle)", versão de 1983. Essa metodologia utiliza, como ferramenta, o compilador Estelle/83, desenvolvido junto ao Grupo de Redes de Computadores (GRC) da Universidade Federal da Paraíba (UFPb). A esse compilador é acrescentado um núcleo de simulação que permite o acompanhamento da execução do sistema especificado.

A METHODOLOGIE FOR VALIDATION, BY SIMULATION, OF COMUNNICATION PROTOCOLS SPECIFIED FORMALLY

ABSTRACT

This work presents a methodologie for validation, by simulation, of protocol formal specifications described with the formal description technique (FDT) Extended State Transition Language (Estelle), version 1983. This methodologie uses the Estelle/83 compiler, developed by the Computer Networks Group (GRC) of Paraiba University (UFPb). A simulation kernel is added to this compiler to observe the execution of the specified system.

1. INTRODUÇÃO

Os sistemas distribuídos, como qualquer sistema de comunicação, se caracterizam pela troca de mensagens entre suas entidades. Tais mensagens precisam fluir ordenadamente através do sistema, para que se tenha uma comunicação eficiente e segura. O conjunto de regras e convenções bem definidas que garante essa comunicação é chamado de **protocolo de comunicação**. A descrição informal ou formal desse conjunto de regras é chamada de **especificação do protocolo**.

Qualquer sistema de comunicação, independentemente dos sistemas envolvidos, pressupõe a existência de certo (s) protocolo (s). Basta observar uma conversação telefônica ou um simples diálogo para perceber que há um conjunto implícito de regras regulamentando a comunicação[1]. Quando duas pessoas conversam, sempre que uma fala a outra ouve e vice-versa. Se aquela que ouve não entende o que a outra fala, ela pede para que essa repita o que falou. Isso exemplifica o uso de regras de sincronização e recuperação de mensagens durante um diálogo, como condição necessária para uma boa comunicação.

A natureza distribuída e heterogênea dos sistemas de comunicação de dados impõe a existência de protocolos bem estruturados e especificados, tal que a sua implementação em diversos sistemas permita a intercomunicabilidade desejada.

Existem certas particularidades, além dos problemas inerentes a todo projeto de desenvolvimento de software, que caracterizam o projeto e implementação dos protocolos, tais como:

(a) garantia da compatibilidade entre os diversos componentes do sistema especificado;

(b) implementação desses componentes por grupos distintos de técnicos que trabalham em organizações diferentes;

(c) dificuldade de compreensão do funcionamento do sistema como um todo, devido ao paralelismo das operações executadas nos diversos componentes.

Devido à grande complexidade dos problemas envolvidos no projeto de protocolos, os especialistas da International Organization for Standardization (ISO) e Comité Consultatif International Télégrafique et Téléfonique (CCITT), que são órgãos voltados para padronizações, organizaram as diversas funções dos componentes de um sistema em camadas (ou níveis). Isso deu origem ao **Modelo de Referência para Interconexão de Sistemas Abertos ("Reference Model for Open Systems Interconnection" - RM-OSI)**[2], que estabelece sete camadas para cada sistema aberto, conforme a Fig. 1.1.

O propósito de cada camada é oferecer certos serviços às camadas superiores. Entre cada par de camadas adjacentes existe uma interface, que define quais operações, primitivas e serviços a camada mais baixa fornece à camada mais alta.

A fim de identificar e relacionar camadas adjacentes, a seguinte notação é adotada:

(a) camada (N): qualquer camada especifica;

(b) camada (N+1): a próxima camada mais alta;

(c) camada (N-1): a próxima camada mais baixa.

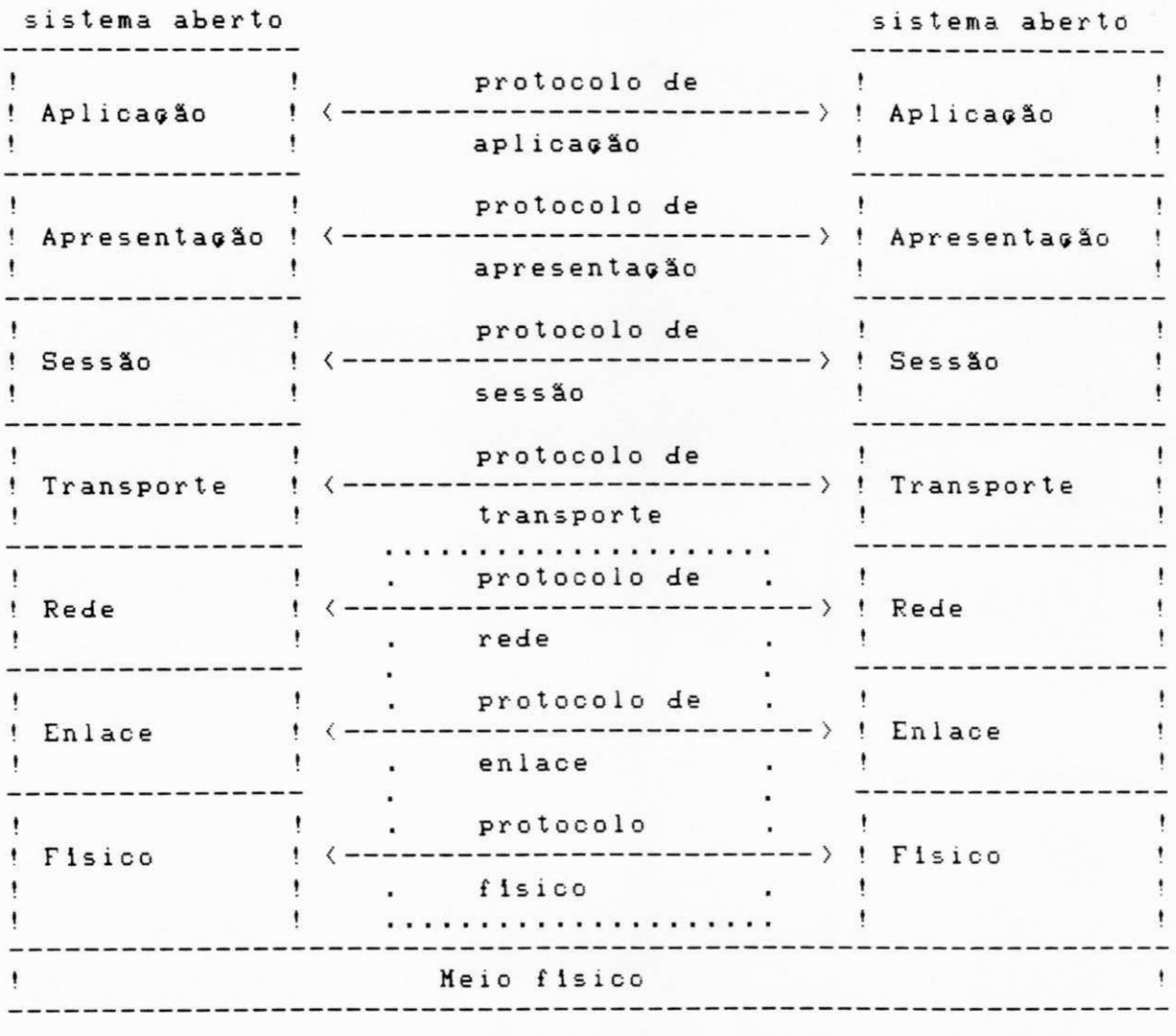


Fig. 1.1 Modelo Básico de Referência OSI/ISO

Resumidamente, as camadas no modelo OSI podem ser definidas da seguinte forma:

(a) FISICA - fornece as características mecânicas, elétricas, funcionais e de procedimento para ativar, manter e desativar conexões físicas para a transmissão de bits entre entidades da camada de enlace; nesse nível, o importante é permitir o envio de uma cadeia de bits pela rede sem se preocupar com o significado desses bits ou como são agrupados.

(b) ENLACE - tem como objetivo detectar e possivelmente corrigir erros que ocorram no nível físico, convertendo um canal de transmissão não confiável em um canal confiável para o uso das entidades no nível de rede; isso é conseguido através da partição da sequência de bits em quadros, inserindo em cada um alguma forma de redundância para detecção de erros.

(c) REDE - fornece uma trajetória de conexão (conexão de rede) entre um par de entidades da camada de transporte, possivelmente passando por nós intermediários, dando a essas entidades independência quanto às questões de roteamento e controle de congestionamento associadas ao estabelecimento e operação da conexão.

(d) TRANSPORTE - fornece uma comunicação fim a fim confiável, através da transferência transparente de dados entre entidades da camada de sessão; esse nível de protocolo é necessário porque o nível de rede não garante que a sequência de bits chegue a seu destino, onde pacotes podem ser perdidos ou mesmo reordenados;

outra função importante desse nível de protocolo é a multiplexação de conexões.

(e) SESSÃO - tem como objetivo organizar e sincronizar o diálogo, e gerenciar a troca de dados entre entidades da camada de apresentação.

(f) APRESENTAÇÃO - tem como objetivo realizar transformações adequadas nos dados, antes de seu envio ao nível de sessão; esse nível de protocolo resolve problemas de diferença de sintaxe entre sistemas abertos.

(g) APLICAÇÃO - serve de janela entre usuários que querem se comunicar através do modelo OSI, através da qual ocorre toda troca de informação significativa para esses usuários.

Os protocolos pertencentes às camadas FISICA, ENLACE e REDE constituem os protocolos de BAIXO NIVEL e os demais compõem os protocolos de ALTO NIVEL.

Alguns conceitos são de vital importância para um bom entendimento do modelo, tais como:

(a) serviço (N): facilidades que a camada (N) e as camadas inferiores fornecem às entidades (N+1), na fronteira entre a camada (N) e a camada (N+1);

(b) ponto de acesso ao serviço (PAS) (N): ponto no qual um par de entidades em camadas adjacentes usam ou oferecem serviços;

(c) protocolo (N): conjunto de regras e formatos (semânticos e sintáticos), que determina o comportamento das entidades (N) que se comunicam, se sincronizam e operam de forma concorrente via os pontos de acesso ao serviço (N).

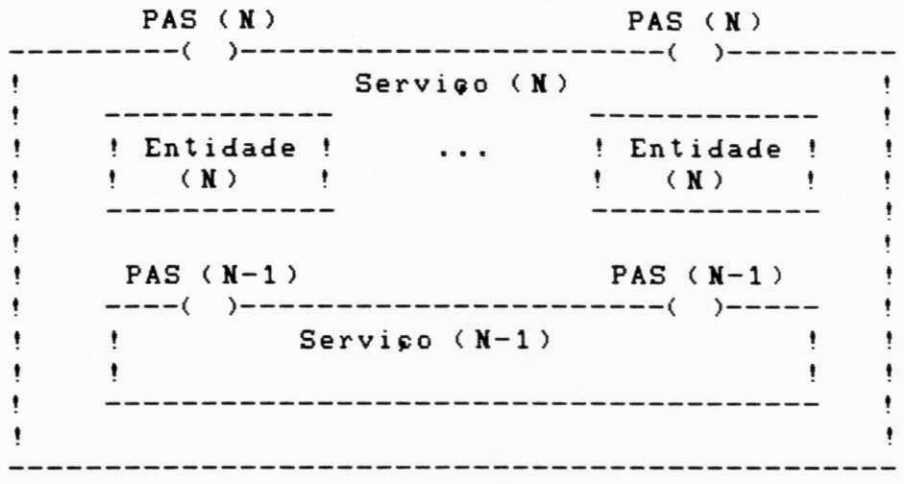


Fig. 1.2 Conceitos de serviço (N), PAS (N) e protocolo (N)

Observando a Fig. 1.2, pode-se dizer que através dos serviços oferecidos pela camada (N-1), as entidades da camada (N) cooperam entre si, conforme o protocolo (N), a fim de oferecer serviços (N) mais elaborados à camada (N+1).

A especificação de serviços (N) é baseada na definição das primitivas de serviço (N), a ordem de execução dessas primitivas e os efeitos da execução de cada uma delas. Os detalhes relativos a essas primitivas não fazem parte dessa especificação, e sim da especificação da interface entre as camadas (N) e (N+1).

Na especificação do protocolo (N) são definidas as entidades cooperantes da camada (N) e como essas entidades fornecem os seus serviços.

Os primeiros protocolos de comunicação foram especificados informalmente, utilizando-se as linguagens naturais como ferramentas. Exemplo disso é a especificação informal do protocolo BSC da IBM. O uso dessa técnica mostrou as seguintes inconveniências:

(a) imprecisão (ambiguidade), permitindo diferentes interpretações para a mesma descrição de um objeto;

(b) ineficiência, geralmente são muito longas as especificações;

(c) impropriedade, não adequada para a verificação e geração automática do respectivo protocolo.

Conscientes dos problemas gerados pelas implementações realizadas a partir de especificações informais, os especialistas evoluíram para as especificações semi-formais. Isso corresponde às descrições realizadas através de linguagem natural associada a representações gráficas e/ou diagramas de estados. Como exemplo de especificação usando linguagem natural associada a diagramas de estado estão as primeiras descrições do protocolo X.25, usadas pelo CCITT. A descrição do protocolo de transporte da ISO é um exemplo de especificação semi-formal, usando linguagem natural, diagramas de estado e representações gráficas.

Apesar do avanço em relação à especificação informal, as especificações semi-formais ainda eram insuficientes. Ficou evidenciado que o uso de tais especificações permitia que erros provenientes da especificação e/ou do design de um protocolo, que poderiam ser detectados e corrigidos nas primeiras fases do seu ciclo de desenvolvimento, proliferavam por todas implementações, tornando o trabalho de depuração de custo elevado e extremamente difícil.

No final da década de 70, técnicas de descrição formal começaram a ser propostas para as especificações formais de serviços e protocolos de comunicação. Segundo o modelo utilizado, essas técnicas podem ser classificadas em:

(a) técnicas baseadas em autômatos finitos ou transição de estados (máquinas de estados finitas, linguagens formais, grafos);

(b) técnicas baseadas em linguagens de programação;

(c) técnicas híbridas.

A ISO e o CCITT, interessados também na fixação de padrões para a especificação formal dos protocolos de comunicação, desenvolveram as seguintes técnicas de descrição formal (TDFs):

(a) "Extended state transition language (Estelle)", técnica baseada no conceito de máquina de estados finita estendida (MEFE), desenvolvida pela ISO;

(b) "Language of temporal ordering specification (Lotos)", técnica baseada em ordem temporal de primitivas de interação, desenvolvida pela ISO;

(c) "Specification and description language (SDL)", técnica originalmente empregada na descrição dos sistemas de chaveamento, desenvolvida pelo CCITT.

A Fig. 1.3 ilustra as várias atividades presentes no ciclo de desenvolvimento de um protocolo:

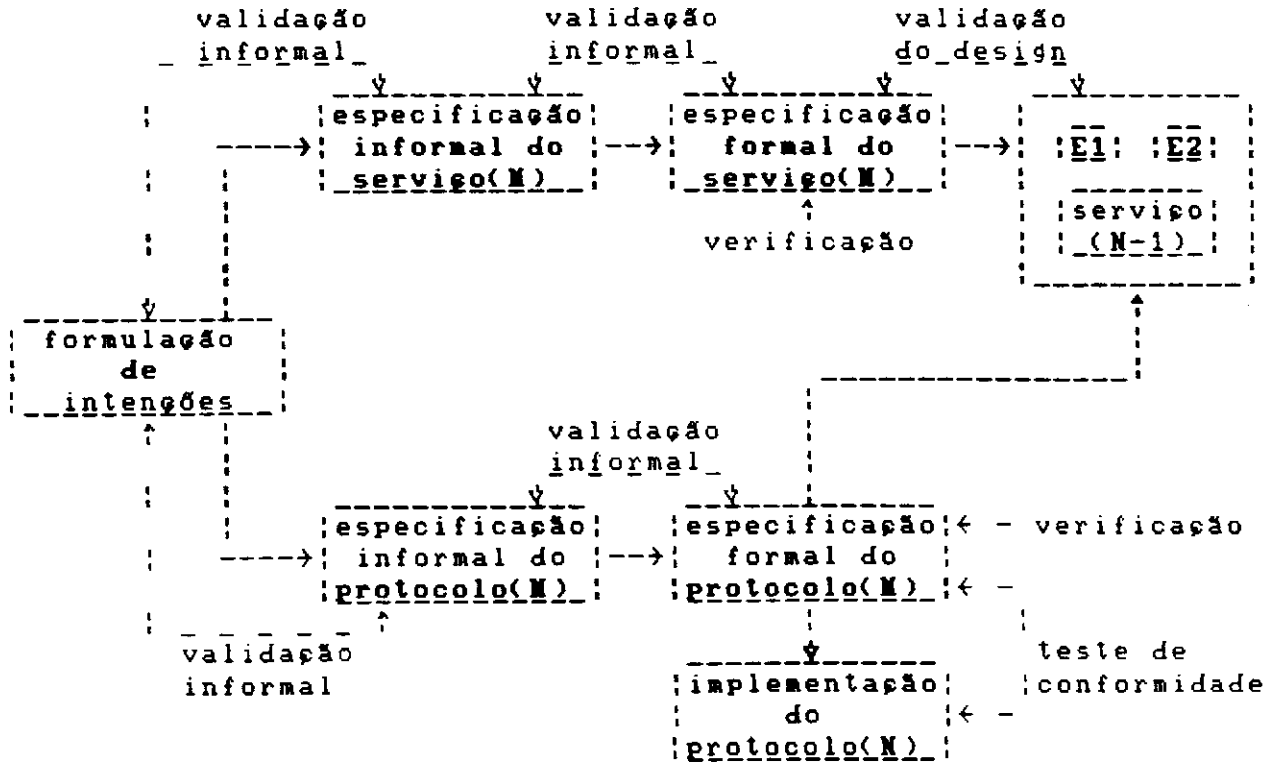


Figura 1.3 Ciclo de desenvolvimento de um protocolo.

Na etapa de formulação de intenções, são definidas, genericamente, as principais características do serviço e do protocolo. Em seguida essas características são transcritas numa linguagem natural, associada a tabelas e/ou diagramas de transição, constituindo-se nas especificações informais. Na etapa seguinte são produzidas as especificações formais, utilizando-se uma técnica de descrição formal, as quais serão as referências confiáveis para as demais fases do ciclo de desenvolvimento do protocolo. A implementação do protocolo (em software, hardware ou firmware) é gerada então a partir das especificações formais.

Associadas às etapas acima, podem ser identificadas as seguintes atividades de validação:

(a) validação informal - onde deve ser verificada informalmente a conformidade das intenções do especificador em relação às especificações geradas;

(b) verificação - onde é examinada a ocorrência de propriedades gerais (ausência de impasses, ausência de ciclos improdutivos, vivacidade, término apropriada etc) e propriedades específicas que garantam a correção do protocolo;

(c) validação do design - onde é verificado se o funcionamento conjunto de duas entidades (N), que operam segundo a especificação do protocolo (N) e se comunicam através do serviço (N-1), satisfaz a especificação do serviço (N) e

(d) **teste de conformidade** - onde é verificada a conformidade da implementação em relação à especificação.

A utilização de TDFs para a especificação formal dos serviços e protocolos (sobretudo os relativos ao modelo OSI), além de fornecer descrições claras e concisas do sistema que está sendo concebido, suportam uma análise rigorosa e validação passo a passo das diferentes etapas do ciclo de vida de um protocolo.

O objetivo desse trabalho é propor uma metodologia para validar, através de simulação, especificações formais e design de protocolos, realizados com a TDF Estelle, versão de 1983. Essa metodologia utiliza, como ferramenta, o compilador Estelle/83, desenvolvido junto ao Grupo de Redes de Computadores (GRC) da Universidade Federal da Paraíba (UFPb). A esse compilador é acrescentado um núcleo de simulação que permite o acompanhamento da execução do sistema especificado.

2. VALIDAÇÃO DO DESIGN DE UM PROTOCOLO

Durante o desenvolvimento de um sistema, normalmente as especificações são elaboradas em diferentes níveis de abstração. A validação do projeto desse sistema é uma atividade importante e tem como objetivo garantir que as várias especificações sejam consistentes. Isso porque à medida que o sistema vai sendo refinado, especificações com mais e mais detalhes vão sendo produzidas, até que se chegue a um nível em que a implementação dessas partes seja feita numa linguagem de programação ou em hardware.

As técnicas de validação desempenham um papel fundamental na investigação do comportamento das especificações e implementações dos protocolos, assegurando-lhes uma maior confiabilidade. Tais técnicas compreendem[3]:

(a) verificação, que utiliza métodos baseados em raciocínio lógico para verificar algumas propriedades desejadas aos protocolos;

(b) simulação, que consiste em exercitar uma entidade (especificação, design ou implementação) objetivando a detecção de erros;

(c) teste, que normalmente é utilizado para verificar a conformidade entre a especificação de um protocolo e a sua implementação.

As técnicas de verificação possibilitam obter resultados conclusivos sobre o comportamento de um protocolo, através do exame exaustivo de todas as situações previstas. Entretanto, a aplicabilidade de tais técnicas, em protocolos complexos, é susceptível a uma série de restrições, para evitar problemas do tipo "explosão de estados".

Simulação e teste são técnicas que não proporcionam resultados definitivos, pois a investigação do protocolo é limitada a alguns cenários definidos a priori. Entretanto , tais técnicas permitem detectar uma série de erros em protocolos complexos, outorgando um certo grau de confiabilidade à entidade analisada.

Para realizar uma análise completa da semântica da especificação de um protocolo, é necessário levar em consideração o ambiente no qual ele será inserido. Em relação ao modelo OSI, essa análise é realizada durante a validação do design do protocolo.

A especificação e a análise estática de um sistema, constituído das entidades E1 e E2 e do serviço (N-1) (Fig. 2.1), podem ser realizadas empregando-se o compilador Estelle/83. Para a análise dinâmica, um conjunto de cenários pode ser aplicado a esse sistema e o seu comportamento pode ser investigado através de sua simulação controlada.[4]

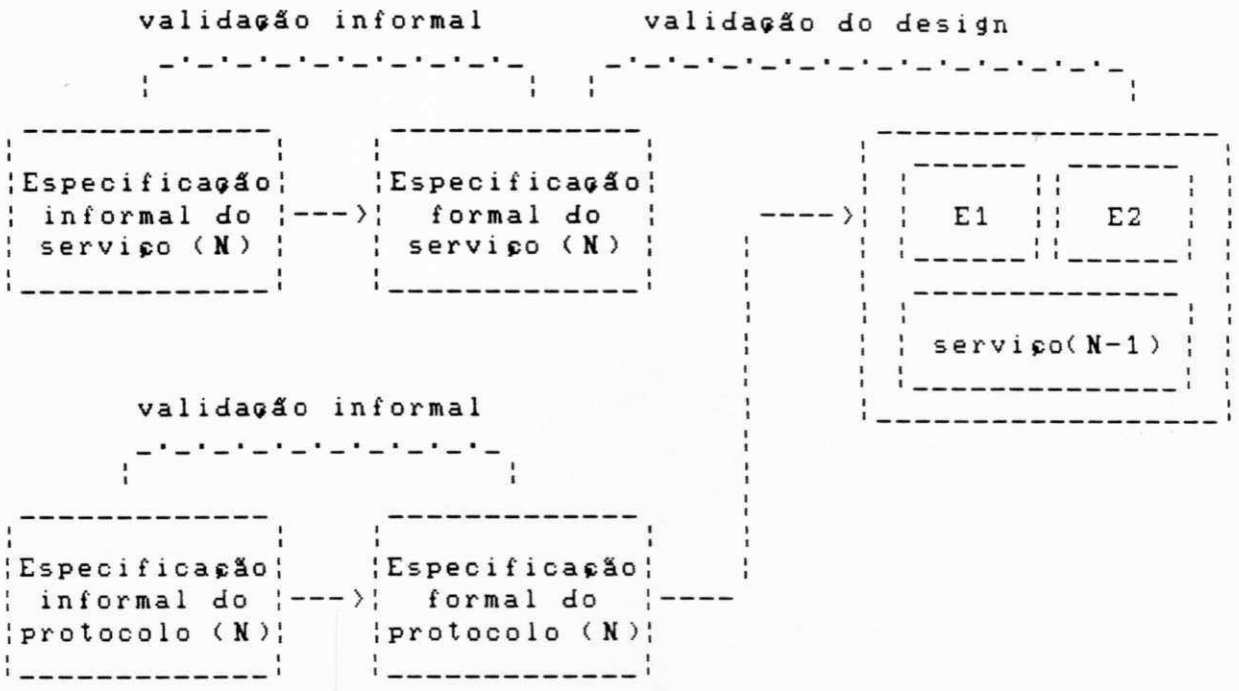


Fig. 2.1 Validação do design do protocolo (N)

Na metodologia proposta, a validação do design é realizada simulando-se as especificações formais do serviço e do protocolo, as quais são estimuladas através de um conjunto de cenários. A detecção de erros é efetuada através do exame comparativo dos traços (sequências válidas de primitivas de serviço e Unidades de Dados de Protocolo (UDPs)) obtidos durante as simulações.

3. O SERVIÇO E O PROTOCOLO ABRACADABRA

Para ilustrar a metodologia, foi utilizado como exemplo o protocolo Abracadabra[5]. Esse protocolo foi definido por especialistas do CCITT e da ISO, sendo que o objetivo principal desse trabalho foi o de promover o emprego de TDFs na especificação dos padrões relativos ao modelo OSI.

Informalmente, o serviço e protocolo Abracadabra são descritos a seguir.

3.1 O serviço Abracadabra

O **Serviço Abracadabra** opera entre um par de estações, sendo que cada estação dispoẽ de uma interface entre o usuário local e as entidades do protocolo. Esse serviço é confiável e orientado à conexão entre um par de usuários. As primitivas utilizadas e o sequenciamento normal das mesmas são:

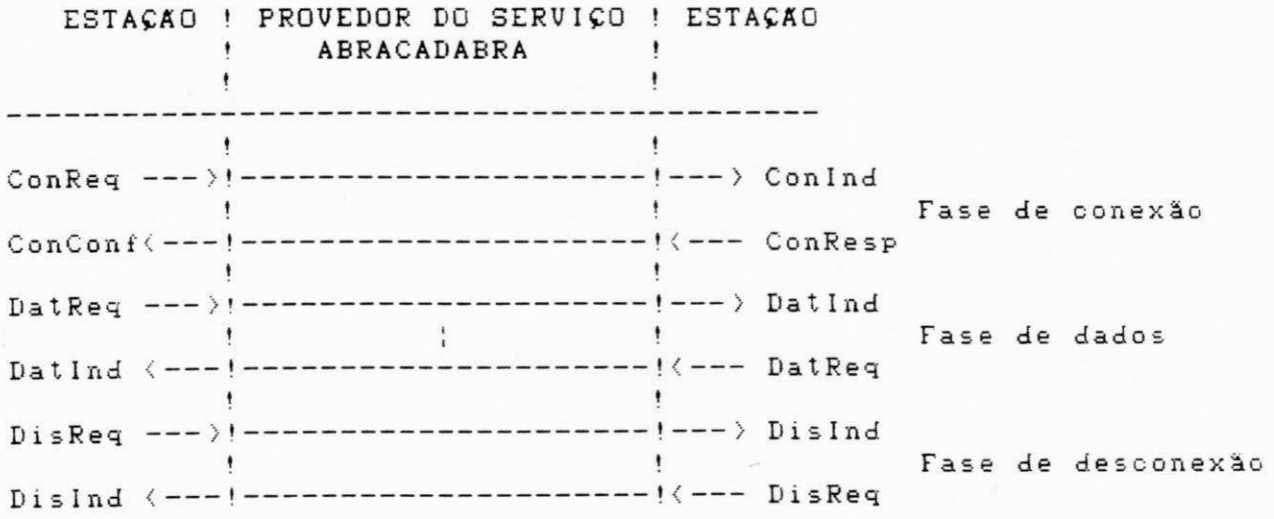


Fig 3.1 Primitivas do serviço Abracadabra

obs: somente as primitivas **DatReq** e **DatInd** carregam um parâmetro, que é a Unidade de Dados de Serviço (UDS)

Em relação à fase de conexão:

(a) poderá ser iniciada por qualquer uma das estações;

(b) se ambas iniciarem a conexão, então em cada lado veicularão somente as primitivas **ConReq** e **ConConf**;

(c) o iniciador de uma conexão poderá abandoná-la, enviando um **DisReq**, antes de receber um **ConConf**;

(d) o respondedor poderá também abandoná-la, enviando um **DisReq**, após ter recebido um **ConInd**.

Em relação à fase de transferência de dados:

(a) estabelecida a conexão, ambas as estações poderão enviar **DatReq** (que serão entregues como **DatInd**);

(b) normalmente as mensagens de dado são preservadas na sua sequência e nos seus conteúdos;

(c) ocorrendo uma desconexão, um número ilimitado de mensagens, já entregues ao serviço, podera ser perdido;

(d) a transferência de dados está sujeita a um controle de fluxo por "back-pressure".

Em relação à fase de desconexão:

(a) qualquer uma das estações poderá iniciar o término de uma conexão já estabelecida;

(b) se ambas iniciarem a desconexão, esta será terminada imediatamente e em cada lado veiculará somente a primitiva **DisReq**;

(c) a qualquer momento o provedor poderá terminar uma conexão (ou a tentativa de estabelecimento desta), sendo que cada estação será informada através de um DisInd (desde que ela não tenha emitido um **DisReq**);

(d) uma vez terminada uma conexão, cada estação poderá iniciar uma nova através de um **ConReq**.

3.2 O protocolo Abracadabra

O **Protocolo Abracadabra** é simétrico, opera entre um par de estações através de um meio "full-duplex" não confiável, fornecendo uma comunicação bilateral, simultânea e confiável ao par de usuários. As duas estações trocam as seguintes UDPs:

UDP	SIGNIFICADO	PRIMITIVAS (PS) CORRESPONDENTES	FASE
CR	requisição de conexão	ConReq, ConInd	Conexão
CC	confirmação de conexão	ConResp, ConConf	
DT	transferência de dados	DatReq, DatInd	Dados
AK	reconhecimento	--------	
DR	requisição de desconexão	DisReq, DisInd	Desconexão ou Erro
DC	confirmação de desconexão	--------	

Fig. 3.2 UDPs e correspondentes Primitivas de Serviço (PSs)

obs: somente as UDPs DT e AK possuem parâmetros. DT carrega uma UDS e ambas carregam um bit de sequenciamento;
cada UDS do Serviço Abracadabra é carregada numa única UDP DT e cada UDP do Protocolo Abracadabra é carregada numa única UDS do Meio de Comunicação.

O Protocolo Abracadabra é parametrizado por duas constantes:

(a) **N** ( ) 0) define o número máximo de tentativas de transmissão de uma mesma UDP, sem a recepção do seu reconhecimento;

(b) **P** ( ) atraso total de trânsito da informação) define o tempo de espera para a retransmissão.

Em relação à fase de conexão:

(a) a sequência normal de UDSs e UDPs, entre duas estações A (iniciadora) e B (respondedora) é **ConReq**(A), **CR**(A-->B), **ConInd**(B), **ConResp**(B), CC(B-->A), **ConConf**(A);

(b) se for recebido DisReq ou DR, o protocolo entrará na fase de desconexão;

(c) outra UDP diferente de **CR**, **CC** ou **DR** deverá ser ignorada;

(d) se não houver resposta ao CR no intervalo P, este será retransmitido;

(e) após **N** tentativas sem sucesso, o protocolo entrará na fase de erro.

Em relação à fase de transferência de dados:

(a) a sequência normal de UDSs e UDPs, entre duas estações A (emissora) e B (receptora) é **DatReq**(A), **DT**(A-->B), **DatInd**(B), **AK**(B-->A);

(b) se após a transmissão de um DT o AK correspondente não for recebido no intervalo P, o DT será retransmitido;

(c) após **N** tentativas sem sucesso, o protocolo entrará na fase de erro;

(d) **DTs** e **AKs** carregam um bit de sequenciamento, que é independente para cada direção de transmissão;

(e) esse bit é inicializado a 0, durante a conexão, e uma vez enviado um **DT**, o correspondente **AK** deverá carregar um bit cujo valor é diferente (o próximo) do transportado pelo **DT**;

(f) se o valor do bit transportado pelo **AK** for incorreto, o protocolo entrará na fase de erro;

(g) ao receber um **DT**, na sequência correta, um **DatInd** deverá ser emitido, assim como um **AK** contendo o próximo valor do bit. Caso contrário, somente um **AK**, contendo o mesmo valor do bit recebido, deverá ser emitido;

(h) se dois **DTs** forem recebidos, antes do envio do **AK** relativo ao primeiro DT, o protocolo entrará na fase de erro;

(i) ao receber um **DR** o protocolo entrará na fase de desconexão;

(j) se um **CR** adicional for recebido antes de qualquer **DT** ou **AK**, um **CC** deverá ser enviado;

(k) se outra UDP, diferente de **DT**, **AK**, **DR** ou **CR** (satisfazendo a condição anterior), for recebida, o protocolo entrará na fase de erro.

Em relação às fases de desconexão e erro:

(a) a sequência normal de UDSs e UDPs, supondo a estação A iniciadora, é DisReq(A), DR(A-->B), DisInd(B), DC(B-->A);

(b) se após a transmissão de um **DR** o DC não for recebido no intervalo **P**, o **DR** será retransmitido;

(c) após **N** tentativas sem sucesso, a conexão será considerada terminada;

(d) se após o envio do **DR** for recebido um **DR** ao invés do DC (**DisReq** "simultâneos"), a conexão será considerada terminada;

(e) se **DRs** adicionais forem recebidos, os **DCs** deverão ser emitidos;

(f) terminada a conexão, qualquer UDP diferente de DR ou CR deverá ser ignorada;

(g) ao detectar um erro no protocolo, a entidade deverá emitir um DisInd e um DR, nesta sequência. O protocolo entrará na fase de erro, que será idêntica à fase de desconexão.

Em relação ao meio de comunicação:

(a) é "full-duplex" e transparente, mas não garante a entrega de mensagens. Estas podem ser perdidas, mas não podem ser desordenadas, corrompidas, inventadas ou duplicadas;

(b) as primitivas **UnitReq** e **UnitInd** carregam UDSs que correspondem às UDPs do Protocolo Abracadabra.

4. A LINGUAGEM Estelle/83

A linguagem Estelle é uma TDF híbrida cujo modelo é baseado numa Máquina de Estados Finita (MEF) estendida pelas construções da linguagem de programação Pascal. É o resultado do trabalho realizado pelo grupo ISO TC 97/SC16/WG1, subgrupo B, criado pela ISO para desenvolver novas técnicas formais para a descrição dos sistemas OSI.

Estelle/83[6] (uma das primeiras versões de Estelle[7]) combina dois tipos de notações:

Estelle/83 = MEF	**ESTADOS** **INTERAÇÕES** **TRANSIÇÕES**	+	**VARIAVEIS** **PARAMETROS** **PRIORIDADES** **Pascal**	= MEFE

Fig. 4.1 MEFE de Estelle/83

Em Estelle/83 uma especificação é constituída de um módulo (**module**) ou de um conjunto de módulos. Cada módulo é representado por uma caixa preta com portas (**port**) de entrada/saída. Canais (**channel**) de comunicação bidirecional podem interligar módulos através de suas portas.

Essas construções são exemplificadas nas Fig. 4.2 e 4.3, onde estão modeladas as arquiteturas do serviço e protocolo Abracadabra respectivamente.

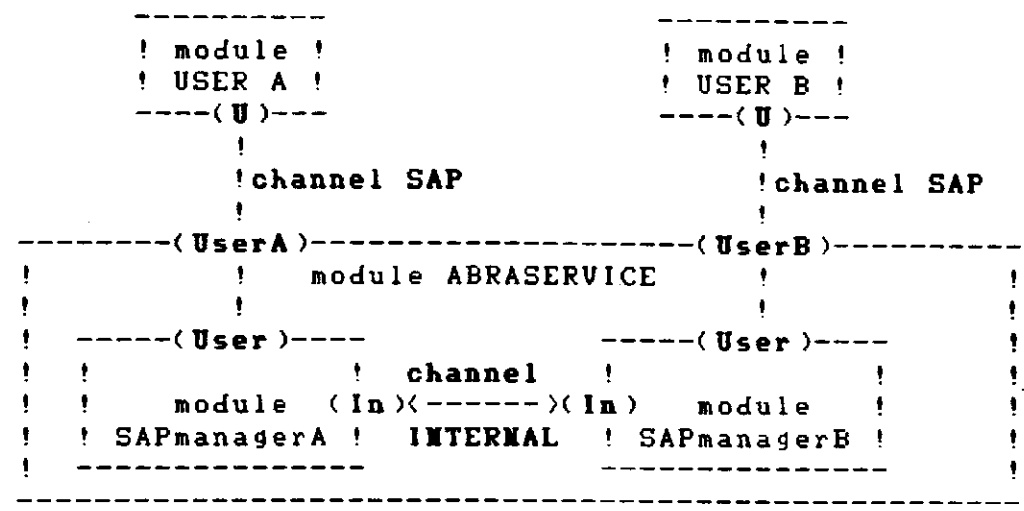


Fig. 4.2 Arquitetura do Serviço Abracadabra em Estelle/83

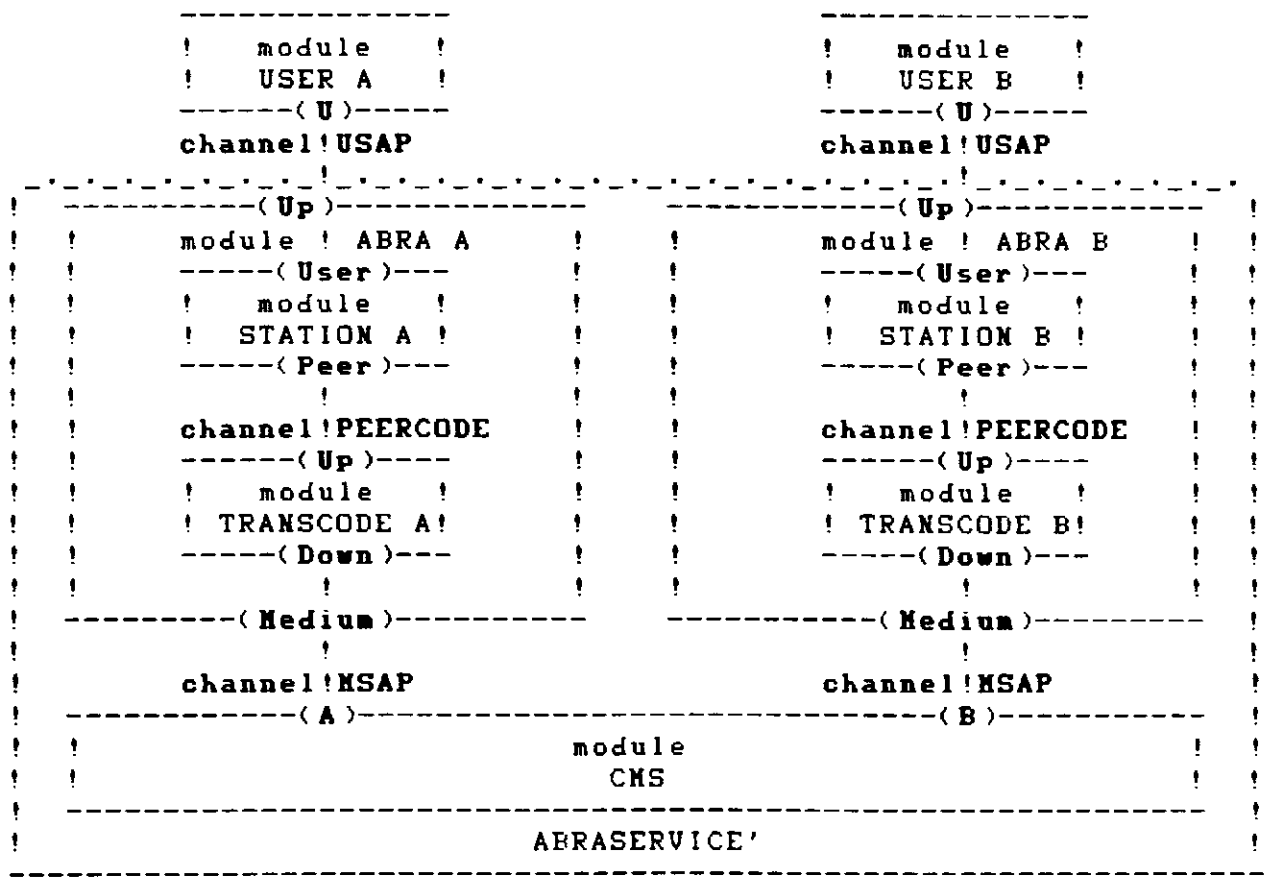


Fig. 4.3 Arquitetura do Protocolo Abracadabra em Estelle/83

Módulos trocam primitivas de comunicação (interações). Um módulo pode enviar uma primitiva (por uma de suas portas) a um outro módulo, desde que ambos estejam interligados através do mesmo canal. O canal é uma construção que permite desvincular a especificação das primitivas de comunicação da especificação dos módulos.

```
channel Ident_Canal(Papel1, Papel2);
  by Papel1 : Ident_Interação(Lista_Parâmetros);
                              ;
              Ident_Interação(Lista_Parâmetros);
  by Papel2 : Ident_Interação(Lista_Parâmetros);
                              ;
              Ident_Interação(Lista_Parâmetros);
end Ident_Canal
```

Fig. 4.4 Sintaxe para a definição de um canal

A especificação da Fig. 4.4 indica que os módulos, a serem conectados (através de suas portas) às extremidades do canal, desempenharão as funções **Papel1** e **Papel2**. O que desempenhar a função **Papel1** pode emitir as primitivas definidas em **Papel1** e deve receber as primitivas definidas em Papel2. O contrário deve ocorrer com o módulo que desempenhar a função Papel2.

Os canais existentes na arquitetura do protocolo Abracadabra (Fig. 4.3) têm a seguinte especificação em Estelle/83:

```
channel USAP(user, provider);
  by user:
       ConReq;
       ConResp;
       DatReq(userdata:userdatatype);
       DisReq;

  by provider:
       ConInd;
       ConConf;
       DatInd(userdata:userdatatype);
       DisInd;
end USAP;
```

```
channel PEERCODE(all);
  by all:
       CR;
       CC;
       DT(seq:seqtype; userdata:userdatatype);
       AK(seq:seqtype);
       DR;
       DC;
end PEERCODE;

channel MSAP(user, provider);
  by user:
       UnitReq(unitdata:unitdatatype);

   by provider:
       UnitInd(unitdata:unitdatatype);
end MSAP;
```

Fig. 4.5 Especificação dos canais no protocolo Abracadabra

Os módulos que possuem suas portas conectadas a um canal do tipo USAP podem desempenhar os papéis **user** e **provider**. O módulo que desempenhar o papel **user** poderá emitir as primitivas ConReq, ConResp, DatReq e DisReq e deverá receber as primitivas ConInd, ConConf, DatInd e DisInd. O inverso deverá ocorrer com o módulo que desempenhar o papel **provider**.

Os módulos que tiverem suas portas conectadas a um canal do tipo PEERCODE poderão emitir e receber (all) as primitivas CR, CC, DT, AK, DR e DC.

Os módulos que possuem suas portas conectadas a um canal do tipo MSAP podem desempenhar os papéis **user** e **provider**. O módulo que desempenhar o papel **user** poderá emitir através de sua porta a primitiva UnitReq e receber a primitiva UnitInd. O comportamento inverso é observado se o módulo desempenhar o papel **provider** na mesma porta,isto é, poderá emitir UnitInd e receber UnitReq.

A comunicação entre módulos pode ser através de "rendez-vous" ou através de filas FIFO ("First In - First Out") associadas às portas desses módulos. No primeiro caso, para que dois módulos possam trocar interações é necessário que ambos (emissor e receptor) estejam prontos (comunicação síncrona). No segundo caso, a primitiva emitida é colacada na fila associada à porta do módulo receptor (comunicação assíncrona).

A especificação de um módulo é constituída de duas partes: cabeça e corpo. A cabeça de um módulo representa o seu nível mais alto de abstração e sua definição é baseada na descrição de suas portas. Essa cabeça define um tipo módulo. Variáveis de um tipo módulo representam cópias desse tipo e todas possuem a mesma visibilidade externa.

```
module Ident_Cabeça;
  Ident_Porta : Ident_Canal(Papel);
                :
  Ident_Porta : Ident_Canal(Papel);
end Ident_Cabeça;
```

Fig. 4.6 Sintaxe para a definição da cabeça de um módulo

Nessa especificação são definidas (indiretamente) as interações de um módulo, vinculando-se suas portas aos canais. Uma porta do tipo **Ident_Canal(Papel)** troca as primitivas definidas no canal **Ident_Canal** e desempenha a função **Papel** em relação a esse canal (o que indica a direção das interações).

Exemplificando, os módulos na Fig. 4.3 têm a seguinte especificação em Estelle/83:

```
module USER;
  U : USAP(user);
end USER;

module ABRA;
  Up     : USAP(provider);
  Medium : MSAP(user);
end ABRA;

module CMS;
  A, B : MSAP(provider);
end CMS;
```

Fig. 4.7 Especificação dos módulos do protocolo Abracadabra

O módulo USER tem uma porta que está conectada a um canal USAP, onde desempenha o papel user. Esse módulo pode, portanto, emitir as primitivas ConReq, ConResp, DatReq e DisReq através da porta U e receber ConInd, ConConf, DatInd e DisInd nessa mesma porta.

O módulo ABRA tem a porta up conectada a um canal USAP, desempenhando o papel provider, através da qual pode emitir as primitivas ConInd, ConConf, DatInd e DisInd para o módulo USER e receber desse módulo as primitivas ConReq, ConResp, DatReq e DisReq. Através da porta medium, que está conectada a um canal MSAP, o módulo ABRA pode emitir a primitiva UnitReq para o módulo CMS e receber desse módulo a primitiva UnitInd, pois está desempenhando o papel user.

O módulo CMS tem as suas duas portas conectadas a um canal MSAP, desempenhando o papel provider. Só poderá, portanto, receber a primitiva UnitReq e emitir a primitiva UnitInd em qualquer uma das portas.

A especificação do corpo de um módulo pode ser realizada através do seu refinamento (**refinement**) em submódulos ou através da definição do seu comportamento em termos de um processo (**process**).

```
refinement Ident_Corpo for Ident_Cabeça;
  <refinamento do corpo>
  <"instanciação">
  <estabelecimento dos "links" de comunicação>
end Ident_Corpo;

process Ident_Corpo for Ident_Cabeça;
  <declarações>
  <"inicializações">
  <transições>
end Ident_Corpo
```

Fig. 4.8 Sintaxes para a definição do corpo de um módulo

No refinamento de um módulo são especificados os seus submódulos (que por sua vez podem ser refinados) e os canais internos (caso haja comunicação entre esses submódulos). Terminado o refinamento, cópias dos submódulos filhos (variáveis do tipo módulo) são criadas e instanciadas (**init**) com os seus respectivos corpos (previamente definidos). Finalmente os links de comunicação são estabelecidos: as portas das instâncias dos submódulos, interligadas através de canais internos, são conectadas (**connect**) e as portas do módulo pai, que devem ser vinculadas às portas de seus filhos, são substituídas (**replace**) pelas portas das respectivas instâncias.

Na especificação de um processo são declarados os objetos a serem manipulados, tais como: as portas do módulo que possuem filas associadas (queued), constantes, tipos, variáveis, os estados de controle da MEFE (variável state), funções e procedimentos. Em seguida são inicializadas a variável de estado de controle (também chamada de principal) e as variáveis de estado adicionais (Pascal). Finalmente o comportamento do módulo é descrito através de um conjunto de transições.

```
trans
                              (condições)
  priority  < número C N >       (em relação às outras transições)
  when      < port.evento >                (interação de entrada)
  provided  < predicado >        (expressão booleana habilitadora)
  from      < estado_a >               (estado de controle vigente)
                            (ações)
  to        < estado_b >             (próximo estado de controle)
    begin          (atualização das variáveis adicionais (Pascal))
      ..........
      out  < port.evento >                  (interação de saída)
      ..........
    end;
```

Fig. 4.9 Sintaxe para a definição de uma transição

Cada transição é composta de duas partes: condições e ações. As condições são constituídas de cláusulas próprias a Estelle/83, sendo que a ordem dessas cláusulas é irrelevante e pelo menos uma cláusula deve ser declarada para cada transição. As ações são constituídas da cláusula to, de declarações Pascal, de extensões (e.g., out) e de restrições (e.g., os apontadores só podem ser utilizados na parte Pascal). Transições que não possuem a cláusula when são denominadas espontâneas.

As construções das Figs. 4.8 e 4.9 são exemplificadas através da seguinte especificação em Estelle/83, relativa à arquitetura do protocolo Abracadabra(Fig. 4.3):

```
refinement ABRA_BODY for ABRA;

  module STATION;

    User : USAP(provider);
    Peer : PEERCODE(all);

  end STATION;

  process STATION_PROC for STATION;

    ...;

    trans

      when User.ConReq
      from CLOSED
      to CRSENT
      begin                                        (1)
        INITVAR;
        out Peer.CR;
        CRretranremaining := N-1
      end;

      ...;

      provided CRretranremaining > 0
      from CRSENT                                  (2)
      to CRSENT
      delay (P)   /*esta cláusula não está implementada no
                    compilador Estelle/83; ela é simulada
                    através das funções TIMEOUT, SETTIMER e
                    RESETTIMER*/
      begin
        CRretranremaining := CRretranremaining-1;
        out Peer.CR
      end;

      ...;

  end STATION_PROC;

  module TRANSCODE;

    Up   : PEERCODE(all);
    Down : MSAP(user);

  end TRANSCODE;
```

```
process TRANSCODE_PROC for TRANSCODE;
  ...; /*especificação do corpo do processo*/
end TRANSCODE_PROC;
     /*instanciação dos módulos em ABRA_BODY*/
S  : STATION with STATION_PROC;
XC : TRANSCODE with TRANSCODE_PROC;
connect S.Peer to XC.Up;
replace Up     by S.User;
        Medium by XC.Down;
end ABRA_BODY;
```

Fig. 4.10 Exemplo de Especificação para a Definição do Refinamento de um Módulo

Observa-se que o módulo ABRA é refinado nos submódulos STATION e TRANSCODE. Cada um desses módulos, por sua vez, tem o seu comportamento descrito através de seus respectivos processos.

O módulo STATION tem duas portas, **User** e **Peer**. Através da porta **User**, esse módulo pode emitir as primitivas ConInd, ConConf, DatInd e DisInd e receber ConReq, ConResp, DatReq e DisReq. Através da porta **Peer**, o módulo STATION pode tanto emitir como receber as UDPs CR, CC, DT, AK, DR e DC. No corpo do processo do módulo STATION, são mostradas duas transições. A transição de número 1 descreve o comportamento do módulo quando estiver recebendo na porta User a primitiva ConReq, estando no estado CLOSED. O módulo muda seu estado para CRSENT e coloca na porta Peer a interação CR. A transição de número 2 é uma transição espontânea, já que não existe a cláusula when. Para que essa transição esteja habilitada, basta que as cláusulas **provided**

e **from** sejam satisfeitas. O módulo permanece no estado CRSENT e coloca na porta Peer a interação CR.

Depois de especificado o módulo TRANSCODE, são realizadas as instanciações dos módulos, que correspondem às criações de variáveis do tipo módulo e de conexões das portas internas e externas ao refinamento. A cláusula **connect** liga a porta **Peer** da instância S (do tipo STATION) à porta **Up** da instância XC (do tipo TRANSCODE). A cláusula **replace** faz as substituições da porta **Up** (do módulo ABRA) pela porta **User** (da instância S) e da porta **Medium** (do módulo ABRA) pela porta **Down** (da instância XC).

5. O COMPILADOR Estelle/83 [8]

O compilador Estelle/83 é uma ferramenta que foi desenvolvida pelo Grupo de Redes de Computadores da Universidade Federal da Paraíba, Campus II-Campina Grande(PB), com o objetivo de viabilizar implementações semi-automáticas de sistemas especificados formalmente na TDF Estelle, versão de 1983.

Esse compilador permite que uma especificação seja fragmentada em vários módulos fonte, podendo cada módulo ser compilado separadamente. Para cada fonte é produzido um segmento de código Pascal, sendo que o código objeto é gerado pelo compilador Pascal. Ao conjunto de códigos objeto, referentes aos fontes Estelle/83, devem ser aglutinados os códigos objeto relativos a uma biblioteca (rotinas de suporte e núcleo), o que produzirá um programa executável (Fig. 5.1).

A Fig. 5.2 mostra a estrutura do compilador Estelle/83. Nela se pode observar que o compilador gera, como um passo intermediário, um programa na linguagem de alto nível Pascal. Esse programa é constituído de um conjunto de estruturas de dados e um conjunto de procedimentos, que refletem os conceitos próprios da TDF Estelle/83. Como a TDF Estelle é baseada em Pascal, naturalmente essa linguagem foi adotada para implementação. Entretanto, há outras arquiteturas de compiladores desenvolvidos com o mesmo objetivo do compilador Estelle/83, em que a linguagem de alto nível gerada pelo pré-processador não é Pascal.

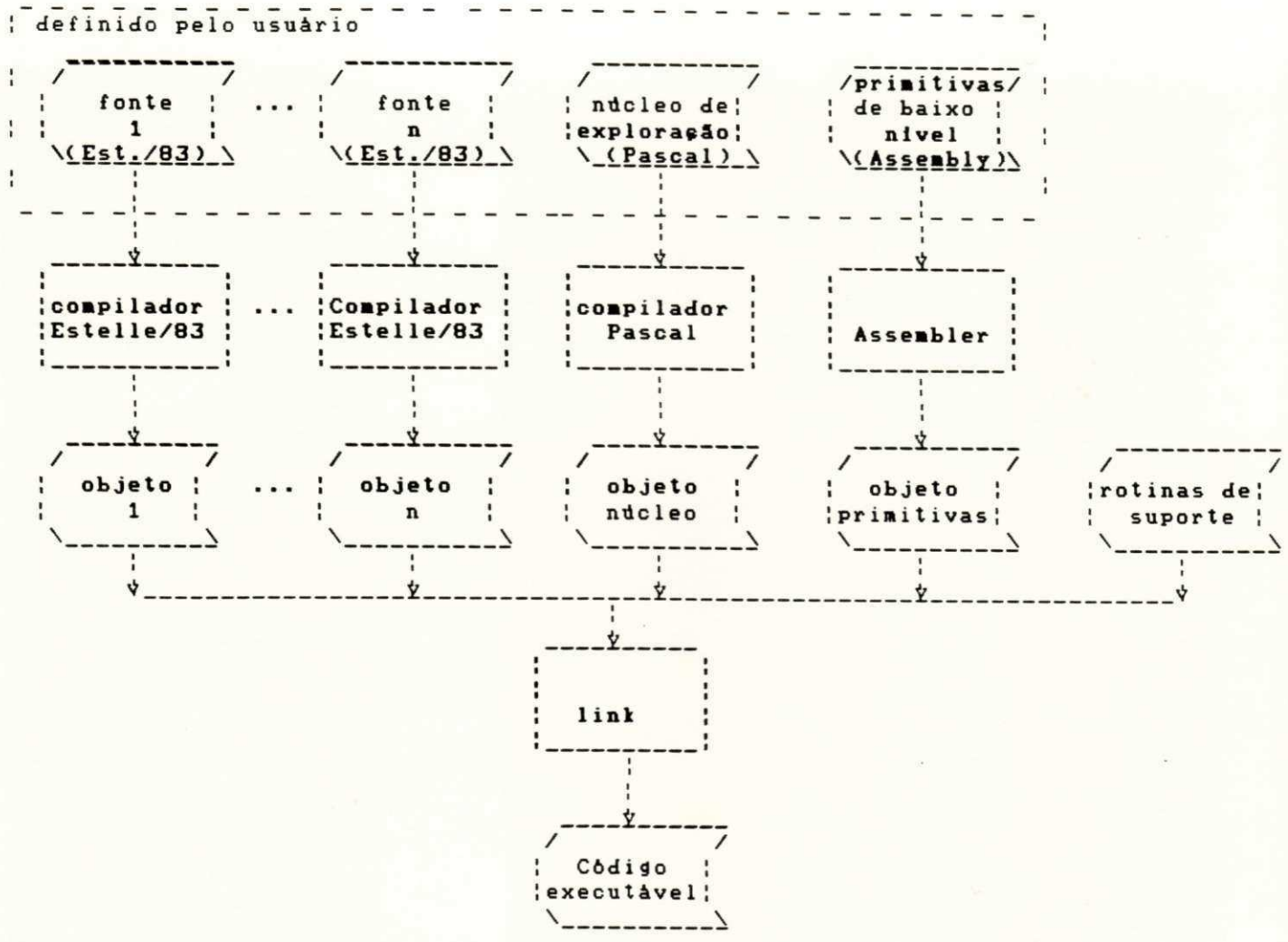


Fig. 5.1 Etapas e procedimentos necessários paras se alcançar um código executável

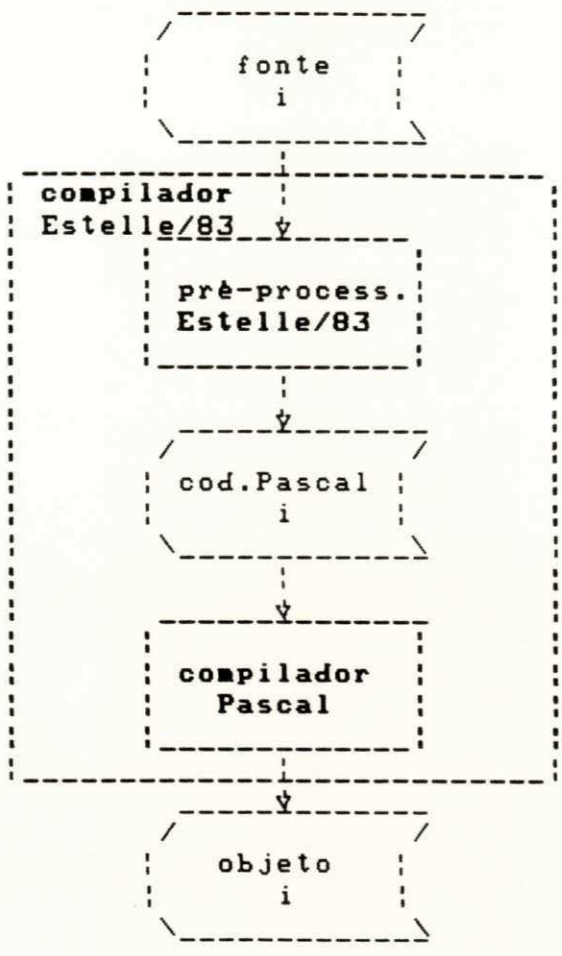


Figura 5.2: Estrutura do compilador Estelle/83.

A compilação completa da especificação compreende as seguintes fases:

(a) análises léxica, sintática e semântica estática;

(b) geração de código.

As análises léxica, sintática e semântica estática verificam, respectivamente, os erros léxicos, sintáticos e de contexto presentes na especificação. Nessa fase, quando ocorre um erro de qualquer tipo, o compilador para de executar e emite uma mensagem para o usuário. O usuário deve então corrigir o erro na

especificação fonte e submetê-la novamente ao compilador. Sempre que um erro for detectado, todo o procedimento de compilação é reiniciado.

A fase de geração de código transforma a representação interna, gerada durante a atividade de análise, numa estrutura próxima à linguagem Pascal. Em seguida, é feita a impressão do código, que é um segmento de programa Pascal,livre de erros léxicos e sintáticos.

Esse segmento de programa Pascal pode ser dividido em:

(a) declarações: constantes e tipos próprios à especificação; registros que representarão os processos, portas (tipo canal) e interações; variáveis para o armazenamento dos endereços do processo, porta e interação que estão sendo tratados; cabeçalhos das funções e procedimentos a serem importados (externos);

(b) procedimentos relativos aos processos contidos na especificação;

(c) procedimentos que inicializam as estruturas de dados criadas pelo compilador.

5.1 Estruturas de dados geradas pelo pré-processador Estelle/83[9]

A estrutura de dados para um processo é composta de

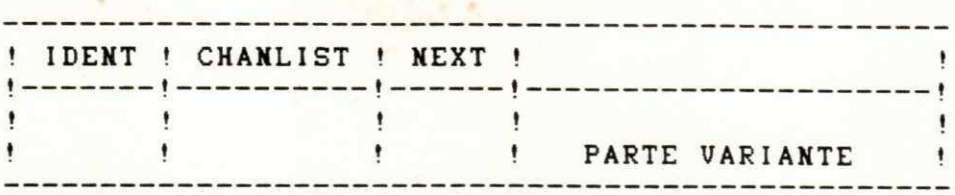


Fig. 5.3 Estrutura de dados de um processo

onde:

(a) o campo **IDENT** (do tipo sequência de caracteres) identifica o nome do processo;

(b) o campo **CHANLIST** (do tipo ponteiro) guarda o endereço da estrutura de dados (do tipo canal) da primeira porta a ser investigada, quando da execução do processo;

(c) o campo **NEXT** (do tipo ponteiro) guarda o endereço do próximo processo a ser investigado. Dessa forma, uma lista encadeada circular interliga todos os processos correspondentes às instâncias dos módulos;

(d) a **PARTE VARIANTE** armazena as variáveis locais ao processo.

O compilador gera um tipo **PROCESS = (POident_processo, POident_processo, ...)**, onde são enumerados todos os processos especificados. Isso pode ser visto no código Pascal (anexo 5) relativo à especificação do protocolo Abracadabra, através da seguinte construção:

PROCESS = (POCMS_PROC, POUSER_PROC, POSTATION_PROC, POTRANSCODE_PROC).

Nesta definição estão declarados os identificadores dos

A variável POVAR guarda o endereço do processo que está sendo executado. A declaração da estrutura de um processo, em Pascal, é:

```
01   TYPE
02     P1TYPE = ->POTYPE;
03     POTYPE = RECORD
04                IDENT : P2TYPE;
05                CHANLIST : C1TYPE;
06                NEXT : P1TYPE;
07                CASE PROCESS OF
08                  P0ident_processo
09                    ( );
10                  P0ident_processo
11                    ( );
12                  ...
13              END;
```

Fig. 5.4 Código Pascal para a Estrutura de um Processo

Em relação ao protocolo exemplo, as linhas 08 a 12 da Fig. 5.4 correspondem a:

```
01        POCMS_PROC :
02          ( );
03        POUSER_PROC :
04          (DOUSER_PROC : RECORD
05             STATE : (IDLE,CALLING,CONNECTED);
06             X,PRIM : USERDTYP;
07             FINAL : BOOLEAN;
08          END);
09        POSTATION_PROC :
10          (DOSTATION_PROC : RECORD
11             STATE : (CLOSED,CRSENT,CRRECV,ESTAB,DRSENT);
12             SENDING : BOOLEAN;
13             SENDSEQ,RECVSEQ : SEQTYPE;
14             OLDSENDSEQ : SEQTYPE;
15             CRRETRANREMAINING : INTEGER;
16             DTRETRANREMAINING : INTEGER;
17             DRRETRANREMAINING : INTEGER;
18             OLDDATA : USERDTYP;
19             DTORAK : BOOLEAN;
20          END);
21        POTRANSCODE_PROC :
22          (DOTRANSCODE_PROC : RECORD
23             SDU : UNITDTYP;
24          END);
```

Fig. 5.5 Código Pascal das Estruturas dos Processos Abracadabra

Essas declarações mostram as definições das variáveis locais de cada processo. A linha 02 está vazia porque o processo CMS_PROC não possui variáveis locais.

A estrutura de dados de uma porta (tipo canal) é composta de

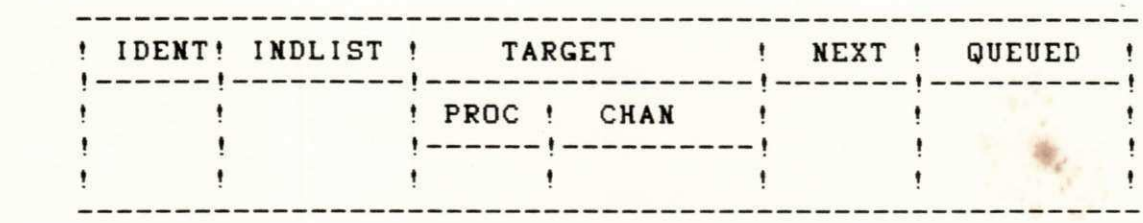


Fig. 5.6 Estrutura de dados de uma porta

onde:

(a) o campo **IDENT** (do tipo inteiro) identifica uma porta. Caso o processo contenha transições espontâneas, uma pseudo-porta (com IDENT = 0), criada pelo compilador, será a primeira a ser investigada;

(b) o campo **INDLIST** (do tipo ponteiro) percorre a lista de portas especificadas através do tipo array;

(c) o campo **TARGET** (do tipo registro) identifica o processo (**PROC**) e a porta (**CHAN**), ambos do tipo ponteiro, conectados à outra extremidade do canal;

(d) o campo **NEXT** (do tipo ponteiro) guarda o endereço da próxima porta a ser investigada. Dessa forma, uma lista encadeada permite percorrer todas as portas de uma mesma instância (processo);

(e) o campo **QUEUED** (do tipo booleano) indica se há (**true**) ou não (**false**) um fila associada à porta. No caso positivo, um campo **HEAD** (do tipo ponteiro) é acrescentado à essa estrutura para guardar o endereço do último elemento colocado na fila.

O compilador gera um tipo **CHANNEL = (C1ident_canal, ..., CNident_canal, C0ident_processo, ...)**, onde são enumerados todos os canais especificados. No código Pascal relativo ao exemplo, eesse tipo está assim definido:

CHANNEL = (C1MSAP,C2USAP,C3PEERCODE,C0USER_PROC,C0STATION_PROC).

A semântica dessa declaração é:

(a) a especificação possui três canais, cujos identificadores são MSAP, USAP e PEERCODE;

(b) somente os processos USER_PROC e STATION_PROC possuem transições espontâneas. Na definição desse tipo, os processos que têm transições espontâneas aparecem com seus identificadores precedidos dos caracteres **C0**.

A variável C0VAR guarda o endereço da porta (do tipo canal) que está sendo tratada. A declaração da estrutura de uma porta, em Pascal, é:

```
01   TYPE
02   P1TYPE = ->POTYPE;
03   C1TYPE = ->COTYPE;
04   S1TYPE = ->SOTYPE;
05   I1TYPE = ->IOTYPE;
06   IOTYPE = RECORD
07              NEXT : I1TYPE;
08              IDENT : BOOLEAN
09            END;
10   AOTYPE = RECORD
11              PROC : P1TYPE;
12              CHAN : C1TYPE
13            END;
14   COTYPE = RECORD
15              IDENT : INTEGER;
16              INDLIST : I1TYPE;
17              TARGET : AOTYPE;
18              NEXT : C1TYPE;
19              CASE QUEUED : BOOLEAN OF
20                FALSE :
21                  ( );
22                TRUE :
23                  (HEAD : S1TYPE)
24            END;
```

Fig. 5.7 Código Pascal para a Estrutura de uma Porta

A estrutura de dados de uma interação é composta de

Fig. 5.8 Estrutura de dados de uma interação

onde:

(a) o campo **NEXT** (do tipo ponteiro) é utilizado, no caso de comunicação através de filas, para construir uma lista encadeada de interações;

(b) na **PARTE VARIANTE** é declarado o corpo da interação. No caso de uma transição espontânea, uma pseudo-interação de entrada é declarada.

Todas as transições possíveis, inclusive as pseudo-interações, são declaradas, em Pascal, da seguinte forma:

```
S1ident_canal = (S1ident_interação, S1ident_interação, ...);
S2ident_canal = (S2ident_interação, S2ident_interação, ...);
...;
SNident_canal = (SNident_interação, SNident_interação, ...);
S0ident_processo = (R1ANY);
S0ident_processo = (R2ANY);
...;
S0ident_processo = (RNANY);
```

Fig. 5.9 Declaração em Pascal das interações

Os tipos **S1ident_canal, S2ident_canal, ..., SNident_canal** enumeram as interações declaradas na especificação dos respectivos canais. Os tipos **S0ident_processo** enumeram as pseudo-interações (**R1ANY, R2ANY, ...**), relativas às transições espontâneas dos respectivos processos. A declaração da estrutura de uma interação, em Pascal, é:

```
01 TYPE
02   S0TYPE = RECORD
03              CASE CHANNEL OF
04                C1ident_canal
05                  ( );
06                C2ident_canal
07                  ( );
08                ...;
09                CNident_canal
10                  ( );
11                C0ident_processo
12                  ( )
13            END;
```

Fig. 5.10 Declaração da Estrutura de uma Interação em Pascal

No código Pascal relativo exemplo, as declarações na Fig. 5.9 estão assim definidas:

```
01    S1MSAP = (S1UnitReq, S1UnitInd);
02    S2USAP = (S2ConReq, S2ConResp, S2DatReq, S2DisReq, S2ConInd,
03              S2ConConf, S2DatInd, S2DisInd);
04    S3PEERCODE = (S3CR, S3CC, S3DT, S3AK, S3DR, S3DC);
05    SOUSER_PROC = (R1ANY);
06    SOSTATION_PROC = (R2ANY);
```

Fig. 5.11 Declaração em Pascal das Interações no Protocolo Abracadabra

Nas linhas 01 a 04 estão declaradas as interações que podem ser trocadas pelos módulos conectados a um determinado canal. Por exemplo, na linha 04 os módulos ligados através do canal PEERCODE podem trocar as interações CR, CC, DT, AK, DR e DC.

Nas linhas 05 e 06 estão definidos os identificadores usados para tratar as pseudo-interações.

5.2 Interligação das estruturas de dados

As estruturas de dados que representam os processos, portas e interações, na especificação Estelle/83, são interligadas através de uma rede de ponteiros, construída por vários procedimentos Pascal, quando da execução do sistema. É com base nessa rede de ponteiros que é feito o gerenciamento de todo o sistema através de um núcleo, que escalona convenientemente os processos a serem executados.

Antes de acompanhar a montagem da rede de ponteiros relativa ao protocolo Abracadabra, é preciso fazer os seguintes esclarecimentos referente ao código Pascal gerado pelo compilador:

(a) As **IOPROCEDURES** são usadas para construir a interligação entre as várias estruturas;

(b) No corpo das **IOPROCEDURES** estão declarados vários procedimentos que compõem as ROTINAS DE SUPORTE, tais como POPROCEDURE, P1PROCEDURE etc, que são responsáveis, em último caso, pela geração passo a passo da rede de ponteiros;

(c) A cada cláusula **refinement** presente na especificação Estelle/83 corresponde uma IOPROCEDURE. O número de vezes em que essas IOPROCEDURES são referenciadas depende da instanciação do módulo refinado;

(d) A cada cláusula **process** na especificação Estelle/83 corresponde também uma IOPROCEDURE. Sempre que uma IOPROCEDURE, relativa a uma cláusula **process**, é referenciada, após sua execução são construídas e interconectadas as estruturas de dados que representam o respectivo processo e suas portas.

O corpo do núcleo deve ser composto basicamente de chamadas aos seguintes procedimentos:

(a) **INICIALIZAÇÃO**, que inicializa as variáveis globais declaradas nesse núcleo;

(b) **SYSTEM_INIT**, que interliga as estruturas de dados geradas pelo compilador, utilizando outros procedimentos;

(c) **SCHEDULER**, que determina a ordem de execução dos procedimentos relativos aos processos da especificação.

SYSTEM_INIT é composto de

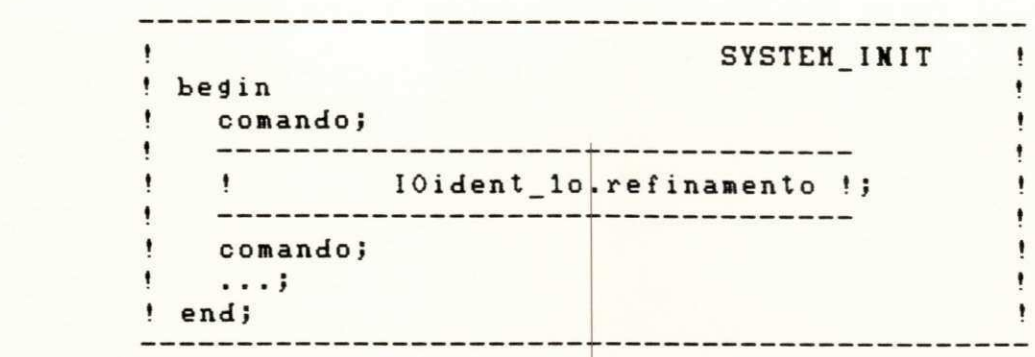


Fig. 5.12 Ações de SYSTEM_INIT

onde:

(a) **I0ident_1o.refinamento**, relativo ao primeiro refinamento realizado na especificação Estelle/83, inicia a interligação das estruturas de dados.

(b) Após a execução desse procedimento, essas estruturas estarão representadas por uma rede de ponteiros.

I0ident_1o.refinamento é composto de

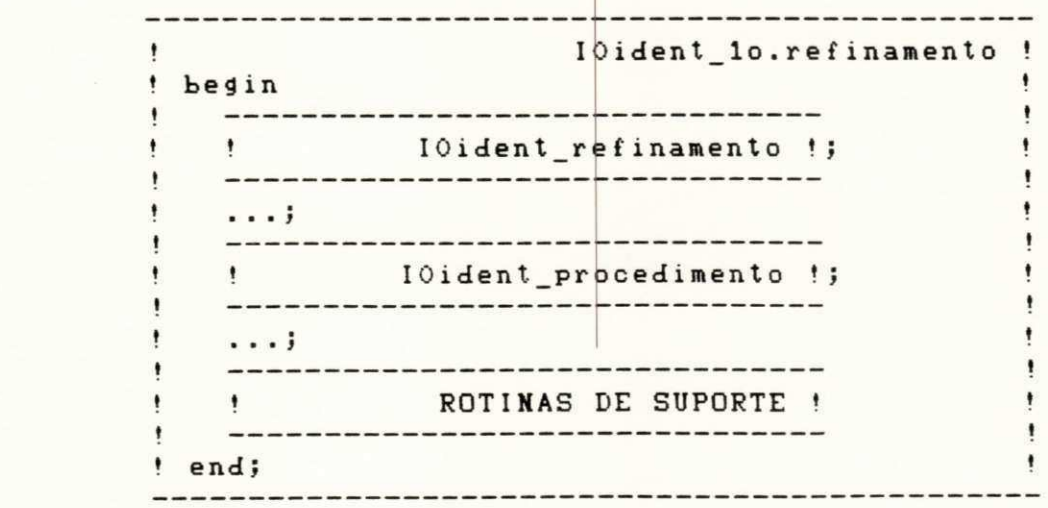


Fig. 5.13 Ações de I0ident_1o.refinamento

onde:

(a) os **IOident_refinamento** e os **IOident_procedimento**, relativos ao 1o nivel de refinamento da especificação Estelle/83, estão presentes no segmento de código Pascal gerado pelo compilador;

(b) para cada instância declarada nesse nível da especificação, existe uma chamada ao **IOident_refinamento** correspondente (instância de um refinamento) ou ao **IOident_procedimento** correspondente (instância de um processo).

As ações para os níveis intermediários de refinamento são semelhantes às apresentadas na Fig. 5.13. Para o último nível de refinamento, **IOident_refinamento** é composto de

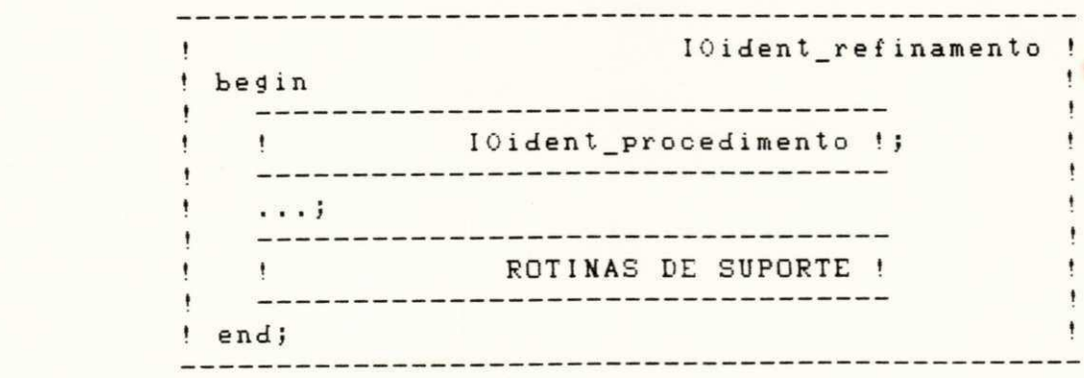


Fig. 5.14 Ações de IOident_refinamento

onde:

(a) os **IOident_procedimento**, relativos aos processos descritos no último nível de refinamento, estão presentes no segmento de código Pascal gerado pelo compilador;

(b) Para cada instância de um processo, declarada nesse nível da especificação, existe uma chamada ao procedimento **IOident_procedimento** correspondente.

As **ROTINAS DE SUPORTE** referem-se aos vários procedimentos que são responsáveis pelas interconexões entre as estruturas de dados dos processos e das portas. Por exemplo, para cada **IOident_procedimento** tem-se:

(a) **P1PROCEDURE**, que inicializa a estrutura de dados de um processo e guarda seu endereço na variável POVAR;

(b) **P3PROCEDURE**, que na primeira execução liga a estrutura de dados do processo à estrutura de dados de sua primeira porta. As outras execuções desse procedimento interligam as estruturas de dados das demais portas desse processo. Ao final de cada execução de P3PROCEDURE, a variável COVAR guardará o endereço da estrutura de dados da última porta interconectada.

A compreensão dessa rede de ponteiros[10], que permite navegar entre processos e portas, é muito importante para o estabelecimento de uma política de escalonamento, que determinará a execução dos procedimentos relativos às instâncias dos processos especificados.

5.3 Interligação das estruturas geradas para o protocolo Abracadabra

O compilador Estelle/83 traduz a cláusula refinement ABRA_SERVICE for ABRACADABRA no procedimento Pascal IOABRA_SERVICE. Esse procedimento representa o mais alto nível de abstração da especificação. O procedimento IOABRA_SERVICE é referenciado somente uma vez, porque só existe uma instância do módulo ABRACADABRA. Esse módulo, que representa o próprio sistema, é descrito através de dois processos (process CMS_PROC for CMS e process USER_PROC for USER) e de um segundo e último refinamento (refinement ABRA_BODY for ABRA). As cláusulas process geram os procedimentos IOCMS_PROC e IOUSER_PROC e a cláusula refinement gera IOABRA_BODY. Pode-se observar, no corpo de IOABRA_SERVICE, que o procedimento IOABRA_BODY é referenciado duas vezes. Isso porque o módulo ABRA é instanciado duas vezes.

De acordo com a arquitetura do protocolo Abracadabra, cada instância do módulo ABRA simula um lado do sistema. Por esse motivo o módulo ABRA é referenciado duas vezes, permitindo a construção e interligação das estruturas dos processos de ambos os lados.

A geração da rede de ponteiros se inicia através da referência (chamada para execução) ao procedimento IOABRA_SERVICE. O bloco de instruções executáveis desse procedimento é mostrado a seguir:

```
01   begin
02     p2var := nil;
03     IOABRA_BODY(p2var);
04     IOABRA_BODY(p2var);
05     IOCMS_PROC(p2var);
06     p7procedure(p2var,p1var);
07     p5procedure(3, 1, p1var);
08     p6procedure(a0var);
09     p5procedure(1, 1, p1var);
10     p6procedure(a1var);
11     a0var.chan->.target := a1var;
12     a1var.chan->.target := a0var;
13     p5procedure(2, 1, p1var);
14     p6procedure(a0var);
15     p5procedure(1, 3, p1var);
16     p6procedure(a1var);
17     a0var.chan->.target := a1var;
18     a1var.chan->.target := a0var;
19     p8procedure(p1var)
20   end;
```

Fig. 5.15 Código Pascal do Procedimento IOABRA_SERVICE

Nesse código as linhas 07 a 12 fazem a interconexão entre as portas de dois módulos, no caso os módulos TRANSCODE e CMS. Isso significa que a porta 2 do módulo TRANSCODE se liga à porta 1 do módulo CMS, de um lado do sistema. A interconexão do outro lado é feita nas linhas 13 a 18. Desse modo é implementado o conceito de canal da especificação.

Na linha 03 da Fig. 5.15 é feita a primeira referência ao procedimento IOABRA_BODY. O bloco de instruções executáveis desse procedimento é mostrado a seguir:

```
01    begin
02      p2var := nil;
03      IOTRANSCODE_PROC(p2var);
04      IOSTATION_PROC(p2var);
05      IOUSER_PROC(p2var);
06      p7procedure(p2var, p1var);
07      p3procedure(false, 1);
08      p5procedure(2, 2, p1var);
09      p6procedure(a0var);
10      p5procedure(3, 1, p1var);
11      p6procedure(a1var);
12      a0var.chan->.target := a1var;
13      a1var.chan->.target := a0var;
14      p5procedure(1, 1, p1var);
15      p6procedure(a0var);
16      p5procedure(2, 1, p1var);
17      p6procedure(a1var);
18      a0var.chan->.target := a1var;
19      a1var.chan->.target := a0var;
20      p5procedure(0, 1, p1var);
21      a0var.chan := c0var;
22      p5procedure(3, 2, p1var);
23      p6procedure(a0var.chan->.target);
24      p8procedure(p1var);
25    end;
```

Fig. 5.16 Código Pascal do Procedimento IOABRA_BODY

As primeiras estruturas começam a ser interligadas através da execução do procedimento IOTRANSCODE_PROC (linha 03). Esse procedimento tem o seguinte bloco de instruções executáveis:

```
01    begin
02      new(p0var, P0TRANSCODE_PROC);
03      p1procedure(TRANSCOD_P?, p1var);
04      p3procedure(true, 1);
05      p3procedure(true, 2);
06    end;
```

Fig. 5.17 Código Pascal do Procedimento IOTRANSCODE_PROC

Na linha 02 é criada a estrutura de dados que representa um processo. Nas linhas 04 e 05 são criadas as estruturas que representam as portas, que conjuntamente são interconectadas à estrutura do processo já criada. A existência do parâmetro true

nas linhas 04 e 05 significa que existem duas filas associadas às portas (uma para cada porta) onde são armazenadas as interações. Após a execução de IOTRANSCODE_PROC, as estruturas estão assim representadas:

Fig. 5.18 Interligação das Estruturas após a Execução de IOTRANSCODE_PROC

Depois de executado IOTRANSCODE_PROC, é referenciado o procedimento IOSTATION_PROC. O bloco de instruções executáveis desse procedimento é mostrado a seguir:

```
01   begin
02     new(p0var, POSTATION_PROC);
03     p1procedure(STATION_PR', p1var);
04     p3procedure(false, 0);
05     p3procedure(true, 1);
06     p3procedure(true, 2);
07     with POVAR->.DOSTATION_PROC do
08       begin
09         state := CLOSED
10       end
11   end;
```

Fig. 5.19 Código Pascal do Procedimento IOSTATION_PROC

Na linha 02 da Fig. 5.19 é criada a estrutura que representa o processo STATION_PROC, que é interligada à estrutura do processo TRANSCODE_PROC (criada anteriormente). Em relação à linha 04, observa-se que:

(a) existem transições espontâneas na especificação do processo, caracterizadas pela presença do parâmetro 0. No processo TRANSCODE_PROC (já analisado) não há esse parâmetro e, portanto, esse processo não possui transições espontâneas;

(b) está associada uma fila de tamanho zero à porta criada para tratar as transições espontâneas, significando que não são armazenadas interações nessa porta.

Na linha 09 é inicializada a variável state, que é usada para controlar a máquina de estados do processo STATION_PROC.

Após a execução de IOSTATION_PROC, as estruturas estão assim representadas:

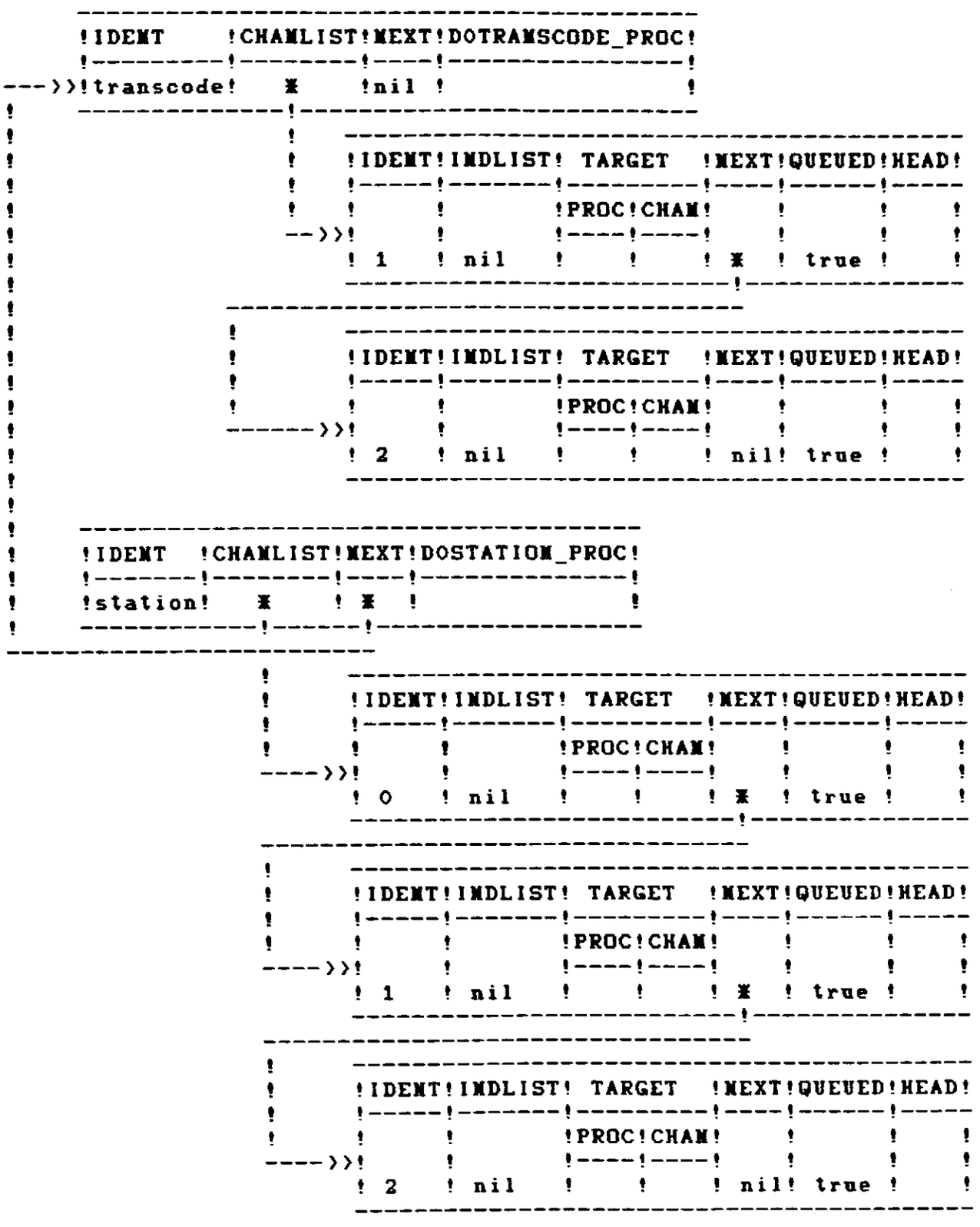


Fig. 5.20 Encadeamento das Estruturas após a Execução de IOSTATION_PROC

Na linha 05 de IOABRA_BODY é chamado para execução o procedimento IOUSER_PROC, cujo bloco de instruções executáveis é:

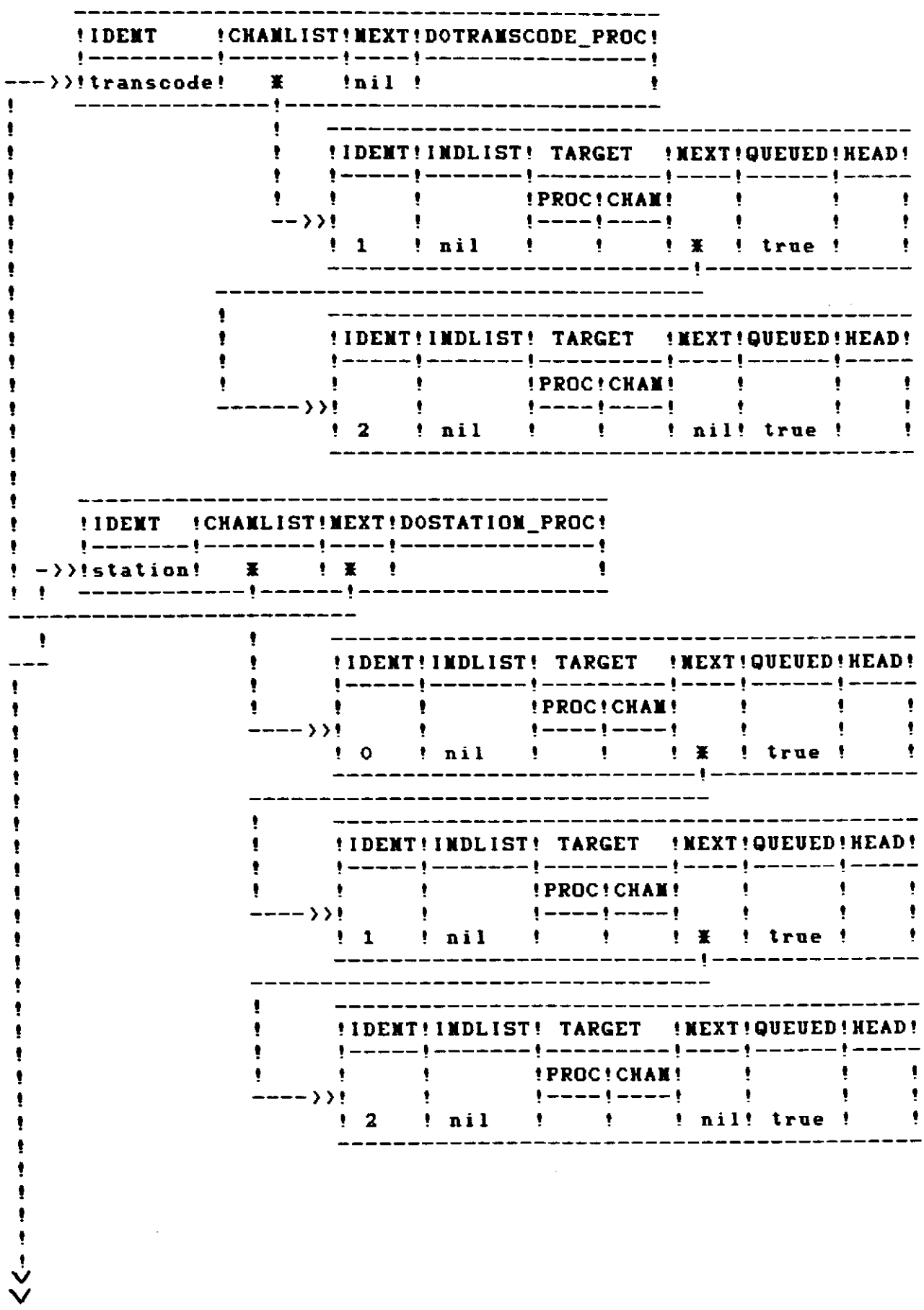
```
01   begin
02     new(p0var,p0USER_PROC);
03     p1procedure(USER_PROC ; p1var);
04     p3procedure(false, 0);
05     p3procedure(true, 1);
06     with p0var->.d0USER_PROC do
07       begin
08         state := IDLE
09       end
10   end;
```

Fig. 5.21 Código Pascal do Procedimento IOUSER_PROC

A exemplo dos procedimentos anteriores, na linha 02 é criada a estrutura de dados para o processo USER_PROC, que é interligada às estruturas já existentes. A linha 04 indica que o processo USER_PROC possui transições espontâneas, cujas interações são tratadas na porta 0 criada nessa linha. A linha 05 indica que o módulo USER possui somente uma porta identificada pelo número 1, onde são enfileiradas as interações.

Executados os procedimentos IOTRANSCODE_PROC, IOSTATION_PROC e IOUSER_PROC, as instruções restantes de IOABRA_BODY fazem as interconexões entre as portas internas dos módulos, concretizando o conceito de canal. Por exemplo, nas linhas 08 a 13 de IOABRA_BODY é feita a conexão entre a porta 2 do módulo STATION e a porta 1 do módulo TRANSCODE e nas linhas 14 a 19 é conectada a porta 1 do módulo USER à porta 1 do módulo STATION. Observando a arquitetura do sistema, verifica-se que a implementação está de acordo com o modelo especificado.

Depois de executado pela primeira vez o procedimento IOABRA_BODY, um lado do sistema já estará configurado.


IDENT
CHANLIST
NEXT
DOTRANSCODE_PROC
transcode
nil
IDENT
INDLIST
TARGET
NEXT
QUEUED
HEAD
PROC
CHAN
1
nil
true
2
nil
true
IDENT
CHANLIST
NEXT
DOSTATION_PROC
station
IDENT
INDLIST
TARGET
NEXT
QUEUED
HEAD
PROC
CHAN
0
nil
true
IDENT
INDLIST
TARGET
NEXT
QUEUED
HEAD
PROC
CHAN
1
nil
true
IDENT
INDLIST
TARGET
NEXT
QUEUED
HEAD
PROC
CHAN
2
nil
true

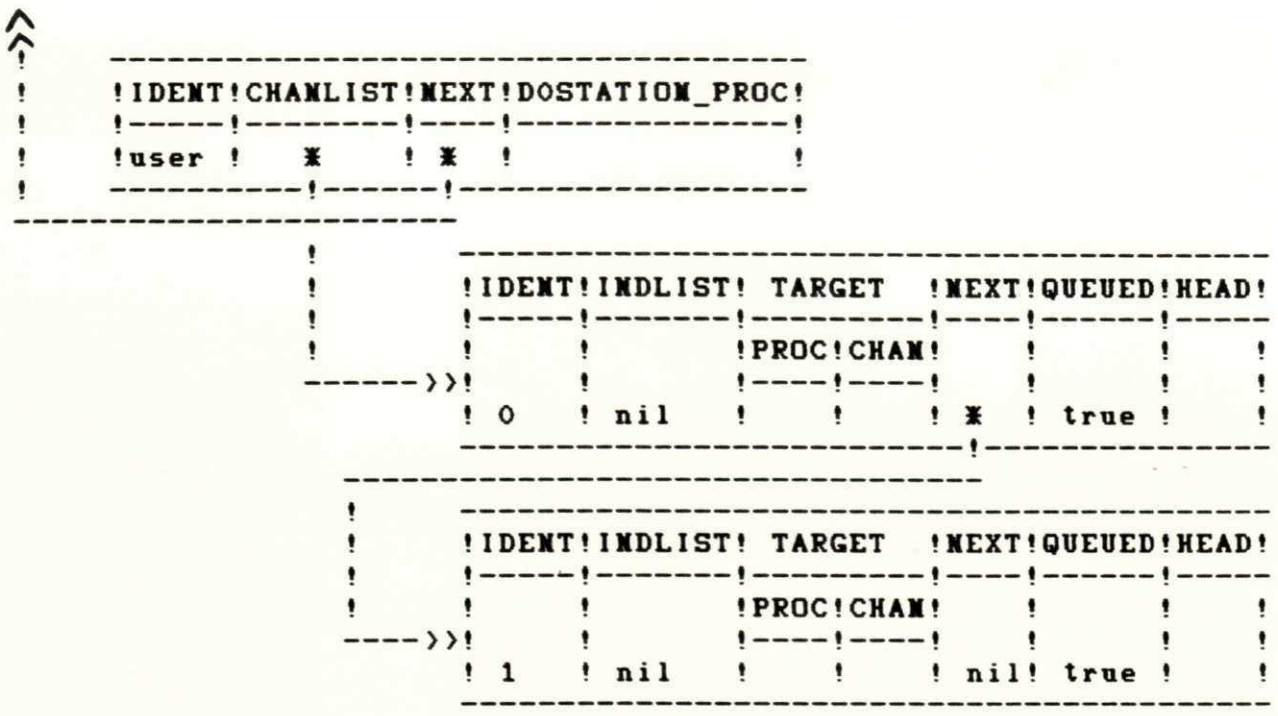


Fig. 5.22 Interligação das Estruturas após a Execução de IOUSER_PROC

IOABRA_SERVICE assume o controle e chama para execução IOABRA_BODY pela segunda vez, a fim de configurar o outro lado do sistema e interconectar suas estruturas às estruturas já existentes. Por último, é executado o procedimento IOCMS_PROC e o controle volta para IOABRA_SERVICE.

Depois de concluida a execução do procedimento IOABRA_SERVICE, o controle é passado para o núcleo e nesse instante a variável POVAR aponta para a estrutura de dados que representa o processo CMS_PROC. Através dessa variável é possivel navegar pela rede de ponteiros, permitindo que seja atingida qualquer uma das estruturas dos processos.

5.4 Tradução das transições Estelle/83 pelo compilador

O compilador Estelle/83 traduz as cláusulas Estelle/83 presentes na especificação das transições de um processo da seguinte forma:

(a) when corresponde a duas declarações if. A primeira verifica se o identificador da porta vigente é o mesmo identificador especificado na cláusula. A segunda verifica se a interação, que está sendo tratada, é a mesma que consta na cláusula. Caso a interação possua parâmetros, o comando with é usado para permitir o acesso aos mesmos;

(b) from corresponde a uma instrução if, que verifica se o estado principal vigente (state) é o mesmo da cláusula;

(c) to corresponde à atribuição do novo estado principal, especificado na cláusula, à variável state;

(d) provided corresponde a uma declaração if, acompanhada da mesma expressão booleana presente na cláusula.

A instrução out é implementada da seguinte forma:

(a) é chamada a POPROCEDURE, que faz parte das rotinas de suporte, para localizar a porta a ser tratada;

(b) é criada e inicializada a estrutura de dados da interação a ser emitida;

(c) é chamado o procedimento OUT, que faz parte do núcleo, para emitir a interação na porta adequada.

Para exemplificar, a seguir são mostradas duas transições relativas à especificação Estelle/83 do protocolo Abracadabra e os seus respectivos códigos Pascal gerados pelo compilador.

```
when User.ConReq
from CLOSED
to CRSENT
begin
  INITVAR;
  out Peer.CR;
  CRretranremaining := M-1
end;
```

```
if c0var->.ident = 1 then
  if s0var->.t2usap = s2ConReq then
    if state in (.CLOSED.) then
      begin
        state := CRSENT;
        INITVAR;
        begin
          p0procedure(2);
          new(s0var,c3peercode,s3CR);
          s0var->.t3peercode := s3CR;
          OUT
        end;
        CRretranremaining := M-1;
        goto 1
      end;
```

Fig. 5.23 Tradução de uma Transição Estelle/83 em Pascal

A cláusula when User.ConReq é traduzida em:

```
if c0var->.ident = 1 then
    if s0var->.t2usap = s2ConReq then.
```

A cláusula from CLOSED é traduzida em if state in (.CLOSED.) e a cláusula to CRSENT em state := CRSENT.

A cláusula out Peer.CR é traduzida em:

```
        begin
          p0procedure(2);
          new(s0var,c3peercode,s3CR);
          s0var->.t3peercode := s3CR;
          OUT
        end;
```

onde:

(a) p0procedure(2) leva a variável COVAR a apontar para a estrutura de dados que representa a segunda porta do módulo STATION;

(b) new(s0var,c3peercode,s3CR) e s0var->.t3peercode:=s3CfR criam e inicializam a estrutura que representa a interação;

(c) o procedimento OUT coloca a interação recém criada na fila.

```
provided CRretranremaining > 0
from CRSENT
to CRSENT
begin
  CRretranremaining := CRretranremaining - 1;
  out Peer.CR
end;


if c0var->.ident = 0 then
  if s0var->.t0station_proc = r2any then
    if state in (.CRSENT.) then
      begin
        state := CRSENT;
        CRretranremaining := CRretranremaining - 1;
        begin
          p0procedure(2);
          new(s0var,c3peercode,s3CR);
          s0var->.t3peercode := s3CR;
          OUT
        end;
        goto 1
      end;
```

Fig. 5.24 Tradução de uma Transição Espontânea Estelle/83 em Pascal

Os dois primeiros ifs verificam se há uma transição espontânea, testando se o identificador da porta é 0 e se a variável s0var está com o valor r2any.

6. METODOLOGIA PARA VALIDAÇÃO, ATRAVES DE SIMULAÇÃO, DE ESPECIFICAÇOES FORMAIS DE PROTOCOLOS DE COMUNICAÇÃO

O uso de uma TDF na fase de especificação facilita bastante o trabalho posterior de validação do design de um sistema, principalmente em se tratando de protocolo de comunicação.

A metodologia que foi empregada para validar um protocolo através de simulação, usando o compilador Estelle/83 como ferramenta, consiste em:

(a) especificar formalmente o serviço e o protocolo na TDF Estelle/83;

(b) submeter as especificações do serviço e do protocolo ao compilador Estelle/83, até que sejam gerados os respectivos segmentos de código Pascal livres de erros léxicos e sintáticos;

(c) executar a especificação do serviço, ligando os códigos relativos ao segmento Pascal (gerado pelo compilador Estelle/83), núcleo e rotinas de suporte. Nessa execução são observados os traços[11], que são sequências de interações emitidas ou recebidas pelos processos envolvidos;

(d) executar a especificação do protocolo, também ligando os códigos do núcleo, rotinas de suporte e do segmento gerado pelo compilador Estelle/83. É importante salientar que o código relativo ao núcleo

precisa ser adaptado às construções do serviço ou do protocolo quando da simulação.

(e) finalmente é examinada a equivalência entre os traços obtidos na simulação do serviço e aqueles obtidos na simulação do protocolo, abstraindo-se as interações dos traços do protocolo que não são visíveis a um observador externo.

6.1 Núcleo de simulação

O núcleo de simulação foi desenvolvido com o objetivo de permitir ao usuário interagir com o sistema, estimulando-o através de mensagens via terminal. Na fase de estabelecimento de conexão, o usuário digita os nomes das primitivas. Na fase de transferência de dados o usuário digita as mensagens diretamente.

O usuário escolhe, a todo instante, o processo que deseja executar. Escolhido um processo, o núcleo examina nas portas desse processo se há alguma interação na fila esperando tratamento:

(a) se houver interação pendente, o núcleo passa o controle para o procedimento Pascal correspondente ao processo. São examinadas as transições sequencialmente, na ordem em que foram descritas na especificação, até que seja encontrada uma transição habilitada:

(a1) caso positivo a interação é tratada, a variável de estado é atualizada e o controle é devolvido ao núcleo. Este volta, então, a solicitar ao usuário a escolha de outro processo para execução;

(a2) caso contrário a interação é mantida na fila e o controle é devolvido para o núcleo.

(b) se não houver interação pendente, então:

(b1) se o processo escolhido não possuir transição espontânea, o controle continua com o núcleo;

(b2) se o processo tiver transição espontânea, é criado um pseudo-sinal (R1ANY,R2ANY etc) para habilitar uma transição espontânea e o controle é passado para o procedimento Pascal relativo a esse processo.

Foi descrito, em linhas gerais, a estratégia adotada por ocasião do desenvolvimento do núcleo de simulação.

A estrutura do núcleo de simulação é composta dos seguintes procedimentos:

(a) **WRITE_DATA** - registra as mensagens recebidas por cada um dos lados do sistema;

(b) **GETPRIM** - recebe a entrada via terminal do nome das primitivas digitadas pelo condutor da simulação;

(c) **ENCERRA_SISTEMA** - atribui valores às variáveis que são usadas para testar condição de parada da execução do sistema;

(d) **GETCH** - recebe as mensagens propriamente ditas que são trocadas entre os lados do sistema e que são entradas via terminal pelo usuário, simulando cada um dos lados;

(e) **SETTIMER** - aciona temporizador nos lados do sistema;

(f) **RESETTIMER** - desliga temporizador nos lados do sistema;

(g) **TIMEOUT** - testa condição de timeout para as mensagens que foram enviadas;

(h) **INICIALIZAÇÃO** - inicializa as variáveis de controle do núcleo;

(i) **OUT** - coloca as interações geradas pelas transições dos processos nas filas que estão associadas a cada uma das portas. Esse procedimento é chamado sempre que uma transição habilitada tem uma interação para ser enviada a outro processo;

(j) **SYSTEM_INIT** - ativa a geração da rede de ponteiros que interliga as estruturas de dados do sistema;

(k) **SCHEDULER** - gerencia todo o sistema e possui o controle de execução, permitindo que outros procedimentos assumam esse controle e devolvam-no depois de executados (o controle de execução sempre retorna para o núcleo). Esse

procedimento é responsável pelo escalonamento dos processos e pela recuperação das interações que estão nas filas aguardando tratamento.

Os procedimentos SETTIMER, RESETTIMER e TIMEOUT são usados para simular a execução da cláusula delay de Estelle, que não está implementada no compilador Estelle/83.

O bloco de instruções executáveis do núcleo é o seguinte:

```
01 begin
02    termin(input);
03    termout(output);
04    inicialização;
05    system_init:
06    scheduler
07 end.
```

Fig. 6.1 Código Pascal das Instruções Executáveis do Núcleo

As instruções nas linhas 02 e 03 são usadas para receber a entrada via terminal e direcionar a saída para o vídeo. Nas demais linhas são chamados os procedimentos INICIALIZAÇAO, SYSTEM_INIT e SCHEDULER (já explicados anteriormente), sendo que esse último é executado até que sejam satisfeitas as condições de encerramento do sistema.

O **SCHEDULER** é composto das ações alternativas

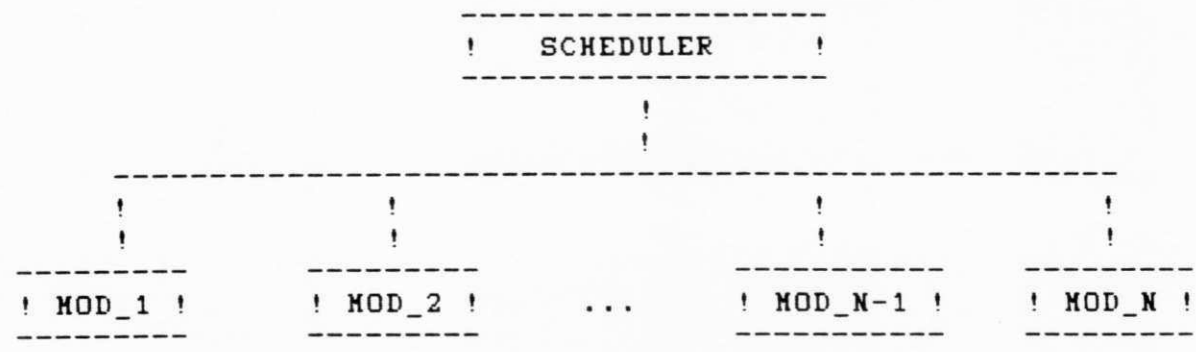


Fig. 6.2 Ações do SCHEDULER

onde:

(a) cada **MOD_I** (I = 1..N) representa um tipo processo da especificação.

MOD_I é composto de

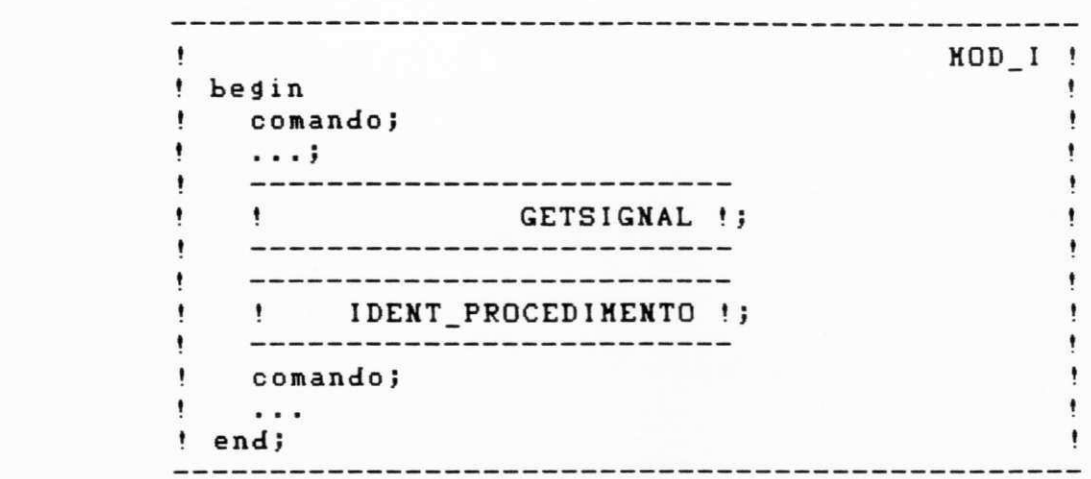


Fig. 6.3 Ações de MOD_I

onde:

(a) **GETSIGNAL** retira a interação da fila da estrutura de dados da porta vigente e guarda seu endereço na variável **SOVAR**;

(b) **IDENT_PROCEDIMENTO** reflete as transições do processo vigente e as chamadas ao procedimento **OUT**. Esse procedimento coloca a interação emitida na fila da estrutura de dados da porta de um outro processo, conectada à porta vigente.

Uma vez iniciada a simulação, é exibida uma tela com os nomes das instâncias dos processos da especificação. Cabe ao usuário conduzir a simulação escolhendo, a cada passo, uma instância para execução. A cada escolha do usuário é ativado o respectivo procedimento.

6.2 Simulação das especificações formais do serviço e protocolo Abracadabra

Na simulação do serviço Abracadabra, após a execução do procedimento SYSTEM_INIT, o sistema Abraservice (correspondente à Fig. 4.2) pode ser esquematizado:

Fig. 6.4 Processos do sistema Abraservice

Na simulação do protocolo Abracadabra, após a execução de SYSTEM_INIT, o sistema Abraservice' (Fig. 4.3) pode ser esquematizado:

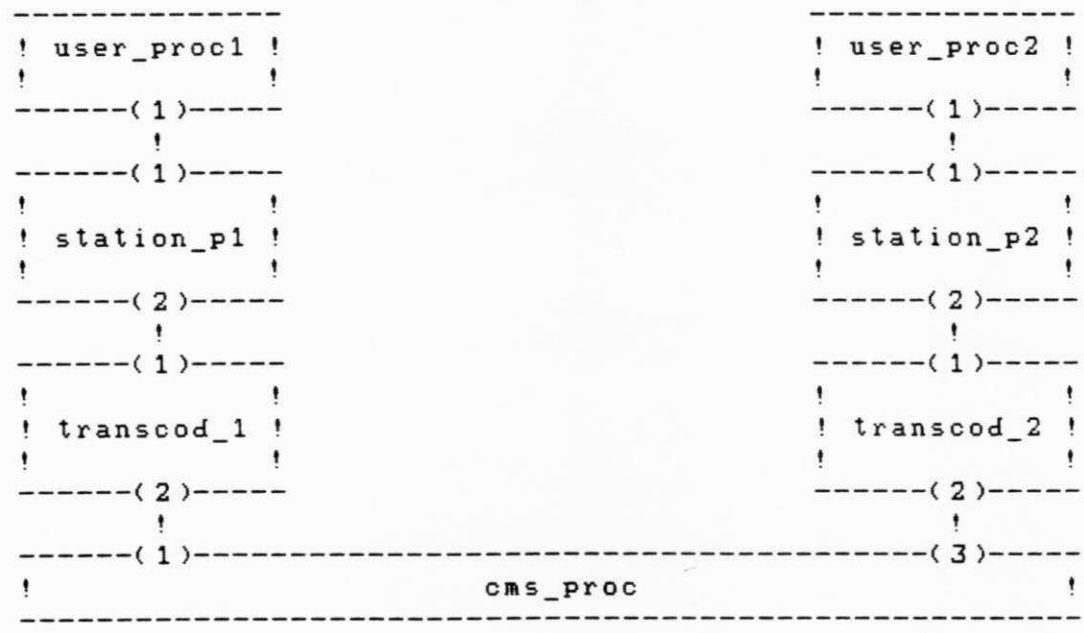


Fig. 6.5 Processos do sistema Abraservice'

Para cada par de processos, por exemplo STATION_P1 e STATION_P2, é gerado um único código Pascal que simula a execução de um e de outro, ora se comportando como o lado 1 do sistema, ora como o lado 2.

6.3 Simulação do sistema Abraservice'

Logo que o procedimento SCHEDULER assume o controle, tem início a simulação propriamente dita. A tela mostrada na Fig. 6.6 é exibida para o condutor da simulação, a fim de que seja escolhido qualquer um dos processos para execução.

```
**********************************
SIMULAÇAO DO PROTOCOLO ABRACADABRA
**********************************
 ** OS NOMES DOS PROCESSOS SAO **
- user_proc1;          - user_proc2;
- station_p1;          - station_p2;
- transcod_1;          - transcod_2;
                - cms_proc
** DIGITE O NOME DO PROCESSO A SER EXECUTADO **
```

Fig. 6.6 Tela para escolhe do processo a ser executado

Caso o condutor digite um nome de processo diferente daqueles mostrados na Fig. 6.6, a seguinte mensagem é exibida no vídeo:

```
** NOME DO PROCESSO EMITIDO INCORRETAMENTE **
** TENTE NOVAMENTE **
```

Depois de digitado um nome de processo válido, é chamado para execução um procedimento que verifica se há alguma interação pendente nas filas associadas a cada uma das portas dos processos. Somente uma interação é tratada por vez. Em primeiro lugar é investigada a fila associada a uma das portas e depois, numa outra execução, é investigada a outra fila. É importante observar que existe prioridade na investigação das filas e somente quando aquela de mais alta prioridade estiver vazia é que a outra fila será investigada.

Para cada par de processos existe um procedimento dentro do SCHEDULER que investiga as filas associadas às portas dos processos:

(a) **MOD_1** - para user_proc1 e user_proc2;

(b) **MOD_2** - para station_p1 e station_p2;

(c) **MOD_3** - para transcod_1 e transcod_2;

(d) **MOD_4** - para cms_proc.

O procedimento GETSIGNAL, do SCHEDULER, retira a interação da fila para que possa ser tratada.

Sempre que um processo é escolhido para execução e não possui interação em suas portas, a seguinte mensagem é exibida para o usuário:

** NAO HA MENSAGENS NAS PORTAS **.

Em seguida a tela da Fig. 6.6 novamente é mostrada ao usuário para que ele possa escolher o nome de outro processo.

No início da simulação, como não há interações transitando no sistema, somente os processos user_proc1 e user_proc2 poderão ser executados, através da criação de um pseudo-sinal que habilitará uma transição espontânea. A habilitação dessa transição permitirá que o usuário entre com as primitivas de estabelecimento de conexão definidas no protocolo Abracadabra.

Para exemplificar, alguns traços relativos ao Abraservice; acompanhados das execuções dos processos de de suas transições, são analisados.

É assumido que o usuário digitou user_proc1. O procedimento MOD_1, cujo código é mostrado abaixo, é chamado para execução:

```
01   procedure MOD_1
02   begin
03     c0var := p0var->.chanlist;
04     c0var := c0var->.next;
05     if c0var->.head = nil then
06       begin
07         c0var := p0var->.chanlist;
08         new( s0var, c0USER_PROC,R1ANY );
09         s0var->.t0USER_PROC := R1ANY;
10         USER_PROC
11       end
12     else
13       begin
14         GETSIGNAL;
15         USER_PROC
16       end
17   end;
```

Fig. 6.7 Código Pascal do Procedimento MOD_1

Nas linhas 03 e 04 a variável c0var vai apontar para a estrutura de dados que representa a porta 1 do processo user_proc1. Na linha 05 é realizado o teste para saber se há alguma interação na fila. Como não há (c0var->.head = nil), as

instruções nas linhas 07 a 10 são executadas. Se houvesse interações pendentes (c0var->.head <> nil), as instruções nas linhas 14 e 15 é que seriam executadas. Depois de criada e inicializada a estrutura de dados do pseudo-sinal que habilitará uma transição espontânea (linhas 08 e 09), o procedimento USER_PROC, que faz parte do segmento de código Pascal gerado pelo compilador Estelle/83, é chamado para execução. Examinando o código desse procedimento, a transição que ficará habilitada será:

```
01 if c0var->.ident = 0 then
02   if s0var->.t0USER_PROC = R1ANY then
03     if state in (.IDLE.) then
04       begin
05         state := CALLING;
06         begin
07           p0procedure(1);
08           new(s0var,c2USAP,s2ConReq);
09           s0var->.t2USAP := s2ConReq;
10           OUT
11         end;
12         goto 1
13       end;
```

Fig. 6.8 Código Pascal da Transição Espontânea Inicialmente Habilitada no Procedimento USER_PROC

A máquina de estados do processo USER_PROC assume o estado CALLING (linha 05) e a variável C0VAR fica apontando para a estrutura de dados da porta 1 desse processo (linha 07). Em seguida, é criada e inicializada a estrutura de dados para tratar a interação ConReq que será emitida. O procedimento OUT (linha 10) localiza a porta que está conectada à porta 1 do processo USER_PROC e coloca a interação na fila. Isso pode ser acompanhado examinando-se o bloco de instruções executáveis desse procedimento.

```
01 begin
02   with c0var->.target.chan-> do
03     if QUEUED then
04       begin
05         if head = nil then
06           begin
07             head := s0var;
08             s0var->.next := nil
09           end
10         else
11           begin
12             s := head;
13             head := s0var;
14             s0var->.next := s
15           end
16       end
17     else
18       RENDEZVOUS
19 end;
```

Fig. 6.9 Código Pascal do Procedimento OUT

Na linha 02, é localizada a estrutura de dados da porta, relativa ao outro processo (no caso STATION_PROC), que está ligada à porta 1 do processo USER_PROC. Na linha 03 é realizado o teste que indicará a disciplina de comunicação. Caso seja através de filas FIFO, na linha 05 é realizado o teste que indicará se a fila está vazia. Em caso positivo, a interação será armazenada (linhas 07 e 08). Em caso contrário, a interação é colocada no final da fila (linhas 12 a 14). Se houver alguma porta com fila associada de tamanho zero (comunicação síncrona), então é chamado o procedimento RENDEZVOUS para tratar imediatamente a interação.

Nesse ponto da simulação, a interação ConReq está armazenada na fila associada à porta 1 do processo STATION_PROC, simulando o lado 1.

O núcleo assume o controle e exibe novamente a tela para que o usuário escolha um processo para ser executado. Escolhendo

station_p1, o controle é passado para o procedimento MOD_2 que irá investigar nas portas 1 e 2 do módulo STATION se existe interação aguardando tratamento.

Primeiramente é observada a porta 2 (de maior prioridade), e, caso haja interação pendente, é chamado o procedimento GETSIGNAL para retirar a interação da fila e passá-la para o procedimento STATION_PROC. Nesse procedimento são examinadas sequencialmente as transições, até que seja encontrada uma que satisfaça as condições que irão habilitá-la para execução.

Como não existe interação na porta 2, a porta 1 é investigada, sendo constatada a presença de uma interação pendente (c0var->.head <> nil).

O procedimento GETSIGNAL é chamado para execução e a interação é recuperada da fila. Em seguida, o controle é passado para o procedimento STATION_PROC, que procurará uma transição habilitada.

```
01 if c0var->.ident = 1 then
02   if s0var->t2USAP = s2ConReq then
03     if state in (.CLOSED.) then
04       begin
05         state := CRSENT;
06         INITVAR;
07         begin
08           p0procedure(2);
09           new(s0var,c3PEERCODE,s3CR);
10           s0var->.t3PEERCODE := s3CR;
11           OUT
12         end;
13         CRRetranRemaining := N-1;
14         goto 1
15       end;
```

Fig. 6.10 Código Pascal da Transição Habilitada no Procedimento STATION_PROC para Tratar a Primitiva ConReq

As condições nas linhas 01 a 03 são satisfeitas e o estado da máquina para o processo STATION_PROC, que está simulando o lado 1, assume o valor CRSENT. Na linha 08 a variável COVAR aponta para a estrutura de dados que representa a porta 2. Nas linhas 09 e 10 é criada e inicializada a estrutura para tratar a interação CR. O procedimento OUT localiza a estrutura da porta, relativa ao outro processo (no caso TRANSCODE), que está conectada à porta 2 de STATION. A interação é então armazenada na fila associada à porta 1 de TRANSCODE.

Se o usuário digitar transcod_1, o procedimento MOD_3 investigará as portas do módulo TRANSCODE e encontrará a interação CR. Em seguida, o procedimento GETSIGNAL é chamado para recuperar a interação e o procedimento TRANSCODE_PROC assume o controle. Então, as transições são investigadas até que uma transição habilitada seja encontrada.

```
01 if c0var->.ident = 1 then
02   if s0var->.t3PEERCODE = s3CR then
03     begin
04       BuildCR(SDU);
05       begin
06         p0procedure(2);
07         new(s0var, c1MSAP, s1UnitReq);
08         s0var->.t1MSAP := s1UnitReq;
09         s0var->.d1UnitReq.UnitData := SDU;
10         OUT
11       end;
12       goto 1
13     end;
```

Fig. 6.11 Código Pascal da Transição Habilitada no Procedimento TRANSCODE_PROC para Tratar a UDP CR

Nessa transição observa-se que não existe a variável state e por conseguinte não há necessidade das cláusulas de Estelle **from** e **to**. Isso porque o módulo TRANSCODE tem somente um estado. As duas primeiras linhas testam a porta e a interação a fim de habilitar (ou não) a transição. Na linha 04 o procedimento BuildCR(SDU) é chamado para realizar o empacotamento da interação CR na SDU, que será enviada para o meio (módulo CMS).

```
01 procedure BuidCR(va SDU : UnitDTyp);
02 begin
03   with p0var->.d0TRANSCODE_PROC do
04     begin
05       SDU.PDU := CR
06     end
07 end;
```

Fig. 6.12 Código Pascal do Procedimento BuildCR

Depois de empacotada a CR, a estrutura para a primitiva UnitReq é criada e inicializada e o procedimento OUT é chamado para colocar a primitiva na porta 1 do módulo CMS.

A partir desse instante, somente o processo **cms_proc** possui interação pendente, aguardando tratamento.

E bom lembrar que se o condutor escolher um processo diferente de CMS_PROC e se esse processo não tiver mensagens nas portas e nem transições espontâneas, a mensagem **NAO HA MENSAGENS NAS PORTAS** aparecerá no vídeo. Se o processo escolhido tiver transições espontâneas e nenhuma estiver habilitada, a mensagem **O PROCESSO NAO TEM TRANSIÇAO HABILITADA** aparecerá no vídeo.

Escolhido CMS_PROC, o procedimento MOD_4 investigará as portas e encontrará a primitiva UnitReq na porta 1. O procedimento GETSIGNAL será executado para retirar a primitiva da fila e em seguida será executado o procedimento CMS_PROC.

```
01 if c0var->.ident = 1 then
02   if s0var->.t1MSAP = s1UnitReq then
03  /* if not PERDA_NO_MEIO then */
04       with s0var->.d1UnitReq do
05         begin
06           begin
07             p0procedure(3);
08             new(s0var, c1MSAP, s1UnitInd);
09             s0var->.d1UnitInd.UnitData := UnitData;
10             OUT
11           end;
12           goto 1
13         end;
```

Fig. 6.13 Código Pascal da Transição Habilitada no Procedimento USER_PROC para Tratar a Primitiva UnitReq

Nessa transição também não há necessidade das cláusulas **from** e to porque o módulo CMS só tem um estado. A função booleana PERDA_NO_MEIO é randômica e permite que seja simulado um meio com perdas. Caso o meio seja confiável, a linha 03 passa a ser um simples comentário.

Nas linhas 07 a 10 (Fig. 6.13), a primitiva UnitInd é armazenada na porta 2 do módulo TRANSCODE, depois de ter sua estrutura criada e inicializada. Começa então a ser simulado o lado 2 do sistema Abracadabra. Os mesmos códigos dos procedimentos USER_PROC, STATION_PROC, TRANSCOD_PROC e CMS_PROC serão usados para simular a execução dos processos user_proc2, station_p2, transcod_2 e cms_proc.

O condutor escolhe transcod_2 para execução. O procedimento MOD_3 assume o controle e investiga as portas, encontrando a primitiva UnitInd na porta 2. Os procedimentos GETSIGNAL e TRANSCODE_PROC são executados nessa sequência. Simulando o lado 2, a transição habilitada de TRANSCODE_PROC é a seguinte:

```
01 if c0var->.ident = 2 then
02   if s0var->.t1MSAP = s1UnitInd then
03     with s0var->.d1UnitInd do
04       if (UnitData.PDU = CR) then
05         begin
06           p0procedure(1);
07           new(s0var, c3PEERCODE, s3CR);
08           s0var->.t3PEERCODE := s3CR;
09           OUT;
10           goto 1
11         end;
```

Fig. 6.14 Transição Habilitada em TRANSCODE_PROC p/tratar UnitInd

Na linha 04, é testado o campo PDU da SDU recebida, para verificar se a UDP é uma CR. Em caso afirmativo, essa UDP é desempacotada. As linhas 06 a 09 criam e inicializam a estrutura da interação CR, colocando-a na fila associada à porta 2 do módulo STATION.

O nome do processo a ser escolhido para execução será **station_p2**. O procedimento MOD_2 investiga as portas e encontrando a interação CR na porta 2, retira-a da fila. O código da próxima transição habilitada é:

```
01 if c0var->.ident = 2 then
02   if s0var->.t3PEERCODE = s3CR then
03     if state in (.CLOSED.) then
04       begin
05         state := CRRECV;
06         begin
07           p0procedure(1);
08           new(s0var, c2USAP, s2ConInd);
09           s0var->.t2USAP := s2ConInd;
10           OUT
11         end;
12         goto 1
13       end;
```

Fig. 6.15 Transição Habilitada em STATION_PROC p/Tratar a UDP CR

O estado do processo STATION_PROC é CRSENT, para o lado 1, e CRRECV, para o lado 2. A primitiva ConInd é colocada na porta 1 do módulo USER (lado 2).

Nesse ponto da simulação o módulo USER pode aceitar o pedido de conexão ou rejeitá-lo. Para continuar com a análise será suposto que o condutor rejeitará o pedido de conexão.

Primeiramente, é escolhido o processo user_proc2 para execução. O procedimento MOD_1 é executado e localiza a primitiva ConInd na fila da porta 1, que é então recuperada pelo procedimento GETSIGNAL. Em seguida, o controle passa para USER_PROC, onde são testadas as transições. A seguinte mensagem aparecerá para o usuário:

USURIO, ENTRE COM O NOME DA PRIMITIVA

O usuário então digitará DisReq, rejeitando o pedido de conexão. Se resolvesse aceitar o pedido, a primitiva a ser digitada seria ConResp. Se for digitado algo diferente de DisReq e ConResp, o sistema emite a seguinte mensagem:

ERRO NO PREENCHIMENTO DA PRIMITIVA

TENTE NOVAMENTE

Depois de executado user_proc2, a primitiva DisReq estará armazenada na fila da porta 1 do módulo STATION.

Escolhendo station_p2 para execução, o procedimento MOD_2 detectará a primitiva DisReq na porta 1 e disparará a execução dos seguintes procedimentos: GETSIGNAL que irá recuperar a primitiva DisReq e STATION_PROC buscará a transição habilitada que fará o tratamento dessa primitiva.

```
01 if c0var->.ident = 1 then
02   if s0var->.t2USAP = s2DisReq then
03     if state in (.CRRECV.) then
04       begin
05         state := CLOSED;
06         begin
07           p0procedure(2);
08           new(s0var, c3PEERCODE, s3DR);
09           s0var->.t3PEERCODE := s3DR;
10           OUT
11         end;
12         goto 1
13       end;
```

Fig 6.16 Transição Habilitada em STATION_PROC p/Tratar DisReq

O estado da máquina para station_p2 será CLOSED (linha 05).

A interação DR será colocada na fila associada à porta 1 do módulo TRANSCODE.

Depois de digitado **transcod_2**, o procedimento MOD_3 constatará que a porta 1 está com mensagem e chamará GETSIGNAL para recuperá-la. O procedimento TRANSCODE_PROC será ativado e a transição habilitada será:

```
01 if c0var->.ident = 1 then
02   if s0var->.t3PEERCODE = s3DR then
03     begin
04       BuildDR(SDU);
05       begin
06         p0procedure(2);
07         new(s0var, c1MSAP, s1UnitReq);
08         s0var->.t1MSAP := s1UnitReq;
09         s0var->.d1UnitReq.UnitData := SDU;
10         OUT
11       end;
12       goto 1
13     end;
```

Fig. 6.17 Transição Habilitada em TRANSCODE_PROC p/Tratar DR

Nessa transição a SDU contendo a interação DR será empacotada e colocada na fila associada à porta 3 do módulo CMS.

O processo CMS_PROC será executado para retirar a mensagem da porta 3 (primitiva UnitReq) e colocá-la na porta 2 do módulo TRANSCODE, no lado 1. Assim, a SDU contendo a interação será acomodada na primitiva UnitInd, que será passada para o módulo TRANSCODE.

Executando o processo **transcod_1**, a primitiva UnitInd será retirada da fila na porta 2 e a interação será desempacotada.

```
01 if c0var->.ident = 2 then
02   if s0var->.t1MSAP = s1UnitInd then
03     with s0var->.d1UnitInd do
04       if (UnitData.PDU = DR) then
05         begin
06           p0procedure(1);
07           new(s0var, c3PEERCODE, s3DR);
08           s0var->.t3PEERCODE := s3DR;
09           OUT;
10           goto 1
11         end;
```

Fig. 6.18 Transição Habilitada em TRANSCODE_PROC p/Tratar UnitInd

Depois de executada essa transição, a interação DR se encontrará na fila da porta 2 do módulo STATION, lado 1.

Station_p1 será executado e a transição habilitada será:

```
01 if c0var->.ident = 2 then
02   if s0var->.t3PEERCODE = s3DR then
03     if state in (.CRSENT.) then
04       begin
05         state := CLOSED;
06         begin
07           p0procedure(1);
08           new(s0var, c2USAP, s2DisInd);
09           s0var->.t2USAP := s2DisInd;
10           OUT
11         end;
12         CRRetranRemaining := -1;
13         goto 1
14       end;
```

Fig. 6.19 Transição Habilitada em STATION_PROC p/Tratar UDP DR

Executada essa transição, o estado da máquina para STATION_PROC ser' CLOSED e a primitiva DisInd será colocada na porta 1 do módulo USER.

O condutor escolherá para execução o processo user_proc1 e a primitiva DisInd será retirada da fila. O estado da máquina de USER_PROC será IDLE e o sistema voltará ao estado inicial, pronto para recomeçar todas as fases previstas no protocolo.

Através da análise do fluxo das interações ora realizada, foram percorridas as fases de estabelecimento de conexão e desconexão do protocolo Abracadabra.

A partir desse instante é assumido que o lado 2 aceitará o pedido de conexão. O usuário então deverá digitar o nome da primitiva ConResp, que será colocada na fila associada à porta 1 do módulo STATION, lado 2.

Será executado station_p2 e a transição habilitada será:

```
01 if c0var->.ident = 1 then
02   if s0var->.t2USAP = s2ConResp then
03     if state in (.CRRECV.) then
04       begin
05         state := ESTAB;
06         begin
07           p0procedure(2);
08           new(s0var, c3PEERCODE, s3CC);
09           s0var->.t3PEERCODE := s3CC;
10           OUT
11         end;
12         goto 1
13       end;
```

Fig. 6.20 Transição Habilitada em STATION_PROC p/Tratar ConResp

Na linha 05 o estado da máquina do protocolo será alterado para ESTAB. A interação CC será colocada na fila associada à porta 1 do módulo TRANSCODE, lado 2.

Executando transcod_2, a interação CC será retirada da fila na porta 1 e empacotada numa SDU que será enviada para o meio através da primitiva UnitReq.

O processo cms_proc, depois de executado, retirará a primitiva UnitReq da fila e passará a SDU para o módulo TRANSCODE, lado 1, através da primitiva UnitInd.

O processo transcod_1 retirará a primitiva UnitInd da fila na porta 2 e desempacotará a interação CC. Essa interação será colocada na fila da porta 2 do módulo STATION, lado 1.

Station_p1 será executado e a transição habilitada será:

```
01 if c0var->.ident = 2 then
02   if s0var->.t3PEERCODE = s3CC then
03     if state in (.CRSENT.) then
04       begin
05         state := ESTAB;
06         begin
07           p0procedure(1);
08           new(s0var, c2USAP, s2ConConf);
09           s0var->.t2USAP := s2ConConf;
10           OUT
11         end;
12         CRRetranRemaining := -1;
13         goto 1
14       end;
```

Fig. 6.21 Transição Habilitada em STATION_PROC p/Tratar UDP CC

Tanto o estado da máquina do protocolo do lado 2 como do lado 1 estarão em ESTAB. A primitiva ConConf será colocada na fila associada à porta 1 do módulo USER.

Executando user_proc1, ficará encerrada a fase de estabelecimento de conexão e será iniciada a fase de transferência de dados.

```
01 if c0var->.ident = 1 then
02   if s0var->.t2USAP = s2ConConf then
03     if state in (.CALLING.) then
04       begin
05         state := CONNECTED;
06         GETCH(X);
06         while (X <> ') do
07           begin
08             p0procedure(1);
09             new(s0var, c2USAP, s2DatReq);
10             s0var->.t2USAP := s2DatReq;
11             s0var->.d2DatReq.UserData := X;
12             OUT;
13             GETCH(X)
14           end;
15         goto 1
16       end;
```

Fig. 6.22 Transição Habilitada em USER_PROC p/Tratar ConConf

Nessa transição, as mensagens de dados serão recebidas pela função GETCH, via teclado. O usuário digitará a mensagem que será imediatamente acomodada na primitiva DatReq, que a conduzirá. Essa primitiva será então colocada na fila da porta 1 do módulo STATION. O usuário poderá digitar quantas mensagens desejar.

Para cada mensagem posta na fila, o procedimento station_p1 precisará ser chamado para tratá-la. Se houver 3 mensagens na fila, o procedimento station_p1 será executado três vezes.

Acompanhando o fluxo de uma mensagem na fase de transferência de dados, inicialmente será executado station_p1, que retirará a primitiva DatReq da fila. A interação DT será colocada na fila associada à porta 1 do módulo TRANSCODE.

```
01 if c0var->.ident = 1 then
02   if s0var->.t2USAP = s2DatReq then
03     if state in (.ESTAB.) then
04       if not sending then
05         begin
06           state := ESTAB;
07           OldData := UserData;
08           begin
09             p0procedure(2);
10             new(s0var, c3PEERCODE, s3DT);
11             s0var->.t3PEERCODE := s3DT;
12             s0var->.d3DT.seq := SendSeq;
13             s0var->.d3DT.UserData := OldData;
14             OUT
15           end;
16           OldSendSeq := SendSeq;
17           SendSeq := (SendSeq + 1) mod 2;
18           Sending := true;
19           DTRetranRemaining := N-1;
20           goto 1
21         end;
```

Fig. 6.23 Transição Habilitada em STATION_PROC p/Tratar DatReq

A interação DT estará na fila da porta 1 do módulo TRANSCODE, lado 1.

Executando transcod_1, essa interação será empacotada numa SDU que será transmitida através da primitiva UnitReq. O módulo CMs receberá essa primitiva e emitirá a primitiva UnitInd para o módulo TRANSCODE, lado 2.

Executando transcod_2, a primitiva UnitInd será retirada da fila. A interação DT será desempacotada e passada para o módulo STATION.

Station_p2 será executado e a transição habilitada será:

```
01 if c0var->.ident = 2 then
02   if s0var->.t3PEERCODE = s3DT then
03     if state in (.ESTAB.) then
04       with s0var->.DT do
05         begin
06           state := ESTAB;
07           if Seq = RecvSeq then
08             begin
09               begin
10                 p0procedure(1);
11                 new(s0var, c2USAP, s2DatInd);
12                 s0var->.t2USAP := s2DatInd;
13                 s0var->.d2DatInd.UserData := UserData;
14                 OUT
15               end;
16               RecvSeq := (RecvSeq+1) mod 2
17             end;
18           begin
19             p0procedure(2);
20             new(s0var, c3PEERCODE, s3AK);
21             s0var->.t3PEERCODE := s3AK;
22             s0var->.d3AK.seq := RecvSeq;
23             OUT
24           end;
25           DTorAK := true;
26           goto 1
27         end;
```

Fig. 6.24 Transição Habilitada em STATION_PROC p/tratar UDP DT

Quando essa transição for executada, duas interações serão emitidas: primeiro, supondo sequenciamento correto de mensagens (linha 07), a primitiva DatInd será colocada na porta 1 do módulo USER, levando consigo a mensagem de dados; segundo, uma mensagem de reconhecimento (interação AK) será colocada na porta 1 do módulo TRANSCODE. Se o sequenciamento não estiver correto, a mensagem de dados não será passada para o módulo USER e somente uma AK será emitida.

Executando user_proc1, a mensagem de dados será recebida e o usuário poderá também mandar mensagens para o outro lado.

Assumindo que o lado 2 não queira mandar mensagens, o sistema estará com uma interação na porta 1 do módulo TRANSCODE, lado 2.

O processo transcod_2 será executado e a interação AK será empacotada na primitiva UnitReq, que será emitida para o meio. O processo cms_proc receberá a primitiva UnitReq e emitirá a primitiva UnitInd para o módulo TRANSCODE. Esse módulo, através do processo transcod_1, receberá a primitiva UnitInd e desempacotará a interação. O módulo STATION, através do processo station_p1, receberá a interação AK. A transição habilitada será:

```
01 if c0var->.ident = 2 then
02   if s0var->.t3PEERCODE = s3AK then
03     if state in (.ESTAB.) then
04       with s0var->.d3AK do
05         begin
06           state := ESTAB;
07           if Seq = SendSeq then
08             begin
09               Sending := false;
10               DTRetranRemaining := -1;
11               DTorAK := true
12             end;
13           goto1
14         end;
```

Fig. 6.25 Transição Habilitada em STATION_PROC p/Tratar UDP AK

6.4 Validação do design do protocolo Abracadabra [14] [15]

O objetivo principal das simulações foi o de avaliar (de uma forma pragmática) se, do ponto de vista de um observador externo, o serviço **ABRASERVICE'** (Fig. 4.3) é equivalente ao serviço **ABRASERVICE** (Fig. 4.2).

Para cada procedimento executado são apresentadas as interações ocorridas. Para uma melhor compreensão dos traços obtidos durante a simulação, os procedimentos foram enumerados. Por exemplo, o traço **ConReq(1-->2), CR(2-->3), UnitReq(3-->4), UnitInd(4-->5), CR(5-->6),ConInd(6-->7),ConResp(7-->6),CC(6-->5), UnitReq(5-->4), UnitInd(4-->3), CC(3-->2), ConConf(2-->1)** significa: **user_proc1(1)** emitiu a primitiva ConReq; **station_p1(2)** recebeu ConReq e emitiu a UDP CR; **transcod_1(3)** recebeu CR e emitiu a primitiva UnitReq; **cms_proc(4)** recebeu UnitReq e emitiu a primitiva UnitInd; **transcod_2(5)** recebeu UnitInd e emitiu a UDP CR; **station_p2(6)** recebeu CR e emitiu a primitiva ConInd; **user_proc2(7)** recebeu ConInd e emitiu a primitiva ConResp; etc...

Esse traço caracteriza o estabelecimento normal de uma conexão, onde o módulo **USER A** inicia o pedido, que é aceito por **USER B**. A recepção da primitiva **ConConf**, pelo **USER A**, determina o final da fase de conexão e o início da fase de transferência de dados.

Para verificar se esse traço é equivalente, do ponto de vista de um observador externo, ao traço referente ao estabelecimento normal de conexão, obtido quando da simulação do sistema apresentado na Fig. 4.3, é necessário abstrair as UDPs e primitivas internas ao **ABRASERVICE'**.

Para as diferentes fases do protocolo (conexão, transferência de dados, desconexão e erro) foram simulados vários cenários, relativos às situações previstas (normais e anormais) e não previstas na especificação. Alguns dos traços obtidos estão apresentados no Anexo 8.

6.4.1 Erros encontrados na simulação

Durante as simulações foram encontrados alguns erros, sendo que o mais importante foi a detecção de uma situação de impasse (deadlock) no serviço Abracadabra.

Essa situação é atingida quando:

(a) executando-se **user_proc1**, a primitiva ConReq será colocada na fila associada à porta 1 de **sap_managA**;

(b) executando-se **user_proc2**, a primitiva ConReq será colocada na fila associada à porta 1 de sap_managB;

(c) executando-se **sap_managA**, a primitiva ConReq será retirada da fila na porta 1 e será emitida a primitiva IConReqInd;

(d) nesse instante, o sistema Abraservice se apresenta com uma primitiva ConReq na fila associada à porta 1 de **sap_managB** e IConReqInd na fila da porta 2 desse mesmo processo. Executando-se **sap_managB**, a primitiva ConReq será retirada da fila na porta 1 e será emitida uma IConReqInd para **sap_managA**;

(e) executando-se **sap_managA**, a interação IConReqInd será retirada da fila e serão emitidas duas interações: ConConf para **user_proc1** e IConRespConf para **sap_managB**;

(f) executando-se **sap_managB**, a interação IConReqInd será retirada da fila associada à porta 2 (existem duas interações nessa porta) e serão emitidas as interações ConConf (para **user_proc2**) e IConRespConf (para **sap_managA**);

(g) ambos os processos **sap_managA** e **sap_managB** estão no estado CONNECTED e com a interação IConRespConf armazenada em suas filas associadas às portas 2. Executando-se qualquer um desses processos, não haverá nenhuma transição habilitada e ficará caracterizada a situação de impasse.

Para corrigir o impasse verificado na simulação, a seguinte transição foi acrescentada à especificação realizada pelos técnicos da ISO e CCITT no documento [5]:

```
when Inchn.IConRespConf
from CONNECTED
to CONNECTED
begin
end;
```

7. CONCLUSÃO

A metodologia proposta baseia-se no uso de especificações formais, descritas na TDF Estelle/83, e utiliza o compilador Estelle/83 como ferramenta para:

(a) detecção de erros estáticos (léxicos, sintáticos e de contexto) presentes nas especificações. Sempre que um erro dessa natureza ocorre, o compilador interrompe a compilação e emite para o usuário uma mensagem que identifica o tipo de erro. O usuário deve então fazer as correções no código fonte da especificação e submetê-la novamente ao compilador;

(b) geração do segmento de código Pascal relativo à especificação Estelle/83, que foi submetida ao compilador. Esse segmento de código Pascal, juntamente com os códigos de um núcleo (exploração ou simulação) e das rotinas de suporte, compõe uma biblioteca Pascal, que possibilita a execução do sistema. A grande vantagem desse processo de obtenção semi-automática da implementação, em relação a uma implementação manual, está em:

(b1) geração automática das estruturas de dados que representarão as construções próprias da linguagem Estelle/83;

(b2) estabelecimento automático das conexões entre os processos que se comunicam através da troca de interações. A implementação dessas conexões é bastante complexa e o compilador, felizmente, já as constrói para o usuário.

Se o objetivo é utilizar esse compilador para gerar semi-automaticamente uma implementação, a biblioteca Pascal deverá conter um núcleo de exploração. Se o objetivo é simular o sistema especificado, esse núcleo deverá viabilizar a simulação.

O núcleo de simulação permite a detecção de erros dinâmicos nas especificações. Isso é realizado através do exame das várias situações possíveis na especificação, onde se busca uma condição não prevista e que é capaz de impedir a evolução do serviço e/ou protocolo.

Para exemplificar a metodologia, foram utilizadas as especificações formais, em Estelle/83, do serviço e do protocolo Abracadabra.

Em relação à validação do design do protocolo Abracadabra, a simulação, embora não pode fornecer um resultado definitivo, permitiu a obtenção de uma série de traços, que confrontados aos traços obtidos quando da simulação do serviço, outorgaram uma maior confiabilidade ao projeto desse protocolo.

O núcleo de simulação foi desenvolvido com o objetivo de permitir ao observador o controle do sistema que está sendo executado. Essa estratégia permite, passo a passo, a condução da simulação, através da escolha em cada passo do procedimento a ser executado, e a investigação do fluxo de interações.

A fim de proceder a uma avaliação mais contundente dessa metodologia e do núcleo de simulação, eles estão sendo empregados na contrução e validação das especificações formais, em Estelle/83, dos protocolos de Transporte (classe 2) e Sessão (núcleo) relativos ao modelo OSI.

REFERENCIAS BIBLIOGRAFICAS

[1] J.Antão B.M., Jacques P. S., W. F. Giozza, J. Marinho de A., "Redes Locais de Computadores - Protocolos de Alto Nivel e Avaliação de Desempenho", McGraw-Hill,1986.

[2] ISO IS 7498 , "Information Processing Systems - Basic Reference Model for Open Systems Interconnection", 1983.

[3] W. Lopes de Souza , S. Stiubiener , "Especificação, Verificação e Teste de Protocolos", Relatório Técnico No. RT-01/88 GRC/UFPb, Campina Grande-Pb (Brasil), 1988.

[4] R. Groz , "Verifying Complex Properties of Protocols on a Simulation: the Observer Approach", anais do Sixth International Workshop on Protocol Specification, Testing and Verification, Gray Rocks INN - Montreal (Canadá), 1986, pp. 7-25 a 7-36.

[5] CCITT Com X-R 29-E, "Guidelines for the Application of Estelle, LOTOS and SDL", 1988, pp. 135-161.

[6] ISO TC97/SC16/WG1 subgroup B , "A FDT Based on an Extended State Transition Model", Working Document, 1983.

[7] ISO IS 9074 , "Information Processing Systems - Open Systems Interconnection - Estelle - A Formal Description Technique Based on an Extended State Transition Model", 1988.

[8] E. Ferneda , "Um Compilador para a Técnica de Descrição Formal Estelle/83", tese de mestrado relativa ao Curso de Mestrado do DSC/CCT/UFPb, Campina Grande (Pb), 1988.

[9] J. N. de Souza, W. Lopes de Souza, "Implementação Semi-automática de Protocolos de Comunicação", submetido à Revista Brasileira de Automática.

[10] G. W. Gerber, "Une Methode dImplantation Automatisée de Systèmes Specifieś Formellement", tese de mestrado, IRO/UdeM, Montreal (Canadá), 1983.

[11] C. Jard, G. V. Bochmann, "An Approach to Testing Specifications", publication #430, IRO/UdeM, Montreal (Canadá), 1981.

[12] R. Dssouli, G. V. Bochmann, "Conformance Testing with Multiple Observers", IRO/UdeM, Montreal (Canadá),1987.

[13] R. Dssouli, G. V. Bochmann, "Error Detection with Multiple Observers", IRO/UdeM, Montreal (Canadá), 1986.

[14] W. L. de Souza,J. N. de Souza, "Simulação de Especificações Formais de Protocolos de Comunicação", Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos, 11-14 de Setembro de 1989, Florianópolis-SC, pp. 103-110.

[15] W. Lopes de Souza, José Neuman de Souza, "Simulation of Protocol Formal Specifications", International Conference on Local Area Networks - INDOLAN'90", 30-31 de Janeiro de 1990, Madras-India, pp. 174-189

ANEXO 1: Especificação do serviço Abracadabra em Estelle/83

```
module ABRACADABRASERVICE;
end ABRACADABRASERVICE;

refinement ABRA_SERVICE for ABRACADABRASERVICE;

type
  UserDTyp = packed array (.1..80.) of char;

channel SAP (user,provider);
  by user:
    ConReq;
    ConResp;
    DatReq(UserData : UserDTyp);
    DisReq;
  by provider:
    ConInd;
    ConConf;
    DatInd(UserData : UserDTyp);
    DisInd;
end SAP;

module USER;
  IpU : SAP (user);
end USER;

  process USER_PROC for USER;

  var
    state : (OCIOSO, CHAMANDO, CONECTADO);
    THIS_SIDE : boolean;
    x, prim : UserDTyp;
    ACABOU, FIM : array (.boolean.) of boolean;

  procedure GETPRIM(var prim : UserDTyp);
  primitive;
  procedure GETCH(var x : UserDTyp);
  primitive;
  procedure WRITE_DATA(UserData : UserDTyp);
  primitive;
  function ENCERRA_SISTEMA : boolean;
  primitive;

  initialize
    begin
      state := OCIOSO;
    end;

  trans
    from OCIOSO
    to CHAMANDO
    begin
      out IpU.ConReq;
    end;
```

```
  when IpU.ConInd
  to OCIOSO
  begin
    out IpU.DisReq;
  end;

  when IpU.ConInd
  from OCIOSO
  to CONECTADO
  begin
    out IpU.ConResp;
  end;

  when IpU.ConInd
  from CHAMANDO
  to CONECTADO
  begin
    out IpU.ConResp;
  end;

  when IpU.DatInd
  from CONECTADO
  to CONECTADO
  begin
    WRITE_DATA(UserData);
  end;

  when IpU.DisInd
  from CONECTADO
  to OCIOSO
  begin
  end;

  when IpU.ConConf
  from CHAMANDO
  to CONECTADO
  begin
    GETCH(x);
    while (x <> ' ') do
      begin
        out IpU.DatReq(x);
        GETCH(x);
      end
  end;

  when IpU.DisInd
  from CHAMANDO
  to OCIOSO
  begin
  end;

end USER_PROC;
```

```
module ABRA;
  user1 : SAP(provider);
  user2 : SAP(provider);
end ABRA;

refinement ABRA_BODY for ABRA;

channel INTERNAL(all);
  by all:
    IConReqInd;
    IConRespConf;
    IDat(UserData : UserDTyp);
    IDis;
end INTERNAL;

  module SAPMANAGER;
    User : SAP(provider);
    Inch : INTERNAL(all);
  end SAPMANAGER;

  process SAP_MANAG for SAPMANAGER;

  var
    state : (DISCONNECTED, CALLED, CALLING, CONNECTED);

  initialize
    begin
      state := DISCONNECTED;
    end;

  trans
    when User.ConReq
    from DISCONNECTED
    to DISCONNECTED
    begin
      out User.DisInd;
    end;

    when Inch.Idat
    from DISCONNECTED
    to DISCONNECTED
    begin
      out Inch.IDis;
    end;

    when User.ConReq
    from DISCONNECTED
    to CALLING
    begin
      out Inch.IConReqInd;
    end;
```

```
when Inch.IConReqInd
from DISCONNECTED
to CALLED
begin
  out User.ConInd;
end;

when User.ConResp
from CALLED
to CONNECTED
begin
  out Inch.IConRespConf;
end;

when User.ConReq
from CALLED
to CONNECTED
begin
  out User.ConConf;
  out Inch.IConRespConf;
end;

when Inch.IConRespConf
from CALLING
to CONNECTED
begin
  out User.ConConf;
end;

when Inch.IConReqInd
from CALLING
to CONNECTED
begin
  out User.ConConf;
  out Inch.IConRespConf;
end;

when User.DatReq
from CONNECTED
to CONNECTED
begin
  out Inch.IDat(UserData);
end;

when Inch.IDat
from CONNECTED
to CONNECTED
begin
  out User.DatInd(UserData);
end;
```

```
when User.DatReq
 from CONNECTED
 to DISCONNECTED
 begin
   out User.DisInd;
   out Inch.IDis;
 end;

 when Inch.IConReqInd
 from CONNECTED
 to CALLED
 begin
   out User.ConInd;
 end;

 from CALLED
 to DISCONNECTED
 begin
   out User.DisInd;
   out Inch.IDis;
 end;

 from CALLING
 to DISCONNECTED
 begin
   out User.DisInd;
   out Inch.IDis;
 end;

 from CONNECTED
 to DISCONNECTED
 begin
   out User.DisInd;
   out Inch.IDis;
 end;

 when User.DisReq
 from CALLED
 to DISCONNECTED
 begin
   out Inch.IDis;
 end;

 when User.DisReq
 from CALLING
 to DISCONNECTED
 begin
   out Inch.IDis;
 end;
```

```
    when User.DisReq
    from CONNECTED
    to DISCONNECTED
    begin
      out Inch.IDis;
    end;

    when Inch.IDis
    from CALLED
    to DISCONNECTED
    begin
      out User.DisInd
    end;

    when Inch.IDis
    from CALLING
    to DISCONNECTED
    begin
      out User.DisInd;
    end;

    when Inch.IDis
    from CONNECTED
    to DISCONNECTED
    begin
      out User.DisInd;
    end;

  end SAP_MANAG;

  A : SAPMANAGER with SAP_MANAG;

  B : SAPMANAGER with SAP_MANAG;

  connect A.Inch to B.Inch;

  replace User1 by A.User;

          User2 by B.User;

end ABRA_BODY;

U1 : USER with USER_PROC;

AS : ABRA with ABRA_BODY;

U2 : USER with USER_PROC;

connect U1.IpU to AS.User1;

        U2.IpU to AS.User2;

end ABRA_SERVICE;
```

```
segment sersai2;

type
  UserDTyp =
    packed array (.1..80.) of char;
  channel =
    (C1SAP,C2INTERNAL,C0USER_PROC,C0SAP_manag);
  S1SAP =
    (S1ConReq,S1ConResp,S1DatReq,S1DisReq,S1ConInd,S1ConConf,S1DatInd,
     S1DisInd);

  S2INTERNAL =
    (S2IConReqInd,S2IConRespConf,S2IDat,S2IDis);
  S0USER_PROC =
    (R1ANY);
  S0SAP_manag =
    (R2ANY);
  S1TYPE =
    ->S0TYPE;
  S0TYPE =
    record
      NEXT : S1TYPE;
      case CHANNEL of
        C1SAP :
          (case T1SAP : S1SAP of
            S1ConReq :
              ();
            S1ConResp :
              ();
            S1DatReq :
              (D1DatReq : record
                 UserData : UserDTyp
               end;
               );
            S1DisReq :
              ();
            S1ConInd :
              ();
            S1ConConf :
              ();
            S1DatInd :
              (D1DatInd : record
                 UserData : UserDTyp
               end;
               );
            S1DisInd :
              ();

          );
        C2INTERNAL :
          (case T2INTERNAL : S2INTERNAL of
```

```
            S2IConReqInd :
              ( );
            S2IConRespConf :
              ( );
            S2IDat :
              (D2IDat : record
                 UserData : UserDTyp
              end;
               );
            S2IDis :
              ( );

          );
        COUSER_PROC :
          (case TOUSER_PROC : SOUSER_PROC of
            R1ANY :
              ( );

          );
        COSAP_manag :
          (case TOSAP_manag : SOSAP_manag of
            R2ANY :
              ( );

          );


  end;
I1TYPE =
  ->IOTYPE;
C1TYPE =
  ->COTYPE;
P1TYPE =
  ->POTYPE;
IOTYPE =
  record
    NEXT : I1TYPE;
    IDENT : integer
  end;
AOTYPE =
  record
    PROC : P1TYPE;
    CHAN : C1TYPE
  end;
COTYPE =
  record
    IDENT : integer;
    INDLIST : I1TYPE;
    TARGET : AOTYPE
  end;
P2TYPE =
  packed array (.1..10.) of char;
PROCESS =
  (POUSER_PROC,POSAP_manag);
POTYPE =
```

```
    record
      IDENT : P2TYPE;
      CHANLIST : C1TYPE;
      NEXT : P1TYPE;
      case PROCESS of
        POUSER_PROC :
          (DOUSER_PROC : record
            state : (OCIOSO,CHAMANDO,CONECTADO);
            THIS_SIDE : boolean;
            x, prim : UserDTyp;
            ACABOU,FIM : array (.boolean.) of boolean;

          end);
        POSAP_manag :
          (DOSAP_manag : record
            state : (DISCONNECTED,CALLED,CALLING,CONNECTED);

          end);


    end;

var
  POVAR : P1TYPE;
  COVAR : C1TYPE;
  SOVAR : S1TYPE;

function FOFUNCTION(INDPOS : integer) : integer;

external;

procedure OUT;

external;

procedure POPROCEDURE(IDENT : integer);

external;

procedure P1PROCEDURE(IDENT : P2TYPE; var P : P1TYPE);

external;

procedure P2PROCEDURE(IDENT : integer);

external;

procedure P3PROCEDURE(Q : boolean; IDENT : integer);

external;

procedure P4PROCEDURE(INDVAL,INDPOS : integer);

external;
```

```
procedure P5PROCEDURE(I,J : integer; P : P1TYPE);

external;

procedure P6PROCEDURE(var X : AOTYPE);

external;

procedure P7PROCEDURE(P : P1TYPE; var Q : P1TYPE);

external;

procedure P8PROCEDURE(P : P1TYPE);

external;

procedure GETPRIM(var prim : userdtyp);

external;

procedure GETCH(var x : UserDTyp);

external;

procedure WRITE_DATA(UserData : UserDTyp);

external;

function ENCERRA_SISTEMA : boolean;

external;

procedure USER_PROC;

external;

procedure USER_PROC;

label
  1, 2, 3;

var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
  SINAL : char;

begin
  C1VAR:=COVAR ;
  S1VAR:=SOVAR ;
  with POVAR->.DOUSER_PROC do
    begin
      if COVAR->.IDENT = 0 then
        if SOVAR->.TOUSER_PROC = R1ANY then
          if state in (.OCIOSO.) then
            if ENCERRA_SISTEMA then
```

```
                begin
                  writeln;writeln('TRANSICAO 1 HABILITADA');writeln;
                  goto 1
                end;

      ;
/*    if COVAR->.IDENT = 0 then
        if SOVAR->.TOUSER_PROC = R1ANY then
          if state in (.OCIOSO.) then
            begin
              GETSIDE(LADO);
              if LADO <> THIS_SIDE then
                 goto 1
            end

      ;   */
      if COVAR->.IDENT = 0 then
        if SOVAR->.TOUSER_PROC = R1ANY then
          if state in (.CONECTADO.) then
            begin
              writeln;writeln('TRANSICAO 3 HABILITADA');writeln;
 writeln;writeln('** CONEXAO JA ESTABELECIDA ** ');
              GETCH(x);
              while (x<> '  ') do
                begin
                  POPROCEDURE(1 );
                  new(SOVAR ,C1SAP ,S1DatReq );
                  SOVAR->.T1SAP:=S1DatReq ;
                  SOVAR->.D1DatReq.UserData:=x ;
                  OUT;
                  GETCH(x)
                end;
 writeln;writeln('** NAO TENHO MENSAGENS P/TRANSMITIR **');
writeln;writeln(' ** DESEJA INICIAR DESCONEXAO, s/n **');
writeln;
              readln(SINAL);
              if (SINAL='s') or (SINAL='S') then
                begin
writeln; writeln('** FINAL DA FASE DE TRANSFERENCIA DE DADOS **');
writeln; writeln('** INICIO DA FASE DE DESCONEXAO **' ); writeln;
                  state:=OCIOSO;
                  begin
                    POPROCEDURE(1 );
                    new(SOVAR ,C1SAP ,S1DisReq );
                    SOVAR->.T1SAP:=S1DisReq ;
                    OUT
                  end
                end;
              goto 1
            end

      ;
      if COVAR->.IDENT = 0 then
        if SOVAR->.TOUSER_PROC = R1ANY then
          if state in (.OCIOSO.) then
```

```
            begin
              writeln;writeln('TRANSICAO 4 HABILITADA');writeln;
2 :           GETPRIM(prim);
              if prim='conreq' then
                begin
                  state:=CHAMANDO;
                  writeln;
                  writeln('** INICIO DA FASE DE CONEXAO  **');
                  writeln;
                  begin
                    POPROCEDURE(1 );
                    new(SOVAR ,C1SAP ,S1ConReq );
                    SOVAR->.T1SAP:=S1ConReq ;
                    OUT
                  end;
writeln; writeln(' O MODULO USUARIO LIBEROU UMA PRIMITIVA ConReq ');
writeln;
                  goto 1
                end
              else
                begin
                  writeln;
                  writeln('** PRIMITIVA EMITIDA INDEVIDAMENTE ** ');
                  writeln;
                  goto 2
                end
            end

      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1SAP = S1ConInd then
          if state in (.OCIOSO.) then
            begin
              writeln;writeln('TRANSICAO 5 HABILITADA');writeln;
writeln; writeln(' o modulo USUARIO recebeu uma PRIMITIVA ConInd ');
3 :           GETPRIM(prim);
              if prim='conresp' then
                begin
                  state:=CONECTADO;
                  begin
                    POPROCEDURE(1 );
                    new(SOVAR ,C1SAP ,S1ConResp );
                    SOVAR->.T1SAP:=S1ConResp ;
                    OUT
                  end;
writeln(' o modulo USUARIO liberou uma PRIMITIVA ConResp ');writeln;
                  goto 1
                end;
              if prim='disreq' then
                begin
                  writeln;
                  writeln('** O LADO CHAMADO NAO ACEITOU A CONEXAO **');
                  writeln;
                    begin
                      POPROCEDURE(1 );
```

```
                    new(SOVAR ,C1SAP ,S1DisReq );
                    SOVAR->.T1SAP:=S1DisReq ;
                    OUT
                  end;
                goto 1
              end
            else
              begin
                writeln;
                writeln('** PRIMITIVA EMITIDA INDEVIDAMENTE ** ');
                writeln;
                goto 3
              end
          end

      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1SAP = S1ConInd then
          if state in (.CHAMANDO.) then
            begin
              writeln;writeln('TRANSICAO 6 HABILITADA');writeln;
              state:=CONECTADO ;
              begin
                POPROCEDURE(1 );
                new(SOVAR ,C1SAP ,S1ConResp );
                SOVAR->.T1SAP:=S1ConResp ;
                OUT
              end;
writeln;writeln('o modulo USER recebeu ConInd e liberou ConResp');
              goto 1
            end

      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1SAP = S1DatInd then
          if state in (.CONECTADO .) then
            with SOVAR->.D1DatInd do
              begin
                writeln;writeln('TRANSICAO 7 HABILITADA');writeln;
                state:=CONECTADO;
                writeln;
                writeln('O LADO CHAMADO/CHAMANTE ACUSA RECEPCAO');
                writeln('DE DADOS ATRAVES DA primitiva DatInd' );
                writeln;
                WRITE_DATA(UserData);
                goto 1
              end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1SAP = S1DisInd then
          if state in (.CONECTADO .) then
            begin
              writeln;writeln('TRANSICAO 8 HABILITADA');writeln;
```

```
                state:=OCIOSO;
writeln;writeln('o modulo USER recebeu DisInd e seu estado e OCIOSO');
                goto 1
              end


       ;
       if COVAR->.IDENT = 1 then
         if SOVAR->.T1SAP = S1ConConf then
           if state in (.CHAMANDO.) then
             begin
               writeln;writeln('TRANSICAO 9 HABILITADA');writeln;
writeln; writeln(' ** FINAL DA FASE DE CONEXAO ** '); writeln;
               state:=CONECTADO ;
               writeln;
               writeln('* INICIO DA FASE DE TRANSFERENCIA DE DADOS *');
               writeln;
               GETCH(x);
               while (x<>'  ') do
                 begin
                   POPROCEDURE(1 );
                   new(SOVAR ,C1SAP ,S1DatReq );
                   SOVAR->.T1SAP:=S1DatReq ;
                   SOVAR->.D1DatReq.UserData:=x ;
                   OUT;
                   GETCH(x)
                 end;
 writeln;writeln('** NAO TENHO MENSAGENS P/TRANSMITIR **'); writeln;
               goto 1
             end

       ;
       if COVAR->.IDENT = 1 then
         if SOVAR->.T1SAP = S1DisInd then
           if state in (.CHAMANDO.) then
             begin
               writeln;writeln('TRANSICAO 10 HABILITADA');writeln;
               state:=OCIOSO ;
writeln(' USUARIO RECEBEU UMA DisInd E ESTA NO ESTADO OCIOSO ');
               goto 1
             end

       ;
       writeln('** O PROCESSO NAO TEM TRANSICAO HABILITADA **');

     end;
   1 : ;
/*1 : case C1VAR->.IDENT of
     0 :
       case S1VAR->.TOUSER_PROC of
         R1ANY :
           dispose(S1VAR ,COUSER_PROC ,R1ANY );

       end;
     1 :
```

```
      case S1VAR->.T1SAP of
        S1ConInd :
          dispose(S1VAR ,C1SAP ,S1ConInd );
        S1DatInd :
          dispose(S1VAR ,C1SAP ,S1DatInd );
        S1DisInd :
          dispose(S1VAR ,C1SAP ,S1DisInd );
        S1ConConf :
          dispose(S1VAR ,C1SAP ,S1ConConf );

      end;

  end;  */

end;

procedure SAP_manag;

external;

procedure SAP_manag;

label
  1;

var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;

begin
  C1VAR:=C0VAR ;
  S1VAR:=S0VAR ;
  with P0VAR->.D0SAP_manag do
    begin
/**   if C0VAR->.IDENT = 1 then
        if S0VAR->.T1SAP = S1ConReq then
          if state in ( .DISCONNECTED .) then
            begin
              state:=DISCONNECTED ;
              begin
                P0PROCEDURE(1 );
                new(S0VAR ,C1SAP ,S1DisInd );
                S0VAR->.T1SAP:=S1DisInd ;
                OUT
              end;
              goto 1
            end


      ;  **/
      if C0VAR->.IDENT = 2 then
        if S0VAR->.T2INTERNAL = S2IDat then
          if state in ( .DISCONNECTED .) then
            with S0VAR->.D2IDat do
              begin
```

```
                writeln;writeln('TRANSICAO 2 HABILITADA');writeln;
                state:=DISCONNECTED ;
                begin
                  POPROCEDURE(2 );
                  new(SOVAR ,C2INTERNAL ,S2IDis );
                  SOVAR->.T2INTERNAL:=S2IDis ;
                  OUT
                end;
                goto 1
              end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1SAP = S1ConReq then
          if state in (.DISCONNECTED .) then
            begin
              writeln;writeln('TRANSICAO 3 HABILITADA');writeln;
              state:=CALLING ;
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C2INTERNAL ,S2IConReqInd );
                SOVAR->.T2INTERNAL:=S2IConReqInd ;
                OUT
              end;
writeln(' RECEBIDA UMA ConReq E LIBERADA UMA IConReqInd ');
              goto 1
            end


      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T2INTERNAL = S2IConReqInd then
          if state in (.DISCONNECTED .) then
            begin
              writeln;writeln('TRANSICAO 4 HABILITADA');writeln;
              state:=CALLED ;
              begin
                POPROCEDURE(1 );
                new(SOVAR ,C1SAP ,S1ConInd );
                SOVAR->.T1SAP:=S1ConInd ;
                OUT
              end;
writeln(' RECEBIDA UMA IConReqInd E LIBERADA UMA ConInd ');
              goto 1
            end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1SAP = S1ConResp then
          if state in (.CALLED .) then
            begin
              writeln;writeln('TRANSICAO 5 HABILITADA');writeln;
              state:=CONNECTED ;
```

```
                begin
                  POPROCEDURE( 2 );
                  new( SOVAR ,C2INTERNAL ,S2IConRespConf );
                  SOVAR->.T2INTERNAL:=S2IConRespConf ;
                  OUT
                end;
writeln( ' RECEBIDA UMA ConResp E LIBERADA UMA IConRespConf ');
                goto 1
              end


       ;
       if COVAR->.IDENT = 1 then
         if SOVAR->.T1SAP = S1ConReq then
           if state in ( .CALLED . ) then
             begin
               writeln;writeln('TRANSICAO 6 HABILITADA');writeln;
               state:=CONNECTED ;
               begin
                 POPROCEDURE( 1 );
                 new( SOVAR ,C1SAP ,S1ConConf );
                 SOVAR->.T1SAP:=S1ConConf ;
                 OUT
               end;
               begin
                 POPROCEDURE( 2 );
                 new( SOVAR ,C2INTERNAL ,S2IConRespConf );
                 SOVAR->.T2INTERNAL:=S2IConRespConf ;
                 OUT
               end;
write('RECEBIDA UMA ConReq E LIBERADAS: ');
write(' UMA ConConf E UMA IConRespConf');
writeln;
                 goto 1
               end


       ;
       if COVAR->.IDENT = 2 then
         if SOVAR->.T2INTERNAL = S2IConRespConf then
           if state in ( .CALLING . ) then
             begin
               writeln;writeln('TRANSICAO 7 HABILITADA');writeln;
               state:=CONNECTED ;
               begin
                 POPROCEDURE( 1 );
                 new( SOVAR ,C1SAP ,S1ConConf );
                 SOVAR->.T1SAP:=S1ConConf ;
                 OUT
               end;
writeln( ' RECEBIDA UMA IConRespConf E LIBERADA UMA ConConf ');
                goto 1
              end
```

```
      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T2INTERNAL = S2IConReqInd then
          if state in (.CALLING .) then
            begin
              writeln;writeln('TRANSICAO 8 HABILITADA');writeln;
              state:=CONNECTED ;
              begin
                POPROCEDURE(1 );
                new(SOVAR ,C1SAP ,S1ConConf );
                SOVAR->.T1SAP:=S1ConConf ;
                OUT
              end;
writeln('LIBERADA uma ConConf - COLISAO DE ConReq -');
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C2INTERNAL ,S2IConRespConf );
                SOVAR->.T2INTERNAL:=S2IConRespConf ;
                OUT
              end;
writeln;writeln('LIBERADA uma IConRespConf');writeln;
              goto 1
            end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1SAP = S1DatReq then
          if state in (.CONNECTED .) then
            with SOVAR->.D1DatReq do
              begin
                writeln;writeln('TRANSICAO 9 HABILITADA');writeln;
                state:=CONNECTED ;
                begin
                  POPROCEDURE(2 );
                  new(SOVAR ,C2INTERNAL ,S2IDat );
                  SOVAR->.T2INTERNAL:=S2IDat ;
                  SOVAR->.D2IDat.UserData:=UserData ;
                  OUT
                end;
writeln(' RECEBIDA UMA DatReq E LIBERADA UMA IDat ');
                goto 1
              end


      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T2INTERNAL = S2IDat then
          if state in (.CONNECTED .) then
            with SOVAR->.D2IDat do
              begin
                writeln;writeln('TRANSICAO 10 HABILITADA');writeln;
                state:=CONNECTED ;
                begin
                  POPROCEDURE(1 );
```

```
                    new(SOVAR ,C1SAP ,S1DatInd );
                    SOVAR->.T1SAP:=S1DatInd ;
                    SOVAR->.D1DatInd.UserData:=UserData ;
                    OUT
                  end;
writeln(' RECEBIDA UMA IDat E LIBERADA UMA DatInd ');
                  goto 1
                end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1SAP = S1DatReq then
          if state in (.CONNECTED .) then
            with SOVAR->.D1DatReq do
              begin
                writeln;writeln('TRANSICAO 11 HABILITADA');writeln;
                state:=DISCONNECTED ;
                begin
                  POPROCEDURE(1 );
                  new(SOVAR ,C1SAP ,S1DisInd );
                  SOVAR->.T1SAP:=S1DisInd ;
                  OUT
                end;
                begin
                  POPROCEDURE(2 );
                  new(SOVAR ,C2INTERNAL ,S2IDis );
                  SOVAR->.T2INTERNAL:=S2IDis ;
                  OUT
                end;
                goto 1
              end


      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T2INTERNAL = S2IConReqInd then
          if state in (.CONNECTED .) then
            begin
              writeln;writeln('TRANSICAO 12 HABILITADA');writeln;
              state:=CALLED ;
              begin
                POPROCEDURE(1 );
                new(SOVAR ,C1SAP ,S1ConInd );
                SOVAR->.T1SAP:=S1ConInd ;
                OUT
              end;
              goto 1
            end


      ;
      if COVAR->.IDENT = 0 then
        if SOVAR->.TOSAP_manag = R2ANY then
          if state in (.CALLED .) then
```

```
            begin
              writeln;writeln('TRANSICAO 13 HABILITADA');writeln;
              state:=DISCONNECTED ;
              begin
                POPROCEDURE(1 );
                new(SOVAR ,C1SAP ,S1DisInd );
                SOVAR->.T1SAP:=S1DisInd ;
                OUT
              end;
writeln('LIBERADA uma DisInd - TRANSICAO ESPONTANEA HABILITADA');
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C2INTERNAL ,S2IDis );
                SOVAR->.T2INTERNAL:=S2IDis ;
                OUT
              end;
              goto 1
            end


      ;
      if COVAR->.IDENT = 0 then
        if SOVAR->.T0SAP_manag = R2ANY then
          if state in (.CALLING .) then
            begin
              writeln;writeln('TRANSICAO 14 HABILITADA');writeln;
              state:=DISCONNECTED ;
              begin
                POPROCEDURE(1 );
                new(SOVAR ,C1SAP ,S1DisInd );
                SOVAR->.T1SAP:=S1DisInd ;
                OUT
              end;
writeln('LIBERADA uma DisInd - TRANSICAO ESPONTANEA HABILITADA');
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C2INTERNAL ,S2IDis );
                SOVAR->.T2INTERNAL:=S2IDis ;
                OUT
              end;
              goto 1
            end


      ;
      if COVAR->.IDENT = 0 then
        if SOVAR->.T0SAP_manag = R2ANY then
          if state in (.CONNECTED .) then
            begin
              writeln;writeln('TRANSICAO 15 HABILITADA');writeln;
              state:=DISCONNECTED ;
              begin
                POPROCEDURE(1 );
                new(SOVAR ,C1SAP ,S1DisInd );
                SOVAR->.T1SAP:=S1DisInd ;
```

```
                OUT
              end;
writeln('LIBERADA uma DisInd - TRANSICAO ESPONTANEA HABILITADA');
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C2INTERNAL ,S2IDis );
                SOVAR->.T2INTERNAL:=S2IDis ;
                OUT
              end;
              goto 1
            end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1SAP = S1DisReq then
          if state in (.CALLED .) then
            begin
              writeln;writeln('TRANSICAO 16 HABILITADA');writeln;
              state:=DISCONNECTED ;
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C2INTERNAL ,S2IDis );
                SOVAR->.T2INTERNAL:=S2IDis ;
                OUT
              end;
writeln(' RECEBIDA UMA DisReq E LIBERADA UMA IDis ');
              goto 1
            end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1SAP = S1DisReq then
          if state in (.CALLING .) then
            begin
              writeln;writeln('TRANSICAO 17 HABILITADA');writeln;
              state:=DISCONNECTED ;
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C2INTERNAL ,S2IDis );
                SOVAR->.T2INTERNAL:=S2IDis ;
                OUT
              end;
writeln(' RECEBIDA UMA DisReq E LIBERADA UMA IDis ');
              goto 1
            end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1SAP = S1DisReq then
          if state in (.CONNECTED .) then
            begin
              writeln;writeln('TRANSICAO 18 HABILITADA');writeln;
```

```
                state:=DISCONNECTED ;
                begin
                  POPROCEDURE( 2 );
                  new( SOVAR ,C2INTERNAL ,S2IDis );
                  SOVAR->.T2INTERNAL:=S2IDis ;
                  OUT
                end;
writeln(' RECEBIDA UMA DisReq E LIBERADA UMA IDis ');
                goto 1
              end


      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T2INTERNAL = S2IDis then
          if state in (.CALLED .) then
            begin
              writeln;writeln('TRANSICAO 19 HABILITADA');writeln;
              state:=DISCONNECTED ;
              begin
                POPROCEDURE( 1 );
                new( SOVAR ,C1SAP ,S1DisInd );
                SOVAR->.T1SAP:=S1DisInd ;
                OUT
              end;
writeln('RECEBIDA UMA IDis E LIBERADA UMA DisInd ');
              goto 1
            end


      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T2INTERNAL = S2IDis then
          if state in (.CALLING .) then
            begin
              writeln;writeln('TRANSICAO 20 HABILITADA');writeln;
              state:=DISCONNECTED ;
              begin
                POPROCEDURE( 1 );
                new( SOVAR ,C1SAP ,S1DisInd );
                SOVAR->.T1SAP:=S1DisInd ;
                OUT
              end;
writeln('RECEBIDA UMA IDis E LIBERADA UMA DisInd ');
              goto 1
            end


      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T2INTERNAL = S2IDis then
          if state in (.CONNECTED .) then
            begin
              writeln;writeln('TRANSICAO 21 HABILITADA');writeln;
              state:=DISCONNECTED ;
```

```
                begin
                  POPROCEDURE(1 );
                  new(SOVAR ,C1SAP ,S1DisInd );
                  SOVAR->.T1SAP:=S1DisInd ;
                  OUT
                end;
writeln('RECEBIDA UMA IDis E LIBERADA UMA DisInd ');
                goto 1
              end
        ;
 /**   if COVAR->.IDENT = 2 then
         if SOVAR->.T2INTERNAL = S2IConRespConf then
           if state in (.CONNECTED.) then
             begin
               writeln;writeln('TRANSICAO 22 HABILITADA');writeln;
               state := CONNECTED;
               dispose(SOVAR);
               goto 1
             end
        ;  **/

        writeln('O PROCESSO NAO TEM TRANSICAO HABILITADA');

      end;
   1 : ;
/*1  : case C1VAR->.IDENT of
      1 :
        case S1VAR->.T1SAP of
          S1ConReq :
            dispose(S1VAR ,C1SAP ,S1ConReq );
          S1ConResp :
            dispose(S1VAR ,C1SAP ,S1ConResp );
          S1DatReq :
            dispose(S1VAR ,C1SAP ,S1DatReq );
          S1DisReq :
            dispose(S1VAR ,C1SAP ,S1DisReq );

        end;
      2 :
        case S1VAR->.T2INTERNAL of
          S2IDat :
            dispose(S1VAR ,C2INTERNAL ,S2IDat );
          S2IConReqInd :
            dispose(S1VAR ,C2INTERNAL ,S2IConReqInd );
          S2IConRespConf :
            dispose(S1VAR ,C2INTERNAL ,S2IConRespConf );
          S2IDis :
            dispose(S1VAR ,C2INTERNAL ,S2IDis );

        end;
      0 :
        case S1VAR->.TOSAP_manag of
          R2ANY :
            dispose(S1VAR ,COSAP_manag ,R2ANY );
```

```
      end;

  end;  */

end;

procedure IOABRA_SERVICE(var P1VAR : P1TYPE);
external;

procedure IOABRA_SERVICE;

var
  P2VAR : P1TYPE;
  A0VAR,A1VAR : A0TYPE;

procedure IOUSER_PROC(var P1VAR : P1TYPE);

begin
  new(P0VAR ,P0USER_PROC );
  P1PROCEDURE('USER_PROC ' ,P1VAR );
  P3PROCEDURE(true ,0 );
  P3PROCEDURE(true ,1 );
  with P0VAR->.D0USER_PROC do
    begin
      state:=0CIOSO ;

    end
end;

procedure IOABRA_BODY(var P1VAR : P1TYPE);

var
  P2VAR : P1TYPE;
  A0VAR,A1VAR : A0TYPE;

procedure IOSAP_manag(var P1VAR : P1TYPE);

begin
  new(P0VAR ,P0SAP_manag );
  P1PROCEDURE('SAP_manag ' ,P1VAR );
  P3PROCEDURE(true ,0 );
  P3PROCEDURE(true ,1 );
  P3PROCEDURE(true ,2 );
  with P0VAR->.D0SAP_manag do
    begin
      state:=DISCONNECTED
    end
end;

begin
  P2VAR:=nil ;
  IOSAP_manag(P2VAR );
  IOSAP_manag(P2VAR );
  P7PROCEDURE(P2VAR ,P1VAR );
  P3PROCEDURE(false ,1 );
```

```
   P3PROCEDURE(false ,2 );
   P5PROCEDURE(2 ,2 ,P1VAR );
   P6PROCEDURE(A0VAR );
   P5PROCEDURE(1 ,2 ,P1VAR );
   P6PROCEDURE(A1VAR );
   A0VAR.CHAN->.TARGET:=A1VAR ;
   A1VAR.CHAN->.TARGET:=A0VAR ;
p5procedure(0 ,1 ,p1var);
a0var.chan := c0var;
p5procedure(2 ,1 ,p1var);
p6procedure(a0var.chan->.target);
p5procedure(0 ,2 ,p1var);
a0var.chan := c0var;
p5procedure(1 ,1 ,p1var);
p6procedure(a0var.chan->.target);
   P8PROCEDURE(P1VAR );

end;

begin
   P2VAR:=nil ;
   IOUSER_PROC(P2VAR );
   IOABRA_BODY(P2VAR );
   IOUSER_PROC(P2VAR );
   P7PROCEDURE(P2VAR ,P1VAR );
   P5PROCEDURE(3 ,1 ,P1VAR );
   P6PROCEDURE(A0VAR );
   P5PROCEDURE(2 ,1 ,P1VAR );
   P6PROCEDURE(A1VAR );
   A0VAR.CHAN->.TARGET:=A1VAR ;
   A1VAR.CHAN->.TARGET:=A0VAR ;
   P5PROCEDURE(1 ,1 ,P1VAR );
   P6PROCEDURE(A0VAR );
   P5PROCEDURE(2 ,2 ,P1VAR );
   P6PROCEDURE(A1VAR );
   A0VAR.CHAN->.TARGET:=A1VAR ;
   A1VAR.CHAN->.TARGET:=A0VAR ;
   P8PROCEDURE(P1VAR );

end;
```

ANEXO 3: Código do núcleo de simulação para o serviço

```
program NUCSER_3(input,output);

const
  IDENTLEN = 10;
  MAXLIN = 80;

type
  DATA_TYPE      = packed array (.1..MAXLIN.) of char;
  CHANNEL        = (COUSER_PROC,COSAP_MANAG);                    /**/
  SOUSER_PROC = (R1ANY);                                        /**/
  SOSAP_MANAG = (R2ANY);                                       /**/
  S1TYPE         = ->SOTYPE;                                          /**
  SOTYPE         = record
                     NEXT : S1TYPE;
                     case CHANNEL of
                       COUSER_PROC :                                 /**/
                         (case TOUSER_PROC : SOUSER_PROC of         /**/
                            R1ANY :                                /**/
                              ( )                                        /**
                         );                                              /**/
                       COSAP_MANAG :                                 /**/
                         (case TOSAP_MANAG : SOSAP_MANAG of        /**/
                            R2ANY :                                     /**
                              ( )                                       /**/
                         );
                   end;
  I1TYPE         = ->IOTYPE;
  C1TYPE         = ->COTYPE;
  P1TYPE         = ->POTYPE;
  IOTYPE         = record
                     NEXT  : I1TYPE;
                     IDENT : integer;
                   end;
  AOTYPE         = record
                     PROC : P1TYPE;
                     CHAN : C1TYPE;
                   end;
  COTYPE         = record
                     IDENT   : integer;
                     INDLIST : I1TYPE;
                     TARGET  : AOTYPE;
                     NEXT    : C1TYPE;
                     case QUEUED : boolean of
                       false :
                         ( );
                       true :
                         (HEAD : S1TYPE);
                   end;
  P2TYPE         = packed array (.1..10.) of char;
  POTYPE         = record
                     IDENT      : P2TYPE;
```

```
                    CHANLIST    : C1TYPE;
                    NEXT,
                    REFINEMENT : P1TYPE
                  end;

var
/* s                : s1type; */
  POVAR             : P1TYPE;
  COVAR             : C1TYPE;
  SOVAR             : S1TYPE;
  LINHA             : packed array(.1..MAXLIN.) of char;                  /**/
  ACABOU            : array (.boolean.) of boolean;                       /**/
  PRIMITIVA : DATA_TYPE;
  RESP              : char;
  MODULO            : p2type;

procedure WRITE_DATA(D : DATA_TYPE); external; /**/
                                                                          /**/
procedure WRITE_DATA;                                                     /**/
                                                                          /**/
begin                                                                     /**/
  write(clock:9,' usuario ');                                             /**/
  if pOvar->.ident = 'user_proc1' then                                    /**/
    write('1 recebeu de 2')                                               /**/
  else                                                                    /**/
    write('2 recebeu de 1');                                              /**/
  writeln(' a mensagem ',D)                                        /**/
end;                                                                      /**/

procedure GETPRIM(var prim : DATA_TYPE);
external;

procedure GETPRIM;

begin
  writeln;
  if pOvar->.ident = 'user_proc1' then
    begin
      writeln;
      writeln('                                              LADO  1  ');
      writeln; writeln;
      writeln('USUARIO, ENTRE COM O NOME DA PRIMITIVA')
    end
  else
    begin
      writeln;
      writeln('                                              LADO  2  ');
      writeln; writeln;
      writeln('USUARIO, ENTRE COM O NOME DA PRIMITIVA')
    end;
  writeln;
  readln(PRIMITIVA);
  if (PRIMITIVA='conreq') or (PRIMITIVA='disreq')
                          or (PRIMITIVA='conresp')
                          or (PRIMITIVA='datreq') then
```

```
    prim := PRIMITIVA
  else
    begin
      writeln; write('ERRO NO PREENCHIMENTO DA PRIMITIVA.');
      writeln(' TENTE NOVAMENTE.');
      GETPRIM(prim)
    end
end;

function ENCERRA_SISTEMA : boolean;  external;

function ENCERRA_SISTEMA;

begin
  writeln;writeln('** DESEJA ENCERRAR EXECUCAO DO SISTEMA? s/n **');
  writeln;
  readln(RESP);
  if (RESP = 's') or (RESP = 'S') then
    begin
      ENCERRA_SISTEMA := true;
      ACABOU(.false.) := true;
      ACABOU(.true.) := true
    end
  else
    ENCERRA_SISTEMA := false
end;

procedure GETCH(var x : DATA_TYPE);       /**/
                                        external;                          /**/
procedure GETCH;                                                           /**/
                                                                           /**/
begin                                                                      /**/
  if p0var->).ident = 'user_proc1' then                 /**/
    begin                                                                  /**/
      writeln;
      writeln('                                          LADO   1   ');
      writeln; writeln;
      write('USUARIO, ENTRE COM SUA MENSAGEM'); writeln;                   /**/
      readln(LINHA);                                                /**/
      if LINHA <> '   ' then
        writeln('USUARIO 1 ENVIOU PARA 2 A MENSAGEM',LINHA)
    end                                                                 /**/
  else                          /**/
    begin                                   /**/
      writeln;
      writeln('                                          LADO   2   ');
      writeln; writeln;
      write('USUARIO, ENTRE COM SUA MENSAGEM'); writeln;                   /**/
      readln(LINHA);                                                /**/
      if LINHA <> '   ' then
        writeln('USUARIO 2 ENVIOU PARA 1 A MENSAGEM',LINHA)
    end;                                                                /**/
  x:=LINHA                                  /**/
end;                                                                       /**
        /**/
procedure IOABRA_SERVICE(var P1VAR : P1TYPE);                              /**/
```

```
  external;                                                          /**/
                                                                     /**/
procedure USER_PROC;                                        /**/
  external;                                                          /**/
                                                                     /**/
procedure SAP_MANAG;                                               /**/
  external;                                                          /**/
                                                                     /**/
procedure INICIALIZACAO;                                           /**/
                                                                     /**/
begin                                                              /**/
  ACABOU(.false.):=false;                                          /**/
  ACABOU(.true.):=false                                            /**/
end;                                                               /**/
                                                                     /**/
procedure OUT; external;

procedure OUT;

var
  mens_pending : integer;
  S : S1TYPE;

procedure RENDEZVOUS;


var
  SAVEPROCESS : P1TYPE;
  SAVECHANNEL : C1TYPE;

begin
/*  writeln('pOvar.ident = ', pOvar->.ident);*/
/*  writeln('cOvar.ident = ', cOvar->.ident);*/
  SAVEPROCESS:=POVAR;
  SAVECHANNEL:=COVAR;
  POVAR:=COVAR->.TARGET.PROC;
  COVAR:=COVAR->.TARGET.CHAN;
/*  writeln('pOvar.ident = ', pOvar->.ident);*/
/*  writeln('cOvar.ident = ', cOvar->.ident);*/
  if POVAR->.IDENT = 'USER_PROC ' then                                 /**/
    USER_PROC                                                 /**/
  else                                                               /**/
    if POVAR->.IDENT = 'SAP_MANAG ' then                          /**/
      SAP_MANAG;                                                /**/
  COVAR:=SAVECHANNEL;
  POVAR:=SAVEPROCESS;
/*  writeln('pOvar.ident = ', pOvar->.ident);*/
/*  writeln('cOvar.ident = ', cOvar->.ident)*/
end;

begin
  with COVAR->.TARGET.CHAN-> do
    if QUEUED then
      begin
        if head = nil then
```

```
            begin
              head := s0var;
              s0var->.next := nil
            end
          else
            begin
              s := head;
              head := s0var;
              s0var->.next := s
            end
        end
      else
        RENDEZVOUS
end;

procedure SYSTEM_INIT;

var
    px      : p1type;
    P       : P1TYPE;
    I       : I1TYPE;
    N       : integer;
    TITLE,
    FIRST,
    OK      : boolean;

begin
  P:=nil;
  IOABRA_SERVICE(P);
  POVAR:=P->.REFINEMENT;
  dispose(P);
  P:=POVAR;
  TITLE:=true;
  OK:=false;
  N:=0;
  repeat
    N:=N+1;
    COVAR:=P->.CHANLIST;
    while COVAR <> nil do
      begin
        if COVAR->.TARGET.PROC <> nil then
          if COVAR->.IDENT > 0 then
            begin
              if TITLE then
                begin
                  TITLE:=false;
/*                writeln;
                  writeln(' *** Dangling connections ***');
                  writeln;
                  writeln(' process        channel index');
                  writeln(' seqnr/id        seqnr   intval');  */
                end;
        /*      write(N:3,'/',P->.IDENT,COVAR->.IDENT:6);    */
              I:=COVAR->.INDLIST;
              if I <> nil then
```

```
                begin
        /*        write('    (');    */
                  FIRST:=true;
                  repeat
                    if FIRST then
                      FIRST:=false
                    else
              /*      write(',');
                    write(I->.IDENT:1);  */
                    I:=I->.NEXT;
                  until I = nil;
        /*        write(')')  */
                end;
        /*    writeln   */
            end;
          COVAR:=COVAR->.NEXT
        end;
      if P->.NEXT <> nil then
        P:=P->.NEXT
      else
        OK:=true
    until OK;
/*writeln;
    writeln('-----------------------------------------------');
    writeln;    */
    if not TITLE then
      begin
        P->.NEXT:=POVAR;
        COVAR:=POVAR->.CHANLIST
      end;
    pOvar->.ident := 'user_proc1';
    pOvar := pOvar->.next;
    pOvar->.ident := 'sap_managA';
    pOvar := pOvar->.next;
    pOvar->.ident := 'sap_managB';
    pOvar := pOvar->.next;
    pOvar->.ident := 'user_proc2';
    pOvar := pOvar->.next;
/*px := pOvar;
    repeat
      writeln(' pOvar.ident = ', pOvar->.ident);
      pOvar := pOvar->.next
    until pOvar = px  */
end;

/* procedure PURGE_SINAL; external;

procedure PURGE_SINAL;

var
  s : sltype;

begin
  if sOvar = cOvar->.head then
    cOvar->.head := nil
```

```
  else
    begin
      s := c0var->.head;
      repeat
        s := s->.next
      until s->.next = s0var;
      s->.next := nil
    end
end;  */

procedure SCHEDULER;

label
  1;

procedure REC_SINAL(d0var : s1type);

var
  s : s1type;

begin
  if d0var = nil then
    begin
      c0var->.head := s0var;
      s0var->.next := nil
    end
  else
    begin
      s:= d0var;
      while s->.next <> nil do
        s:=s->.next;
      s->.next:=s0var;
      s0var->.next:=nil
    end
end;

procedure GETSIGNAL;

var
   s : s1type;

begin
  s0var := c0var->.head;
  s := nil;
  while s0var->.next <> nil do
    begin
      s := s0var;
      s0var := s0var->.next
    end;
  if s = nil then
    c0var->.head := nil
  else
    s->.next := nil;
end;
```

```
procedure MOD_1;

begin
  c0var := p0var->.chanlist;
  c0var := c0var->.next;
  if c0var->.head = nil then
    begin
      c0var := p0var->.chanlist;
      new(s0var, COUSER_PROC,R1ANY);
      s0var->.TOUSER_PROC := R1ANY;
      USER_PROC
    end
  else
    begin
      GETSIGNAL;
      USER_PROC
    end
end;

procedure MOD_2;

var
  c1, c2 : c1type;
  sm : s1type;
  nm : integer;
  t0var,d0var : s1type;

begin
  c0var := p0var->.chanlist;
  c1 := c0var;
  c0var := c0var->.next;
  if c0var->.head <> nil then
    begin
      sm := c0var->.head;
      nm := 0;
      repeat
        nm := nm+1;
        sm := sm->.next
      until sm = nil; writeln;
      writeln('existe(m) ',nm:1,' mensagem(ns) na porta User'); writeln
      GETSIGNAL;
      t0var := s0var;
      d0var := c0var->.head;
      SAP_MANAG;
      if t0var = s0var then
        REC_SINAL(d0var)
    end
  else
    begin
      c0var := c0var->.next;
      if c0var->.head = nil then
        begin
          writeln;
          writeln('AS PORTAS User E INch NAO TEM MENSAGENS');
```

```
          writeln('E GERADO UM PSEUDO-SINAL PARA HABILITAR');
          writeln('TRANSICOES ESPONTANEAS');writeln;
          c0var := c1;
          new(s0var,COSAP_MANAG,R2ANY);
          s0var->.TOSAP_MANAG := R2ANY;
          SAP_MANAG
        end
      else
        begin
          sm := c0var->.head;
          nm := 0;
          repeat
            nm := nm+1;
            sm := sm->.next
          until sm = nil; writeln;
          writeln('existe(m) ',nm:1,' mensagem(ns) na porta INch');
          writeln('UMA MENSAGEM NA PORTA INch VAI SER TRATADA');writeln;
          GETSIGNAL;
          t0var := s0var;
          d0var := c0var->.head;
          SAP_MANAG;
          if t0var = s0var then
            REC_SINAL(d0var)
        end
    end
end;

begin
  writeln;writeln;writeln;writeln;writeln;writeln;writeln;
  writeln('                              **********************************');
  writeln;
  writeln('                              SIMULACAO DO  SERVICO  ABRACADABRA');
  writeln;
  writeln('                              **********************************');
  writeln; writeln;writeln;writeln;writeln;writeln;writeln;writeln;
  repeat
1:  writeln;
    writeln('        ** OS NOMES DOS PROCESSOS SAO: **        ');
    writeln('                - user_proc1; ');
    writeln('                - user_proc2; ');
    writeln('                - sap_managA; ');
    writeln('                - sap_managB; ');
    writeln;
    writeln(' ** DIGITE O NOME DO PROCESSO A SER EXECUTADO ** ');
    readln(MODULO);
    if (MODULO <> 'user_proc1') AND (MODULO <> 'user_proc2') AND
       (MODULO <> 'sap_managA') AND (MODULO <> 'sap_managB') then
      begin
        writeln(' ** NOME DO PROCESSO EMITIDO INCORRETAMENTE ** ');
        writeln; writeln(' ** TENTE NOVAMENTE ** ');
        goto 1
      end;
    if MODULO = 'user_proc1' then
      begin
        if p0var->.ident = 'user_proc1' then
```

```
          MOD_1
        else
          begin
            repeat
              pOvar := pOvar->.next
            until pOvar->.ident = 'user_proc1';
            MOD_1
          end
      end;
    if MODULO = 'user_proc2' then
      begin
        if pOvar->.ident = 'user_proc2' then
          MOD_1
        else
          begin
            repeat
              pOvar := pOvar->.next
            until pOvar->.ident = 'user_proc2';
            MOD_1
          end
      end;
    if MODULO = 'sap_managA' then
      begin
        if pOvar->.ident = 'sap_managA' then
          MOD_2
        else
          begin
            repeat
              pOvar := pOvar->.next
            until pOvar->.ident = 'sap_managA';
            MOD_2
          end
      end;
    if MODULO = 'sap_managB' then
      begin
        if pOvar->.ident = 'sap_managB' then
          MOD_2
        else
          begin
            repeat
              pOvar := pOvar->.next
            until pOvar->.ident = 'sap_managB';
            MOD_2
          end
      end
  until ACABOU(.true.) and ACABOU(.false.);                    /**/
writeln;
writeln('                              O SISTEMA FOI DESCONECTADO');
writeln;writeln
end;

begin
  TERMIN(INPUT);
  TERMOUT(OUTPUT);
  INICIALIZACAO;                                               /**/
```

```
  SYSTEM_INIT;
  SCHEDULER
end.
```

ANEXO 4: Especificação formal do protocolo Abracadabra em Estelle/83

```
module ABRACADABRA;
end ABRACADABRA;

refinement ABRA_SERVICE for ABRACADABRA;

  CONST
    MAXLIN = 80;
    N = 5;
    P = 10;

  type
    seqtype = 0..1;
    UserDTyp = pached array (.1..MAXLIN.) of char;
    PDUType = (CR, CC, DT, AK, DR, DC);
    UnitDTyp = record
                 PDU : PDUType;
                 seqno : seqtype;
                 UData : UserDTyp;
               end;

  channel MSAP (user, provider);
    by user:
      UnitReq(UnitData : UnitDTyp);
    by provider:
      UnitInd(UnitData :UnitDTyp);
  end MSAP;

  module CMS;
    A, B : MSAP(provider);
  end CMS;

  process CMS_PROC for CMS;

    function PERDA_NO_MEIO : boolean;

        begin
          if random(clock) > 0.9 then
            PERDA_NO_MEIO := true
          else
            PERDA_NO_MEIO := false
        end;

    trans
      when A.UnitReq
      provided not PERDA_NO_MEIO
      begin
        out B.UnitInd(UnitData);
      end;
```

```
  when B.UnitReq
  provided not PERDA_NO_MEIO
  begin
    out A.UnitInd(UnitData);
  end;

end CMS_PROC;

module ABRA
  Medium : MSAP(user);
end ABRA;

refinement ABRA_BODY for ABRA;

channel USAP(user, provider);
  by user:
    ConReq;
    ConResp;
    DatReq(UserData : UserDTyp);
    DisReq;
  by provider:
    ConInd;
    ConConf;
    DatInd(UserData : UserDTyp);
    DisInd;
end USAP;

channel PEERCODE(all);
  by all:
    CR;
    CC;
    DT(seq : seqtype; UserData : UserDTyp);
    AK(seq : seqtype);
    DR;
    DC;
end PEERCODE;

module USER;
  IpU : USAP(user);
end USER;

process USER_PROC for USER;

var
  state : (IDLE, CALLING, CONNECTED);
  x : UserDTyp;
  ACABOU1, ACABOU2 : boolean;

procedure GETPRIM(var prim : UserDTyp);
primitive;
procedure GETCH(var x : UserDTyp);
primitive;
procedure WRITE_DATA(UserData : UserDTyp);
primitive;
```

```
initialize
  begin
    state := IDLE;
  end;


trans
  from IDLE
  fo CALLING
  begin
    out IpU.ConReq;
  end;

  when IpU.ConInd
  to IDLE
  begin
    out IpU.DisReq;
  end;

  when IpU.ConInd
  from IDLE
  to CONNECTED
  begin
    out IpU.ConResp;
  end;

  when IpU.ConInd
  from CALLING
  to CONNECTED
  begin
    out IpU.ConResp;
  end;

  when IpU.DatInd
  from CONNECTED
  to CONNECTED
  begin
    WRITE_DATA(UserData);
  end;

  when IpU.DisInd
  from CONNECTED
  to IDLE
  begin
  end;

  when IpU.ConConf
  from CALLING
  to CONNECTED
  begin
    GETCH(x);
    while (x <> ' ') do
      begin
        out IpU.DatReq(x);
        GETCH(x);
```

```
      end;
  end;



  when IpU.DisInd
  from CALLING
  to IDLE
  begin
  end;

end USER_PROC;

module STATION;
  IpUser : USAP(provider);
  Peer : PEERCODE(all);
end STATION;

process STATION_PROC for STATION;

queued IpUser, Peer;

var
  state : (CLOSED, CRSENT, CRRECV, ESTAB, DRSENT);
  sending : boolean;
  sendseq, recvseq : seqtype;
  oldsendseq : seqtype;
  CRretranremaining : integer;
  DTretranremaining : integer;
  DRretranremaining : integer;
  olddata : UserDTyp;
  DTorAK : boolean;

procedure INITVAR;
  begin
    sending := false;
    sendseq := 0;
    recvseq := 0;
    CRretranremaining := -1;
    DTretranremaining := -1;
    DRretranremaining := -1;
    DTorAK := false;
  end;

initialize
  begin
    state := CLOSED;
  end;

trans
  when IpUser.ConReq
  from CLOSED
  to CRSENT
  begin
```

```
  INITVAR;
  out Peer.CR;
  CRretranremaining := N-1;
end;


when Peer.CC
from CRSENT
to ESTAB
begin
  out IpUser.ConConf;
  CRretranremaining := -1;
end;

when Peer.CR
from CRSENT
to ESTAB
begin
  out IpUser.ConConf;
  CRretranremaining := -1;
end;

when Peer.DR
from CRSENT
to CLOSED
begin
  out IpUser.DisInd;
  CRretranremaining := -1;
end;

when IpUser.DisReq
from CRSENT
to DRSENT
begin
  out Peer.DR;
  CRretranremaining := -1;
  DRretranremaining := N-1;
end;

provided CRretranremaining > 0
/*delay(P)*/
from CRSENT
to CRSENT
begin
  CRretranremaining := CRretranremaining - 1;
  out Peer.CR;
end;

provided CRretranremaining = 0
from CRSENT
to DRSENT
/*delay(P)*/
begin
  out IpUser.DisInd;
  out Peer.DR;
```

```
  CRretranremaining := -1;
  DRretranremaining := N-1;
end;



when Peer.CR
from CLOSED
to CRRECV
begin
  INITVAR
  out IpUser.ConInd;
end;

when IpUser.ConResp
from CRRECV
to ESTAB
begin
  out Peer.CC;
end;

when IpUser.DisReq
from CRRECV
to CLOSED
begin
  out Peer.DR;
end;

when Peer.DR
from CRRECV
to CLOSED
begin
  out IpUser.DisInd;
  out Peer.DC;
end;

when IpUser.DatReq
from ESTAB
to ESTAB
provided not sending
begin
  olddata := UserData;
  out Peer.DT(sendseq, olddata);
  oldsendseq := sendseq;
  sendseq := (sendseq+1) mod 2;
  sending := true;
  DTretranremaining := N-1;
end;

when Peer.AK
from ESTAB
to ESTAB
begin
  if seq = sendseq then
    begin
```

```
      sending := false;
      DTretranremaining := -1;
      DTorAK := true;
    end
end;

when Peer.AK
from ESTAB
to DRSENT
begin
  if seq <> sendseq then
    begin
      out IpUser.DisInd;
      out Peer.DR;
      DTorAK := true;
      DTretranremaining := -1;
      DRretranremaining := N-1;
    end
end;

when Peer.DT
from ESTAB
to ESTAB
begin
  if seq = recvseq then
    begin
      out IpUser.DatInd(UserData);
      recvseq := (recvseq+1) mod 2;
    end;
  out Peer.AK(recvseq);
  DTorAK := true;
end;

when Peer.CR
from ESTAB
to ESTAB
provided not DTorAK
begin
  out Peer.CC;
end;

when Peer.CR
from ESTAB
to DRSENT
provided DTorAK
begin
  out IpUser.DisInd;
  out Peer.DR;
  DTretranremaining := -1;
  DRretranremaining := N-1;
end;

when Peer.CC
from ESTAB
to DRSENT
```

```
begin
  out IpUser.DisInd;
  out Peer.DR;
  DTretranremaining := -1;
  DRretranremaining := N-1;
end;

when Peer.DC
begin
  out IpUser.DisInd;
  out Peer.DR;
  DTretranremaining := -1;
  DRretranremaining := N-1;
end;

provided DTretranremaining > 0
from ESTAB
to ESTAB
/*delay(P)*/
begin
  DTretranremaining := DTretranremaining - 1;
  out Peer.DT(oldsendseq, olddata);
end;

provided DTretranremaining = 0
/*delay(P)*/
begin
  out IpUser.DisInd;
  out Peer.DR;
  DTretranremaining := -1;
  DRretranremaining := N-1;
end;

when IpUser.DisReq
from ESTAB
to DRSENT
begin
  out Peer.DR;
  DTretranremaining := -1;
  DRretranremaining := N-1;
end;

when Peer.DC
from DRSENT
to CLOSED
begin
  DRretranremaining := -1;
end;

when Peer.DR
from DRSENT
to CLOSED
begin
  DRretranremaining := -1;
end;
```

```
when Peer.DR
from ESTAB
to CLOSED
begin
  out IpUser.DisInd;
  out Peer.DC;
  DTretranremaining := -1;
end;

when Peer.DR
from CLOSED
to CLOSED
begin
  out Peer.DC
end;

provided DRretranremaining > 0
from DRSENT
to DRSENT
/*delay(P)*/
begin
  DRretranremaining := DRretranremaining - 1;
  out Peer.DR;
end;

provided DRretranremaining = 0
from DRSENT
to CLOSED
 /*delay(P)*/
begin
  DRretranremaining := -1;
end;

when Peer.CR
from CRRECV
to CRRECV
begin
end;

when Peer.CC
from CLOSED
to CLOSED
begin
end;

when Peer.CC
from CRRECV
to CRRECV
begin
```

```
end;



when Peer.CC
from DRSENT
to DRSENT
begin
end;

when Peer.DT
from CLOSED
to CLOSED
begin
end;

when Peer.DT
from CRSENT
to CRSENT
begin
end;

when Peer.DT
from CRRECV
to CRRECV
begin
end;

when Peer.DT
from DRSENT
to DRSENT
begin
end;

when Peer.AK
from CLOSED
to CLOSED
begin
end;

when Peer.AK
from CRSENT
to CRSENT
begin
end;

when Peer.AK
from CRRECV
to CRRECV
begin
end;

when Peer.AK
```

```
from DRSENT
to DRSENT
begin
end;


when Peer.DC
from CLOSED
to CLOSED
begin
end;

when Peer.DC
from CRSENT
to CRSENT
begin
end;

when Peer.DC
from CRRECV
to CRRECV
begin
end;

when IpUser.ConReq
from CRSENT
to CRSENT
begin
end;

when IpUser.ConReq
from CRRECV
to CRRECV
begin
end;

when IpUser.ConReq
from ESTAB
to ESTAB
begin
end;

when IpUser.ConReq
from DRSENT
to DRSENT
begin
end;

when IpUser.ConResp
from CLOSED
to CLOSED
begin
end;

when IpUser.ConResp
```

```
from CRSENT
to CRSENT
begin
end;


when IpUser.ConResp
from ESTAB
to ESTAB
begin
end;

when IpUser.ConResp
from DRSENT
to DRSENT
begin
end;

when IpUser.DatReq
from CLOSED
to CLOSED
begin
end;

when IpUser.DatReq
from CRSENT
to CRSENT
begin
end;

when IpUser.DatReq
from CRRECV
to CRRECV
begin
end;

when IpUser.DatReq
from DRSENT
to DRSENT
begin
end;

when IpUser.DisReq
from CLOSED
to CLOSED
begin
end;

when IpUser.DisReq
from DRSENT
to DRSENT
begin
end;

end STATION_PROC;
```

```
module TRANSCODE;
  IpUp : PEERCODE(all);
  Down : MSAP(user);
end TRANSCODE;

process TRANSCODE_PROC for TRANSCODE;

var
  SDU : UnitDTyp;

procedure BuildCR(var SDU:UnitDTyp);
  begin
    SDU.PDU := CR;
  end;

procedure BuildCC(var SDU:UnitDTyp);
  begin
    SDU.PDU := CC;
  end;

procedure  BuildDT(seq:seqtype;   data:UserDTyp;   var  SDU  :
                                                    UnitDType)
  begin
    SDU.PDU := DT;
    SDU.seqno  := seq;
    SDU.Udata := data;
  end;

procedure BuildAK(seq : seqtype; var SDU : UnitDTyp);
  begin
    SDU.PDU := AK;
    SDU.seqno := seq;
  end;

procedure BuildDR(var SDU : UnitDTyp);
  begin
    SDU.PDU := DR;
  end;

procedure BuildDC(var SDU : UnitDTyp);
  begin
    SDU.PDU := DC;
  end;

trans
  when IpUp.CR
  begin
    BuildCR(SDU);
    out Down.UnitReq(SDU);
  end;

  when IpUp.CC
  begin
    BuildCC(SDU);
```

```
  out Down.UnitReq(SDU);
end;



when IpUp.DT
begin
  BuildDT(seq, userdata, SDU);
  out Down.UnitReq(SDU);
end;

when IpUp.AK
begin
  BuildAK(seq, SDU);
  out Down.UnitReq(SDU);
end;

when IpUp.DR
begin
  BuildDR(SDU);
  out Down.UnitReq(SDU);
end;

when IpUp.DC
begin
  BuildDC(SDU);
  out Down.UnitReq(SDU);
end;

when Down.UnitInd
begin
  if (UnitData.PDU = CR) then
    out IpUp.CR
end;

when Down.UnitInd
begin
  if (UnitData.PDU = CC) then
    out IpUp.CC
end;

when Down.UnitInd
begin
  if (UnitData.PDU = DT) then
    out IpUp.DT(UnitData.seqno, UnitData.Udata);
end;

when Down.UnitInd
begin
  if (UnitData.PDU = AK) then
    out IpUp.AK(UnitData.seqno)
end;

when Down.UnitInd
```

```
      begin
        if (UnitData.PDU = DR) then
          out IpUp.DR
      end;


      when Down.UnitInd
      begin
        if (UnitData.PDU = DC) then
          out IpUp.DC
      end;

    end TRANSCODE_PROC;

    XC : TRANSCODE  with TRANSCODE_PROC;
    S  : STATION    with STATION_PROC;
    U  : USER       with USER_PROC;

    connect S.Peer to XC.IpUp;
            U.IpU to S.IpUser;

    replace Medium by XC.Down;

  end ABRA_BODY;

  A0 : ABRA with ABRA_BODY;
  A1 : ABRA with ABRA_BODY;
  CM : CMS  with CMS_PROC;

  connect A0.Medium to CM.A;
          A1.Medium to CM.B;

end ABRA_SERVICE;
```

ANEXO 5: Código Pascal do protocolo

```
segment prosai7;

const
  MAXLIN = 80;
  N = 5;
  P = 10;

type
  SeqType =
    0..1;
  UserDTyp =
    packed array (.1..MAXLIN.) of char;
  PduType =
    (CR,CC,DT,AK,DR,DC);
  UnitDTyp =
    record
      Pdu : PduType;
      SeqNo : SeqType;
      Udata : UserDTyp
    end;
  channel =
    (C1MSAP,C2USAP,C3PEERCODE,COUSER_PROC,COSTATION_PROC);
  S1MSAP =
    (S1UnitReq,S1UnitInd);
  S2USAP =
    (S2ConReq,S2ConResp,S2DatReq,S2DisReq,S2ConInd,S2ConConf,S2DatInd,
     S2DisInd);
  S3PEERCODE =
    (S3CR,S3CC,S3DT,S3AK,S3DR,S3DC);
  SOUSER_PROC =
    (R1ANY);
  SOSTATION_PROC =
    (R2ANY);
  S1TYPE =
    ->SOTYPE;
  SOTYPE =
    record
      NEXT : S1TYPE;
      case CHANNEL of
        C1MSAP :
          (case T1MSAP : S1MSAP of
            S1UnitReq :
              (D1UnitReq : record
                UnitData : UnitDTyp
              end;
              );
            S1UnitInd :
              (D1UnitInd : record
                UnitData : UnitDTyp
              end;
              );
```

```
  );
C2USAP :
  (case T2USAP : S2USAP of
    S2ConReq :
      ( );
    S2ConResp :
      ( );
    S2DatReq :
      (D2DatReq : record
        UserData : UserDTyp
      end;
      );
    S2DisReq :
      ( );
    S2ConInd :
      ( );
    S2ConConf :
      ( );
    S2DatInd :
      (D2DatInd : record
        UserData : UserDTyp
      end;
      );
    S2DisInd :
      ( );

  );
C3PEERCODE :
  (case T3PEERCODE : S3PEERCODE of
    S3CR :
      ( );
    S3CC :
      ( );
    S3DT :
      (D3DT : record
        seq : SeqType;
        UserData : UserDTyp
      end;
      );
    S3AK :
      (D3AK : record
        seq : SeqType
      end;
      );
    S3DR :
      ( );
    S3DC :
      ( );

  );
COUSER_PROC :
  (case TOUSER_PROC : SOUSER_PROC of
    R1ANY :
      ( );
```

```
        );
      COSTATION_PROC :
        (case TOSTATION_PROC : SOSTATION_PROC of
          R2ANY :
            ( );

        );


  end;
I1TYPE =
  ->IOTYPE;
C1TYPE =
  ->COTYPE;
P1TYPE =
  ->POTYPE;
IOTYPE =
  record
    NEXT : I1TYPE;
    IDENT : integer
  end;
AOTYPE =
  record
    PROC : P1TYPE;
    CHAN : C1TYPE
  end;
COTYPE =
  record
    IDENT : integer;
    INDLIST : I1TYPE;
    TARGET : AOTYPE
  end;
P2TYPE =
  packed array (.1..10.) of char;
PROCESS =
  (POCMS_PROC,POUSER_PROC,POSTATION_PROC,POTRANSCOD_PROC);
POTYPE =
  record
    IDENT : P2TYPE;
    CHANLIST : C1TYPE;
    NEXT : P1TYPE;
    case PROCESS of
      POCMS_PROC :
        ( );
      POUSER_PROC :
        (DOUSER_PROC : record
          state : (IDLE,CALLING,CONNECTED);
          x, prim : UserDTyp;
          FINAL : boolean;

        end);
      POSTATION_PROC :
        (DOSTATION_PROC : record
          state : (CLOSED,CRSENT,CRRECV,ESTAB,DRSENT);
```

```
            sending : boolean;
            SendSeq,RecvSeq : SeqType;
            OldSendSeq : SeqType;
            CRRetranRemaining : integer;
            DTRetranRemaining : integer;
            DRRetranRemaining : integer;
            OldData : UserDTyp;
            DTorAK : boolean;

          end);
        POTRANSCOD_PROC :
          (DOTRANSCOD_PROC : record
            SDU : UnitDTyp;
          end);


    end;

var
  POVAR : P1TYPE;
  COVAR : C1TYPE;
  SOVAR : S1TYPE;

function POFUNCTION(INDPOS : integer) : integer;

external;

procedure OUT;

external;

procedure P0PROCEDURE(IDENT : integer);

external;

procedure P1PROCEDURE(IDENT : P2TYPE; var P : P1TYPE);

external;

procedure P2PROCEDURE(IDENT : integer);

external;

procedure P3PROCEDURE(Q : boolean; IDENT : integer);

external;

procedure P4PROCEDURE(INDVAL,INDPOS : integer);

external;

procedure P5PROCEDURE(I,J : integer; P : P1TYPE);

external;
```

```
procedure P6PROCEDURE(var X : AOTYPE);
external;
procedure P7PROCEDURE(P : P1TYPE; var Q : P1TYPE);
external;
procedure P8PROCEDURE(P : P1TYPE);
external;
procedure GETPRIM(var prim : UserDTyp);
external;
procedure GETCH(var x : UserDTyp);
external;
procedure WRITE_DATA(UserData : UserDTyp);
external;
procedure SETTIMER(DELAY : integer);
external;
procedure RESETTIMER;
external;
function TIMEOUT : boolean;
external;
procedure ENCERRA_SISTEMA;
external;
procedure RECUPERA_SINAL;
external;
procedure CMS_PROC;
external;
procedure CMS_PROC;
label
  1;
var
  C1VAR : C1TYPE;
```

```
  S1VAR : S1TYPE;

 /**  function PERDA_NO_MEIO : boolean;

begin
  if random(clock ) > 0.9 then
    PERDA_NO_MEIO:=true
  else
    PERDA_NO_MEIO:=false
end;**/

begin
  C1VAR:=COVAR ;
  S1VAR:=SOVAR ;
  if COVAR->.IDENT = 1 then
    if SOVAR->.T1MSAP = S1UnitReq then
/*    if not PERDA_NO_MEIO then*/
        with SOVAR->.D1UnitReq do
          begin
            writeln;writeln('TRANSICAO 1 HABILITADA');writeln;
            begin
              POPROCEDURE(3 );
              new(SOVAR ,C1MSAP ,S1UnitInd );
              SOVAR->.T1MSAP:=S1UnitInd ;
              SOVAR->.D1UnitInd.UnitData:=UnitData ;
              OUT
            end;
writeln;writeln('o modulo CMS recebeu uma UnitReq e liberou uma ');
writeln('UnitInd ');writeln;
            goto 1
          end


  ;
  if COVAR->.IDENT = 3 then
    if SOVAR->.T1MSAP = S1UnitReq then
/*    if not PERDA_NO_MEIO then*/
        with SOVAR->.D1UnitReq do
          begin
            writeln;writeln('TRANSICAO 2 HABILITADA');writeln;
            begin
              POPROCEDURE(1 );
              new(SOVAR ,C1MSAP ,S1UnitInd );
              SOVAR->.T1MSAP:=S1UnitInd ;
              SOVAR->.D1UnitInd.UnitData:=UnitData ;
              OUT
            end;
writeln;writeln('o modulo CMS recebeu uma UnitReq e liberou uma ');
writeln('UnitInd ');writeln;
            goto 1
          end


  ;
  writeln;writeln('** O PROCESSO NAO TEM TRANSICAO HABILITADA **');
```

```
  1 : case C1VAR->.IDENT of
    1 :
      case S1VAR->.T1MSAP of
        S1UnitReq :
          dispose(S1VAR ,C1MSAP ,S1UnitReq );

      end;
    3 :
      case S1VAR->.T1MSAP of
        S1UnitReq :
          dispose(S1VAR ,C1MSAP ,S1UnitReq );

      end;

  end;

end;

procedure USER_PROC;

external;

procedure USER_PROC;

label
  1, 2, 3;

var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
  SINAL : char;

begin
  C1VAR:=C0VAR ;
  S1VAR:=S0VAR ;
  with P0VAR->.D0USER_PROC do
    begin
      if C0VAR->.IDENT = 0 then
        if S0VAR->.T0USER_PROC = R1ANY then
          if state in (.CONNECTED.) then
            begin
              writeln;writeln('TRANSICAO 1 HABILITADA');writeln;
 writeln;writeln('** CONEXAO JA ESTABELECIDA ** ');
              GETCH(x);
              while (x<>'  ') do
                begin
                  P0PROCEDURE(1 );
                  new(S0VAR ,C2USAP ,S2DatReq );
                  S0VAR->.T2USAP:=S2DatReq ;
                  S0VAR->.D2DatReq.UserData:=x ;
                  OUT;
                  GETCH(x)
                end;
 writeln;writeln('** NAO TENHO MENSAGENS P/TRANSMITIR **');
```

```
writeln;writeln(' ** DESEJA INICIAR DESCONEXAO, s/n **');
writeln;
                readln(SINAL);
                if (SINAL='s') or (SINAL='S') then
                  begin
writeln; writeln('** FINAL DA FASE DE TRANSFERENCIA DE DADOS **');
writeln; writeln('** INICIO DA FASE DE DESCONEXAO **' ); writeln;
                    state:=IDLE ;
                    begin
                      POPROCEDURE(1 );
                      new(SOVAR ,C2USAP ,S2DisReq );
                      SOVAR->.T2USAP:=S2DisReq ;
                      OUT
                    end;
                    ENCERRA_SISTEMA
                  end;
                goto 1
              end

        ;
        if COVAR->.IDENT = 0 then
          if SOVAR->.T0USER_PROC = R1ANY then
            if state in (.IDLE .) then
              begin
                writeln;writeln('TRANSICAO 2 HABILITADA');writeln;
2 :             GETPRIM(prim);
                if prim='conreq' then
                  begin
                    state:=CALLING ;
                    writeln;
                    writeln('** INICIO DA FASE DE CONEXAO  **');
                    writeln;
                    begin
                      POPROCEDURE(1 );
                      new(SOVAR ,C2USAP ,S2ConReq );
                      SOVAR->.T2USAP:=S2ConReq ;
                      OUT
                    end;
writeln; writeln(' O MODULO USUARIO LIBEROU UMA PRIMITIVA ConReq ');
writeln;
                    goto 1
                  end
                else
                  begin
                    writeln;
                    writeln('** PRIMITIVA EMITIDA INDEVIDAMENTE ** ');
                    writeln;
                    goto 2
                  end
              end

        ;
        if COVAR->.IDENT = 1 then
          if SOVAR->.T2USAP = S2ConInd then
            if state in (.IDLE .) then
```

```
              begin
                writeln;writeln('TRANSICAO 3 HABILITADA');writeln;
writeln; writeln(' o modulo USUARIO recebeu uma PRIMITIVA ConInd ');
3 :           GETPRIM(prim);
              if prim='conresp' then
                begin
                  state:=CONNECTED;
                  begin
                    POPROCEDURE(1 );
                    new(SOVAR ,C2USAP ,S2ConResp );
                    SOVAR->.T2USAP:=S2ConResp ;
                    OUT
                  end;
writeln(' o modulo USUARIO liberou uma PRIMITIVA ConResp ');writeln;
                  goto 1
                end;
              if prim='disreq' then
                begin
                  writeln;
                  writeln('** O LADO CHAMADO NAO ACEITOU A CONEXAO **');
                  writeln;
                    begin
                      POPROCEDURE(1 );
                      new(SOVAR ,C2USAP ,S2DisReq );
                      SOVAR->.T2USAP:=S2DisReq ;
                      OUT
                    end;
                  ENCERRA_SISTEMA;
                  goto 1
                end
              else
                begin
                  writeln;
                  writeln('** PRIMITIVA EMITIDA INDEVIDAMENTE ** ');
                  writeln;
                  goto 3
                end
            end

      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T2USAP = S2ConInd then
          if state in (.CALLING .) then
            begin
              writeln;writeln('TRANSICAO 4 HABILITADA');writeln;
              state:=CONNECTED ;
              begin
                POPROCEDURE(1 );
                new(SOVAR ,C2USAP ,S2ConResp );
                SOVAR->.T2USAP:=S2ConResp ;
                OUT
              end;
              goto 1
            end
```

```
      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T2USAP = S2DatInd then
          if state in (.CONNECTED .) then
            with SOVAR->.D2DatInd do
              begin
                writeln;writeln('TRANSICAO 5 HABILITADA');writeln;
                state:=CONNECTED ;
                writeln;
                writeln('O LADO CHAMADO/CHAMANTE ACUSA RECEPCAO');
                writeln('DE DADOS ATRAVES DA primitiva DatInd' );
                writeln;
                WRITE_DATA(UserData );
                goto 1
              end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T2USAP = S2DisInd then
          if state in (.CONNECTED .) then
            begin
              writeln;writeln('TRANSICAO 6 HABILITADA');writeln;
              state:=IDLE ;
              ENCERRA_SISTEMA;
              goto 1
            end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T2USAP = S2ConConf then
          if state in (.CALLING.) then
            begin
              writeln;writeln('TRANSICAO 7 HABILITADA');writeln;
writeln; writeln(' ** FINAL DA FASE DE CONEXAO ** '); writeln;
              state:=CONNECTED ;
              writeln;
              writeln('* INICIO DA FASE DE TRANSFERENCIA DE DADOS *');
              writeln;
              GETCH(x );
              while (x() '  ') do
                begin
                  POPROCEDURE(1 );
                  new(SOVAR ,C2USAP ,S2DatReq );
                  SOVAR->.T2USAP:=S2DatReq ;
                  SOVAR->.D2DatReq.UserData:=x ;
                  OUT;
                  GETCH(x)
                end;
 writeln;writeln('** NAO TENHO MENSAGENS P/TRANSMITIR **'); writeln;
              goto 1
            end

      ;
```

```
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T2USAP = S2DisInd then
          if state in (.CALLING .) then
            begin
              writeln;writeln('TRANSICAO 8 HABILITADA');writeln;
              state:=IDLE ;
              ENCERRA_SISTEMA;
              goto 1
            end

      ;
      writeln('** O PROCESSO NAO TEM TRANSICAO HABILITADA **')

    end;
  1 : case C1VAR->.IDENT of
    0 :
      case S1VAR->.TOUSER_PROC of
        R1ANY :
          dispose(S1VAR ,C0USER_PROC ,R1ANY );

      end;
    1 :
      case S1VAR->.T2USAP of
        S2ConInd :
          dispose(S1VAR ,C2USAP ,S2ConInd );
        S2DatInd :
          dispose(S1VAR ,C2USAP ,S2DatInd );
        S2DisInd :
          dispose(S1VAR ,C2USAP ,S2DisInd );
        S2ConConf :
          dispose(S1VAR ,C2USAP ,S2ConConf );

      end;

  end;

end;

procedure STATION_PROC;

external;

procedure STATION_PROC;

var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
  SALVA_PROCESSO : P1TYPE;
  ACHA_TRANS : boolean;

procedure INITVAR;

begin
  with P0VAR->.D0STATION_PROC do
    begin
```

```
        sending := false;
        SendSeq:=0 ;
        RecvSeq:=0 ;
        CRRetranRemaining:=-1 ;
        DTRetranRemaining:=-1 ;
        DRRetranRemaining:=-1 ;
        DTorAK:=false
      end
end;

procedure P1;

label
  1;

begin
  with POVAR->.DOSTATION_PROC do
    begin
      if COVAR->.IDENT = 1 then
        if SOVAR->.T2USAP = S2ConReq then
          if state in (.CLOSED .) then
            begin
              writeln;writeln('TRANSICAO 1 HABILITADA');writeln;
              state:=CRSENT ;
              INITVAR;
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C3PEERCODE ,S3CR );
                SOVAR->.T3PEERCODE:=S3CR ;
                SETTIMER(P);
writeln('o modulo STATION liberou uma PDU CR ');
                OUT
              end;
              CRRetranRemaining:=N - 1 ;
              ACHA_TRANS := true;
              goto 1
            end

      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T3PEERCODE = S3CC then
          if state in (.CRSENT .) then
            begin
              writeln;writeln('TRANSICAO 2 HABILITADA');writeln;
              state:=ESTAB ;
              RESETTIMER;
              begin
                POPROCEDURE(1 );
                new(SOVAR ,C2USAP ,S2ConConf );
                SOVAR->.T2USAP:=S2ConConf ;
writeln('o modulo STATION liberou uma PRIMITIVA ConConf ');
                OUT
              end;
              CRRetranRemaining:=-1 ;
              ACHA_TRANS:=true;
```

```
            goto 1
          end

      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T3PEERCODE = S3CR then
          if state in ( .CRSENT . ) then
            begin
              writeln;writeln('TRANSICAO 3 HABILITADA');writeln;
              state:=ESTAB ;
              RESETTIMER;
              begin
                POPROCEDURE( 1 );
                new( SOVAR ,C2USAP ,S2ConConf );
                SOVAR->.T2USAP:=S2ConConf ;
                OUT
              end;
              CRRetranRemaining:=-1 ;
writeln('o modulo STATION liberou uma PRIMITIVA ConConf');
              ACHA_TRANS := true;
              goto 1
            end

      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T3PEERCODE = S3DR then
          if state in ( .CRSENT . ) then
            begin
              writeln;writeln('TRANSICAO 4 HABILITADA');writeln;
              state:=CLOSED ;
              RESETTIMER;
              begin
                POPROCEDURE( 1 );
                new( SOVAR ,C2USAP ,S2DisInd );
                SOVAR->.T2USAP:=S2DisInd ;
                OUT
              end;
              CRRetranRemaining:=-1 ;
              ACHA_TRANS := true;
              goto 1
            end

      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T2USAP = S2DisReq then
          if state in ( .CRSENT . ) then
            begin
              writeln;writeln('TRANSICAO 5 HABILITADA');writeln;
              state:=DRSENT ;
              RESETTIMER;
              begin
                POPROCEDURE( 2 );
                new( SOVAR ,C3PEERCODE ,S3DR );
                SOVAR->.T3PEERCODE:=S3DR ;
                SETTIMER( P );
```

```
            OUT
          end;
          CRRetranRemaining:=-1 ;
          DRRetranRemaining:=N - 1 ;
          ACHA_TRANS := true;
          goto 1
        end

;
if COVAR->.IDENT = 0 then
  if SOVAR->.TOSTATION_PROC = R2ANY then
    if (CRRetranRemaining > 0) and (TIMEOUT) then
      if state in (.CRSENT .) then
        begin
          writeln;writeln('TRANSICAO 6 HABILITADA');writeln;
          state:=CRSENT ;
          CRRetranRemaining:=CRRetranRemaining - 1 ;
          begin
            POPROCEDURE(2 );
            new(SOVAR ,C3PEERCODE ,S3CR );
            SOVAR->.T3PEERCODE:=S3CR ;
            OUT
          end;
          ACHA_TRANS:=true;
          goto 1
        end

;
if COVAR->.IDENT = 0 then
  if SOVAR->.TOSTATION_PROC = R2ANY then
    if (CRRetranRemaining = 0) and (TIMEOUT) then
      if state in (.CRSENT .) then
        begin
          writeln;writeln('TRANSICAO 7 HABILITADA');writeln;
          state:=DRSENT ;
          RESETTIMER;
          begin
            POPROCEDURE(1 );
            new(SOVAR ,C2USAP ,S2DisInd );
            SOVAR->.T2USAP:=S2DisInd ;
            OUT
          end;
          begin
            POPROCEDURE(2 );
            new(SOVAR ,C3PEERCODE ,S3DR );
            SOVAR->.T3PEERCODE:=S3DR ;
            SETTIMER(P);
            OUT
          end;
          CRRetranRemaining:=-1 ;
          DRRetranRemaining:=N - 1 ;
          ACHA_TRANS:=true;
          goto 1
        end
```

```
      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T3PEERCODE = S3CR then
          if state in (.CLOSED .) then
            begin
              writeln;writeln('TRANSICAO 8 HABILITADA');writeln;
              state:=CRRECV ;
              INITVAR;
              begin
                POPROCEDURE(1 );
                new(SOVAR ,C2USAP ,S2ConInd );
                SOVAR->.T2USAP:=S2ConInd ;
writeln('o modulo STATION liberou uma PRIMITIVA ConInd ');
                OUT
              end;
              ACHA_TRANS:=true;
              goto 1
            end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T2USAP = S2ConResp then
          if state in (.CRRECV .) then
            begin
              writeln;writeln('TRANSICAO 9 HABILITADA');writeln;
              state:=ESTAB ;
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C3PEERCODE ,S3CC );
                SOVAR->.T3PEERCODE:=S3CC ;
                OUT
              end;
writeln('o modulo STATION liberou uma PDU CC ');
              ACHA_TRANS:=true;
              goto 1
            end


      ;
      if COVAR->.IDENT = 1 then
        if SOVAR->.T2USAP = S2DisReq then
          if state in (.CRRECV .) then
            begin
              writeln;writeln('TRANSICAO 10 HABILITADA');writeln;
              state:=CLOSED ;
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C3PEERCODE ,S3DR );
                SOVAR->.T3PEERCODE:=S3DR ;
                OUT
              end;
              ACHA_TRANS:=true;
              goto 1
            end
```

```
;
if COVAR->.IDENT = 2 then
  if SOVAR->.T3PEERCODE = S3DR then
    if state in (.CRRECV .) then
      begin
        writeln;writeln('TRANSICAO 11 HABILITADA');writeln;
        state:=CLOSED ;
        begin
          POPROCEDURE(1 );
          new(SOVAR ,C2USAP ,S2DisInd );
          SOVAR->.T2USAP:=S2DisInd ;
          OUT
        end;
        begin
          POPROCEDURE(2 );
          new(SOVAR ,C3PEERCODE ,S3DC );
          SOVAR->.T3PEERCODE:=S3DC ;
          OUT
        end;
        ACHA_TRANS:=true;
        goto 1
      end


;
if COVAR->.IDENT = 1 then
  if SOVAR->.T2USAP = S2DatReq then
    if state in (.ESTAB .) then
      if not Sending then
      begin
        writeln;writeln('TRANSICAO 12 HABILITADA');writeln;
        with SOVAR->.D2DatReq do
          begin
            state:=ESTAB ;
            OldData:=UserData ;
            begin
              POPROCEDURE(2 );
              new(SOVAR ,C3PEERCODE ,S3DT );
              SOVAR->.T3PEERCODE:=S3DT ;
              SOVAR->.D3DT.seq:=SendSeq ;
              SOVAR->.D3DT.UserData:=OldData ;
              SETTIMER(P);
              OUT
            end;
            OldSendSeq:=SendSeq ;
            SendSeq:=(SendSeq + 1 ) mod 2 ;
            Sending:=true ;
            DTRetranRemaining:=N - 1 ;
            ACHA_TRANS:=true;
            goto 1
          end
      end
/*    else
```

```
            begin
              with S0VAR->.D2DatReq do
                begin
writeln('MENSAGEM NAO PODE SER RETIRADA DA FILA ');
                  SALVA_PROCESSO := P0VAR;
                  RECUPERA_SINAL;
                  olddata := userdata;
                  begin
                    new( S0VAR ,C2USAP ,S2DatReq );
                    S0VAR->.T2USAP:=S2DatReq ;
                    S0VAR->.D2DatReq.UserData:=olddata;
                    OUT
                  end;
                  p0var := SALVA_PROCESSO;
                  c0var := p0var->.chanlist;
                  new( S0VAR,C0STATION_PROC,R2ANY );
                  S0VAR->.T0STATION_PROC := R2ANY;
                  ACHA_TRANS:=true;
                  goto 1
                end
            end */
      ;
      if C0VAR->.IDENT = 2 then
        if S0VAR->.T3PEERCODE = S3AK then
          if state in ( .ESTAB .) then
            with S0VAR->.D3AK do
              begin
                writeln;writeln('TRANSICAO 13 HABILITADA');writeln;
                state:=ESTAB ;
                if Seq = SendSeq then
                  begin
                    RESETTIMER;
                    Sending:=false ;
                    DTRetranRemaining:=-1 ;
                    DTorAK:=true
                  end
                ;
                ACHA_TRANS:=true;
                dispose( s0var );
                goto 1
              end

      ;
      if C0VAR->.IDENT = 2 then
        if S0VAR->.T3PEERCODE = S3AK then
          if state in ( .ESTAB .) then
            with S0VAR->.D3AK do
              begin
                writeln;writeln('TRANSICAO 14 HABILITADA');writeln;
                state:=DRSENT ;
                if Seq <> SendSeq then
                  begin
                    RESETTIMER;
                    begin
                      POPROCEDURE( 1 );
```

```
                    new( SOVAR ,C2USAP ,S2DisInd );
                    SOVAR->.T2USAP:=S2DisInd ;
                    OUT
                  end;
                  begin
                    POPROCEDURE( 2 );
                    new( SOVAR ,C3PEERCODE ,S3DR );
                    SOVAR->.T3PEERCODE:=S3DR ;
                    SETTIMER( P );
                    OUT
                  end;
                  DTorAK:=true ;
                  DTRetranRemaining:=-1 ;
                  DRRetranRemaining:=N - 1
                end
              ;
              ACHA_TRANS:=true;
              goto 1
            end

;
if COVAR->.IDENT = 2 then
  if SOVAR->.T3PEERCODE = S3DT then
    if state in ( .ESTAB .) then
      with SOVAR->.D3DT do
        begin
          writeln;writeln('TRANSICAO 15 HABILITADA');writeln;
          state:=ESTAB ;
          if Seq = RecvSeq then
            begin
              begin
                POPROCEDURE( 1 );
                new( SOVAR ,C2USAP ,S2DatInd );
                SOVAR->.T2USAP:=S2DatInd ;
                SOVAR->.D2DatInd.UserData:=UserData ;
                OUT
              end;
              RecvSeq:=( RecvSeq + 1 ) mod 2 ;

            end
          ;
          begin
            POPROCEDURE( 2 );
            new( SOVAR ,C3PEERCODE ,S3AK );
            SOVAR->.T3PEERCODE:=S3AK ;
            SOVAR->.D3AK.seq:=RecvSeq ;
            OUT;
          end;
          DTorAK:=true ;
          ACHA_TRANS:=true;
          goto 1
        end

;
```

```
if COVAR->.IDENT = 2 then
  if SOVAR->.T3PEERCODE = S3CR then
    if state in (.ESTAB .) then
      if not DTorAK then
        begin
          writeln;writeln('TRANSICAO 16 HABILITADA');writeln;
          state:=ESTAB ;
          begin
            POPROCEDURE(2 );
            new(SOVAR ,C3PEERCODE ,S3CC );
            SOVAR->.T3PEERCODE:=S3CC ;
            OUT
          end;
          ACHA_TRANS:=true;
          goto 1
        end


;
if COVAR->.IDENT = 2 then
  if SOVAR->.T3PEERCODE = S3CR then
    if state in (.ESTAB .) then
      if DTorAK then
        begin
          writeln;writeln('TRANSICAO 17 HABILITADA');writeln;
          state:=DRSENT ;
          RESETTIMER;
          begin
            POPROCEDURE(1 );
            new(SOVAR ,C2USAP ,S2DisInd );
            SOVAR->.T2USAP:=S2DisInd ;
            OUT
          end;
          begin
            POPROCEDURE(2 );
            new(SOVAR ,C3PEERCODE ,S3DR );
            SOVAR->.T3PEERCODE:=S3DR ;
            SETTIMER(P);
            OUT
          end;
          DTRetranRemaining:=-1 ;
          DRRetranRemaining:=N - 1 ;
          ACHA_TRANS:=true;
          goto 1
        end

;
if COVAR->.IDENT = 2 then
  if SOVAR->.T3PEERCODE = S3CC then
    if state in (.ESTAB .) then
      begin
        writeln;writeln('TRANSICAO 18 HABILITADA');writeln;
        state:=DRSENT ;
        RESETTIMER;
```

```
                begin
                  POPROCEDURE( 1 );
                  new( SOVAR ,C2USAP ,S2DisInd );
                  SOVAR->.T2USAP:=S2DisInd ;
                  OUT
                end;
                begin
                  POPROCEDURE( 2 );
                  new( SOVAR ,C3PEERCODE ,S3DR );
                  SOVAR->.T3PEERCODE:=S3DR ;
                  SETTIMER( P );
                  OUT
                end;
                DTRetranRemaining:=-1 ;
                DRRetranRemaining:=N - 1 ;
                ACHA_TRANS:=true;
                goto 1
              end

        ;
        if COVAR->.IDENT = 2 then
          if SOVAR->.T3PEERCODE = S3DC then
            if state in ( .ESTAB .) then
            begin
              writeln;writeln('TRANSICAO 19 HABILITADA');writeln;
              state:=DRSENT;
              RESETTIMER;
              begin
                POPROCEDURE( 1 );
                new( SOVAR ,C2USAP ,S2DisInd );
                SOVAR->.T2USAP:=S2DisInd ;
                OUT
              end;
              begin
                POPROCEDURE( 2 );
                new( SOVAR ,C3PEERCODE ,S3DR );
                SOVAR->.T3PEERCODE:=S3DR ;
                SETTIMER( P );
                OUT
              end;
              DTRetranRemaining:=-1 ;
              DRRetranRemaining:=N - 1 ;
              ACHA_TRANS:=true;
              goto 1
            end

          ;
      end;

  1 : ;
end;

procedure P2;

label
```

```
  1;

begin
  with POVAR->.DOSTATION_PROC do
    begin
      if ACHA_TRANS then
        goto 1
      ;
      if COVAR->.IDENT = 0 then
        if SOVAR->.TOSTATION_PROC = R2ANY then
          if (DTRetranRemaining > 0) and (TIMEOUT) then
            if state in (.ESTAB .) then
              begin
                writeln;writeln('TRANSICAO 20 HABILITADA');writeln;
                state:=ESTAB ;
                DTRetranRemaining:=DTRetranRemaining - 1 ;
                begin
                  POPROCEDURE(2 );
                  new(SOVAR ,C3PEERCODE ,S3DT );
                  SOVAR->.T3PEERCODE:=S3DT ;
                  SOVAR->.D3DT.seq:=OldSendSeq ;
                  SOVAR->.D3DT.UserData:=OldData ;
                  OUT
                end;
                goto 1
              end

      ;
      if COVAR->.IDENT = 0 then
        if SOVAR->.TOSTATION_PROC = R2ANY then
          if (DTRetranRemaining = 0) and (TIMEOUT) then
            begin
              writeln;writeln('TRANSICAO 21 HABILITADA');writeln;
              RESETTIMER;
              begin
                POPROCEDURE(1 );
                new(SOVAR ,C2USAP ,S2DisInd );
                SOVAR->.T2USAP:=S2DisInd ;
                OUT
              end;
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C3PEERCODE ,S3DR );
                SOVAR->.T3PEERCODE:=S3DR ;
                SETTIMER(P);
                OUT
              end;
              DTRetranRemaining:=-1 ;
              DRRetranRemaining:=N - 1 ;
              goto 1
            end


      ;
      if COVAR->.IDENT = 1 then
```

```
        if SOVAR->.T2USAP = S2DisReq then
          if state in (.ESTAB .) then
            begin
              writeln;writeln('TRANSICAO 22 HABILITADA');writeln;
              state:=DRSENT ;
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C3PEERCODE ,S3DR );
                SOVAR->.T3PEERCODE:=S3DR ;
                SETTIMER(P);
                OUT
              end;
              DTRetranRemaining:=-1 ;
              DRRetranRemaining:=N - 1 ;
            endgoto 1


      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T3PEERCODE = S3DC then
          if state in (.DRSENT .) then
            begin
              writeln;writeln('TRANSICAO 23 HABILITADA');writeln;
              state:=CLOSED ;
              RESETTIMER;
              DRRetranRemaining:=-1 ;
writeln('** FINAL DA FASE DE DISCONEXAO **');
              dispose(sOvar);
              goto 1
            end


      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T3PEERCODE = S3DR then
          if state in (.DRSENT .) then
            begin
              writeln;writeln('TRANSICAO 24 HABILITADA');writeln;
              state:=CLOSED ;
              RESETTIMER;
              DRRetranRemaining:=-1 ;
              dispose(sOvar);
              goto 1
            end


      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T3PEERCODE = S3DR then
          if state in (.ESTAB .) then
            begin
              writeln;writeln('TRANSICAO 25 HABILITADA');writeln;
              state:=CLOSED ;
              begin
                POPROCEDURE(1 );
```

```
            new( SOVAR ,C2USAP ,S2DisInd );
            SOVAR->.T2USAP:=S2DisInd ;
            OUT
          end;
          begin
            POPROCEDURE( 2 );
            new( SOVAR ,C3PEERCODE ,S3DC );
            SOVAR->.T3PEERCODE:=S3DC ;
            OUT
          end;
          DTRetranRemaining:=-1 ;
          goto 1
        end

;
if COVAR->.IDENT = 2 then
  if SOVAR->.T3PEERCODE = S3DR then
    if state in ( .CLOSED .) then
      begin
        writeln;writeln('TRANSICAO 26 HABILITADA');writeln;
        state:=CLOSED ;
        begin
          POPROCEDURE( 2 );
          new( SOVAR ,C3PEERCODE ,S3DC );
          SOVAR->.T3PEERCODE:=S3DC ;
          OUT
        end;
        goto 1
      end


;
if COVAR->.IDENT = 0 then
  if SOVAR->.TOSTATION_PROC = R2ANY then
    if ( DRRetranRemaining > 0) and ( TIMEOUT ) then
      if state in ( .DRSENT .) then
        begin
          writeln;writeln('TRANSICAO 27 HABILITADA');writeln;
          state:=DRSENT ;
          DRRetranRemaining:=DRRetranRemaining - 1 ;
          begin
            POPROCEDURE( 2 );
            new( SOVAR ,C3PEERCODE ,S3DR );
            SOVAR->.T3PEERCODE:=S3DR ;
            OUT
          end;
          goto 1
        end


;
if COVAR->.IDENT = 0 then
  if SOVAR->.TOSTATION_PROC = R2ANY then
    if ( DRRetranRemaining = 0) and ( TIMEOUT ) then
```

```
      if state in (.DRSENT .) then
        begin
          writeln;writeln('TRANSICAO 28 HABILITADA');writeln;
          state:=CLOSED ;
          RESETTIMER;
          DRRetranRemaining:=-1 ;
          dispose(s0var);
          goto 1
        end


;
if C0VAR->.IDENT = 2 then
  if S0VAR->.T3PEERCODE = S3CR then
    if state in (.CRRECV .) then
      begin
        writeln;writeln('TRANSICAO 29 HABILITADA');writeln;
        state:=CRRECV ;
        dispose(s0var);
        goto 1
      end


;
if C0VAR->.IDENT = 2 then
  if S0VAR->.T3PEERCODE = S3CC then
    if state in (.CLOSED .) then
      begin
        writeln;writeln('TRANSICAO 30 HABILITADA');writeln;
        state:=CLOSED ;
        dispose(s0var);
        goto 1
      end


;
if C0VAR->.IDENT = 2 then
  if S0VAR->.T3PEERCODE = S3CC then
    if state in (.CRRECV .) then
      begin
        writeln;writeln('TRANSICAO 31 HABILITADA');writeln;
        state:=CRRECV ;
        dispose(s0var);
        goto 1
      end


;
if C0VAR->.IDENT = 2 then
  if S0VAR->.T3PEERCODE = S3CC then
    if state in (.DRSENT .) then
      begin
        writeln;writeln('TRANSICAO 32 HABILITADA');writeln;
        state:=DRSENT ;
```

```
          dispose( s0var );
          goto 1
        end


;
if COVAR->.IDENT = 2 then
  if S0VAR->.T3PEERCODE = S3DT then
    if state in ( .CLOSED .) then
      with S0VAR->.D3DT do
        begin
          writeln;writeln('TRANSICAO 33 HABILITADA');writeln;
          state:=CLOSED ;
          dispose( s0var );
          goto 1
        end


;
if COVAR->.IDENT = 2 then
  if S0VAR->.T3PEERCODE = S3DT then
    if state in ( .CRSENT .) then
      with S0VAR->.D3DT do
        begin
          writeln;writeln('TRANSICAO 34 HABILITADA');writeln;
          state:=CRSENT ;
          dispose( s0var );
          goto 1
        end


;
if COVAR->.IDENT = 2 then
  if S0VAR->.T3PEERCODE = S3DT then
    if state in ( .CRRECV .) then
      with S0VAR->.D3DT do
        begin
          writeln;writeln('TRANSICAO 35 HABILITADA');writeln;
          state:=CRRECV ;
          dispose( s0var );
          goto 1
        end


;
if COVAR->.IDENT = 2 then
  if S0VAR->.T3PEERCODE = S3DT then
    if state in ( .DRSENT .) then
      with S0VAR->.D3DT do
        begin
          writeln;writeln('TRANSICAO 36 HABILITADA');writeln;
          state:=DRSENT ;
          dispose( s0var );
          goto 1
        end
```

```
;
if COVAR->.IDENT = 2 then
  if SOVAR->.T3PEERCODE = S3AK then
    if state in (.CLOSED .) then
      with SOVAR->.D3AK do
        begin
          writeln;writeln('TRANSICAO 37 HABILITADA');writeln;
          state:=CLOSED ;
          dispose(s0var);
          goto 1
        end


;
if COVAR->.IDENT = 2 then
  if SOVAR->.T3PEERCODE = S3AK then
    if state in (.CRSENT .) then
      with SOVAR->.D3AK do
        begin
          state:=CRSENT ;
          dispose(s0var);
          goto 1
        end


;
if COVAR->.IDENT = 2 then
  if SOVAR->.T3PEERCODE = S3AK then
    if state in (.CRRECV .) then
      with SOVAR->.D3AK do
        begin
          writeln;writeln('TRANSICAO 38 HABILITADA');writeln;
          state:=CRRECV ;
          dispose(s0var);
          goto 1
        end


;
if COVAR->.IDENT = 2 then
  if SOVAR->.T3PEERCODE = S3AK then
    if state in (.DRSENT .) then
      with SOVAR->.D3AK do
        begin
          writeln;writeln('TRANSICAO 39 HABILITADA');writeln;
          state:=DRSENT ;
          dispose(s0var);
          goto 1
        end


;
if COVAR->.IDENT = 2 then
```

```
  if S0VAR->.T3PEERCODE = S3DC then
    if state in (.CLOSED .) then
      begin
        writeln;writeln('TRANSICAO 40 HABILITADA');writeln;
        state:=CLOSED ;
        dispose(s0var);
        goto 1
      end


;
if C0VAR->.IDENT = 2 then
  if S0VAR->.T3PEERCODE = S3DC then
    if state in (.CRSENT .) then
      begin
        writeln;writeln('TRANSICAO 41 HABILITADA');writeln;
        state:=CRSENT ;
        dispose(s0var);
        goto 1
      end


;
if C0VAR->.IDENT = 2 then
  if S0VAR->.T3PEERCODE = S3DC then
    if state in (.CRRECV .) then
      begin
        writeln;writeln('TRANSICAO 42 HABILITADA');writeln;
        state:=CRRECV ;
        dispose(s0var);
        goto 1
      end


;
if C0VAR->.IDENT = 1 then
  if S0VAR->.T2USAP = S2ConReq then
    if state in (.CRSENT .) then
      begin
        writeln;writeln('TRANSICAO 43 HABILITADA');writeln;
        state:=CRSENT ;
        dispose(s0var);
        goto 1
      end


;
if C0VAR->.IDENT = 1 then
  if S0VAR->.T2USAP = S2ConReq then
    if state in (.CRRECV .) then
      begin
        writeln;writeln('TRANSICAO 44 HABILITADA');writeln;
        state:=CRRECV ;
        dispose(s0var);
        goto 1
```

```
        end


;
if COVAR->.IDENT = 1 then
  if SOVAR->.T2USAP = S2ConReq then
    if state in ( .ESTAB . ) then
      begin
        writeln;writeln('TRANSICAO 45 HABILITADA');writeln;
        state:=ESTAB ;
        dispose( s0var );
        goto 1
      end


;
if COVAR->.IDENT = 1 then
  if SOVAR->.T2USAP = S2ConReq then
    if state in ( .DRSENT . ) then
      begin
        state:=DRSENT ;
        writeln;writeln('TRANSICAO 46 HABILITADA');writeln;
        dispose( s0var );
        goto 1
      end


;
if COVAR->.IDENT = 1 then
  if SOVAR->.T2USAP = S2ConResp then
    if state in ( .CLOSED . ) then
      begin
        writeln;writeln('TRANSICAO 47 HABILITADA');writeln;
        state:=CLOSED ;
        dispose( s0var );
        goto 1
      end


;
if COVAR->.IDENT = 1 then
  if SOVAR->.T2USAP = S2ConResp then
    if state in ( .CRSENT . ) then
      begin
        writeln;writeln('TRANSICAO 48 HABILITADA');writeln;
        state:=CRSENT ;
        dispose( s0var );
        goto 1
      end


;
if COVAR->.IDENT = 1 then
  if SOVAR->.T2USAP = S2ConResp then
    if state in ( .ESTAB . ) then
```

```
        begin
          writeln;writeln('TRANSICAO 49 HABILITADA');writeln;
          state:=ESTAB ;
          dispose(s0var);
          goto 1
        end


;
if COVAR->.IDENT = 1 then
  if S0VAR->.T2USAP = S2ConResp then
    if state in (.DRSENT .) then
      begin
        writeln;writeln('TRANSICAO 50 HABILITADA');writeln;
        state:=DRSENT ;
        dispose(s0var);
        goto 1
      end


;
if COVAR->.IDENT = 1 then
  if S0VAR->.T2USAP = S2DatReq then
    if state in (.CLOSED .) then
      with S0VAR->.D2DatReq do
        begin
          writeln;writeln('TRANSICAO 51 HABILITADA');writeln;
          state:=CLOSED ;
          dispose(s0var);
          goto 1
        end


;
if COVAR->.IDENT = 1 then
  if S0VAR->.T2USAP = S2DatReq then
    if state in (.CRSENT .) then
      with S0VAR->.D2DatReq do
        begin
          writeln;writeln('TRANSICAO 52 HABILITADA');writeln;
          state:=CRSENT ;
          dispose(s0var);
          goto 1
        end


;
if COVAR->.IDENT = 1 then
  if S0VAR->.T2USAP = S2DatReq then
    if state in (.CRRECV .) then
      with S0VAR->.D2DatReq do
        begin
          writeln;writeln('TRANSICAO 53 HABILITADA');writeln;
          state:=CRRECV ;
          dispose(s0var);
```

```
                goto 1
              end


      ;
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T2USAP = S2DatReq then
          if state in ( .DRSENT .) then
            with S0VAR->.D2DatReq do
              begin
                writeln;writeln('TRANSICAO 54 HABILITADA');writeln;
                state:=DRSENT ;
                dispose( s0var );
                goto 1
              end


      ;
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T2USAP = S2DisReq then
          if state in ( .CLOSED .) then
            begin
              writeln;writeln('TRANSICAO 55 HABILITADA');writeln;
              state:=CLOSED ;
              dispose( s0var );
              goto 1
            end


      ;
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T2USAP = S2DisReq then
          if state in ( .DRSENT .) then
            begin
              writeln;writeln('TRANSICAO 56 HABILITADA');writeln;
              state:=DRSENT ;
              dispose( s0var );
              goto 1
            end


      ;

    end;
  1 : ;
/*    case C1VAR->.IDENT of
    1 :
      case S1VAR->.T2USAP of
        S2ConReq :
          dispose( S1VAR ,C2USAP ,S2ConReq );
        S2DisReq :
          dispose( S1VAR ,C2USAP ,S2DisReq );
        S2ConResp :
          dispose( S1VAR ,C2USAP ,S2ConResp );
        S2DatReq :
```

```
            dispose(S1VAR ,C2USAP ,S2DatReq );

        end;
      2 :
        case S1VAR->.T3PEERCODE of
          S3CC :
            dispose(S1VAR ,C3PEERCODE ,S3CC );
          S3CR :
            dispose(S1VAR ,C3PEERCODE ,S3CR );
          S3DR :
            dispose(S1VAR ,C3PEERCODE ,S3DR );
          S3AK :
            dispose(S1VAR ,C3PEERCODE ,S3AK );
          S3DT :
            dispose(S1VAR ,C3PEERCODE ,S3DT );
          S3DC :
            dispose(S1VAR ,C3PEERCODE ,S3DC );

        end;
      0 :
        case S1VAR->.TOSTATION_PROC of
          R2ANY :
            dispose(S1VAR ,COSTATION_PROC ,R2ANY );

        end;

    end;  */

end;

begin
  ACHA_TRANS := false;
  C1VAR:=COVAR ;
  S1VAR:=SOVAR ;
  begin
    P1;
    P2
  end
end;



procedure TRANSCOD_PROC;

external;

procedure TRANSCOD_PROC;

label
  1;

var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
```

```
procedure BuildCR(var SDU : UnitDTyp);

begin
  with POVAR->.DOTRANSCOD_PROC do
    begin
      SDU.PDU:=CR
    end
end;

procedure BuildCC(var SDU : UnitDTyp);

begin
  with POVAR->.DOTRANSCOD_PROC do
    begin
      SDU.PDU:=CC
    end
end;

procedure BuildDT(Seq : SeqType; Data : UserDTyp; var SDU : UnitDTyp);

begin
  with POVAR->.DOTRANSCOD_PROC do
    begin
      SDU.PDU:=DT ;
      SDU.SeqNo:=Seq ;
      SDU.Udata:=Data
    end
end;

procedure BuildAK(Seq : SeqType; var SDU : UnitDTyp);

begin
  with POVAR->.DOTRANSCOD_PROC do
    begin
      SDU.PDU:=AK ;
      SDU.SeqNo:=Seq
    end
end;

procedure BuildDR(var SDU : UnitDTyp);

begin
  with POVAR->.DOTRANSCOD_PROC do
    begin
      SDU.PDU:=DR
    end
end;

procedure BuildDC(var SDU : UnitDTyp);

begin
  with POVAR->.DOTRANSCOD_PROC do
    begin
      SDU.PDU:=DC
    end
```

```
end;

begin
  C1VAR:=C0VAR ;
  S1VAR:=S0VAR ;
  with P0VAR->.DOTRANSCOD_PROC do
    begin
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T3PEERCODE = S3CR then
          begin
            writeln;writeln('TRANSICAO 1 HABILITADA');writeln;
            BuildCR(SDU );
            begin
              POPROCEDURE(2 );
              new(S0VAR ,C1MSAP ,S1UnitReq );
              S0VAR->.T1MSAP:=S1UnitReq ;
              S0VAR->.D1UnitReq.UnitData:=SDU ;
              OUT
            end;
writeln;writeln('o modulo TRANSCODE recebeu uma PDU  CR e liberou ');
writeln('uma UnitReq(CR) ');writeln;
            goto 1
          end

      ;
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T3PEERCODE = S3CC then
          begin
            writeln;writeln('TRANSICAO 2 HABILITADA');writeln;
            BuildCC(SDU );
            begin
              POPROCEDURE(2 );
              new(S0VAR ,C1MSAP ,S1UnitReq );
              S0VAR->.T1MSAP:=S1UnitReq ;
              S0VAR->.D1UnitReq.UnitData:=SDU ;
              OUT
            end;
writeln;writeln('o modulo TRANSCODE recebeu uma PDU  CC e liberou ');
writeln('uma UnitReq(CC) ');writeln;
            goto 1
          end

      ;
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T3PEERCODE = S3DT then
          with S0VAR->.D3DT do
            begin
              writeln;writeln('TRANSICAO 3 HABILITADA');writeln;
              BuildDT(Seq ,UserData ,SDU );
              begin
                POPROCEDURE(2 );
                new(S0VAR ,C1MSAP ,S1UnitReq );
                S0VAR->.T1MSAP:=S1UnitReq ;
                S0VAR->.D1UnitReq.UnitData:=SDU ;
                OUT
```

```
                end;
writeln;writeln('o modulo TRANSCODE recebeu uma PDU  DT e liberou ');
writeln('uma UnitReq(DT) ');writeln;
                goto 1
              end

        ;
        if COVAR->.IDENT = 1 then
          if SOVAR->.T3PEERCODE = S3AK then
            with SOVAR->.D3AK do
              begin
                writeln;writeln('TRANSICAO 4 HABILITADA');writeln;
                BuildAK(Seq ,SDU );
                begin
                  POPROCEDURE(2 );
                  new(SOVAR ,C1MSAP ,S1UnitReq );
                  SOVAR->.T1MSAP:=S1UnitReq ;
                  SOVAR->.D1UnitReq.UnitData:=SDU ;
                  OUT
                end;
writeln;writeln('o modulo TRANSCODE recebeu uma PDU  AK e liberou ');
writeln('uma UnitReq(AK) ');writeln;
                goto 1
              end

        ;
        if COVAR->.IDENT = 1 then
          if SOVAR->.T3PEERCODE = S3DR then
            begin
              writeln;writeln('TRANSICAO 5 HABILITADA');writeln;
              BuildDR(SDU );
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C1MSAP ,S1UnitReq );
                SOVAR->.T1MSAP:=S1UnitReq ;
                SOVAR->.D1UnitReq.UnitData:=SDU ;
                OUT
              end;
writeln;writeln('o modulo TRANSCODE recebeu uma PDU  DR e liberou ');
writeln('uma UnitReq(DR) ');writeln;
              goto 1
            end

        ;
        if COVAR->.IDENT = 1 then
          if SOVAR->.T3PEERCODE = S3DC then
            begin
              writeln;writeln('TRANSICAO 6 HABILITADA');writeln;
              BuildDC(SDU );
              begin
                POPROCEDURE(2 );
                new(SOVAR ,C1MSAP ,S1UnitReq );
                SOVAR->.T1MSAP:=S1UnitReq ;
                SOVAR->.D1UnitReq.UnitData:=SDU ;
                OUT
```

```
            end;
writeln;writeln('o modulo TRANSCODE recebeu uma PDU  DC e liberou ');
writeln('uma UnitReq(DC) ');writeln;
            goto 1
          end

      ;
      if C0VAR->.IDENT = 2 then
        if S0VAR->.T1MSAP = S1UnitInd then
          with S0VAR->.D1UnitInd do
            if (UnitData.PDU = CR ) then
              begin
                writeln;writeln('TRANSICAO 7 HABILITADA');writeln;
                POPROCEDURE(1 );
                new(S0VAR ,C3PEERCODE ,S3CR );
                S0VAR->.T3PEERCODE:=S3CR ;
                OUT;
writeln;writeln('o modulo TRANSCODE recebeu uma UnitInd(CR) e liberou');
writeln(' uma PDU CR ');writeln;
                goto 1
              end

      ;
      if C0VAR->.IDENT = 2 then
        if S0VAR->.T1MSAP = S1UnitInd then
          with S0VAR->.D1UnitInd do
            if (UnitData.PDU = CC ) then
              begin
                writeln;writeln('TRANSICAO 8 HABILITADA');writeln;
                POPROCEDURE(1 );
                new(S0VAR ,C3PEERCODE ,S3CC );
                S0VAR->.T3PEERCODE:=S3CC ;
                OUT;
writeln;writeln('o modulo TRANSCODE recebeu uma UnitInd(CC) e liberou');
writeln(' uma PDU CC ');writeln;
                goto 1
              end

      ;
      if C0VAR->.IDENT = 2 then
        if S0VAR->.T1MSAP = S1UnitInd then
          with S0VAR->.D1UnitInd do
            if (UnitData.PDU = DT ) then
              begin
                writeln;writeln('TRANSICAO 9 HABILITADA');writeln;
                POPROCEDURE(1 );
                new(S0VAR ,C3PEERCODE ,S3DT );
                S0VAR->.T3PEERCODE:=S3DT ;
                S0VAR->.D3DT.seq:=UnitData.SeqNo ;
                S0VAR->.D3DT.UserData:=UnitData.UData ;
                OUT;
writeln;writeln('o modulo TRANSCODE recebeu uma UnitInd(DT) e liberou');
writeln(' uma PDU DT ');writeln;
                goto 1
              end
```

```
      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T1MSAP = S1UnitInd then
          with SOVAR->.D1UnitInd do
            if (UnitData.PDU = AK ) then
              begin
                writeln;writeln('TRANSICAO 10 HABILITADA');writeln;
                POPROCEDURE(1 );
                new(SOVAR ,C3PEERCODE ,S3AK );
                SOVAR->.T3PEERCODE:=S3AK ;
                SOVAR->.D3AK.seq:=UnitData.SeqNo ;
                OUT;
writeln;writeln('o modulo TRANSCODE recebeu uma UnitInd(AK) e liberou');
writeln(' uma PDU AK ');writeln;
                 goto 1
                end

      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T1MSAP = S1UnitInd then
          with SOVAR->.D1UnitInd do
            if (UnitData.PDU = DR ) then
              begin
                writeln;writeln('TRANSICAO 11 HABILITADA');writeln;
                POPROCEDURE(1 );
                new(SOVAR ,C3PEERCODE ,S3DR );
                SOVAR->.T3PEERCODE:=S3DR ;
                OUT;
writeln;writeln('o modulo TRANSCODE recebeu uma UnitInd(DR) e liberou')
writeln(' uma PDU DR ');writeln;
                 goto 1
                end

      ;
      if COVAR->.IDENT = 2 then
        if SOVAR->.T1MSAP = S1UnitInd then
          with SOVAR->.D1UnitInd do
            if (UnitData.PDU = DC ) then
              begin
                writeln;writeln('TRANSICAO 12 HABILITADA');writeln;
                POPROCEDURE(1 );
                new(SOVAR ,C3PEERCODE ,S3DC );
                SOVAR->.T3PEERCODE:=S3DC ;
                OUT;
writeln;writeln('o modulo TRANSCODE recebeu uma UnitInd(DC) e liberou')
writeln(' uma PDU DC ');writeln;
                 goto 1
                end

      ;

    end;
  1 : case C1VAR->.IDENT of
    1 :
```

```
        case S1VAR->.T3PEERCODE of
          S3CR :
            dispose(S1VAR ,C3PEERCODE ,S3CR );
          S3CC :
            dispose(S1VAR ,C3PEERCODE ,S3CC );
          S3DT :
            dispose(S1VAR ,C3PEERCODE ,S3DT );
          S3AK :
            dispose(S1VAR ,C3PEERCODE ,S3AK );
          S3DR :
            dispose(S1VAR ,C3PEERCODE ,S3DR );
          S3DC :
            dispose(S1VAR ,C3PEERCODE ,S3DC );

        end;
      2 :
        case S1VAR->.T1MSAP of
          S1UnitInd :
            dispose(S1VAR ,C1MSAP ,S1UnitInd );

        end;

    end;

end;

procedure IOABRA_SERVICE(var P1VAR : P1TYPE);

external;

procedure IOABRA_SERVICE;

var
  P2VAR : P1TYPE;
  A0VAR,A1VAR : A0TYPE;

procedure IOCMS_PROC(var P1VAR : P1TYPE);

begin
  new(P0VAR ,P0CMS_PROC );
  P1PROCEDURE('CMS_PROC  ' ,P1VAR );
  P3PROCEDURE(true ,1 );
  P3PROCEDURE(true ,2 );
  P3PROCEDURE(true ,3 );

end;

procedure IOABRA_BODY(var P1VAR : P1TYPE);

var
  P2VAR : P1TYPE;
  A0VAR,A1VAR : A0TYPE;

procedure IOUSER_PROC(var P1VAR : P1TYPE);
```

```
begin
  new(P0VAR ,P0USER_PROC );
  P1PROCEDURE('USER_PROC ' ,P1VAR );
  P3PROCEDURE(false ,0 );
  P3PROCEDURE(true, 1);
  with P0VAR->.D0USER_PROC do
    begin
      state:=IDLE
    end
end;

procedure IOSTATION_PROC(var P1VAR : P1TYPE);

begin
  new(P0VAR ,P0STATION_PROC );
  P1PROCEDURE('STATION_PR' ,P1VAR );
  P3PROCEDURE(false ,0 );
  P3PROCEDURE(true ,1 );
  P3PROCEDURE(true ,2 );
  with P0VAR->.D0STATION_PROC do
    begin
      state:=CLOSED
    end
end;

procedure IOTRANSCOD_PROC(var P1VAR : P1TYPE);

begin
  new(P0VAR ,P0TRANSCOD_PROC );
  P1PROCEDURE('TRANSCOD_P' ,P1VAR );
  P3PROCEDURE(true ,1 );
  P3PROCEDURE(true ,2 )
end;

begin
  P2VAR:=nil ;
  IOTRANSCOD_PROC(P2VAR );
  IOSTATION_PROC(P2VAR );
  IOUSER_PROC(P2VAR );
  P7PROCEDURE(P2VAR ,P1VAR );
  P3PROCEDURE(false ,1 );
  P5PROCEDURE(2 ,2 ,P1VAR );
  P6PROCEDURE(A0VAR );
  P5PROCEDURE(3 ,1 ,P1VAR );
  P6PROCEDURE(A1VAR );
  A0VAR.CHAN->.TARGET:=A1VAR ;
  A1VAR.CHAN->.TARGET:=A0VAR ;
  P5PROCEDURE(1 ,1 ,P1VAR );
  P6PROCEDURE(A0VAR );
  P5PROCEDURE(2 ,1 ,P1VAR );
  P6PROCEDURE(A1VAR );
  A0VAR.CHAN->.TARGET:=A1VAR ;
  A1VAR.CHAN->.TARGET:=A0VAR ;
p5procedure(0 ,1 ,p1var);
a0var.chan:=c0var;
```

```
p5procedure(3 ,2 ,p1var);
p6procedure(a0var.chan->.target);
  P8PROCEDURE(P1VAR );
end;

begin
  P2VAR:=nil ;
  IOABRA_BODY(P2VAR );
  IOABRA_BODY(P2VAR );
  IOCMS_PROC(P2VAR );
  P7PROCEDURE(P2VAR ,P1VAR );
  P5PROCEDURE(3 ,1 ,P1VAR );
  P6PROCEDURE(A0VAR );
  P5PROCEDURE(1 ,1 ,P1VAR );
  P6PROCEDURE(A1VAR );
  A0VAR.CHAN->.TARGET:=A1VAR ;
  A1VAR.CHAN->.TARGET:=A0VAR ;
  P5PROCEDURE(2 ,1 ,P1VAR );
  P6PROCEDURE(A0VAR );
  P5PROCEDURE(1 ,3 ,P1VAR );
  P6PROCEDURE(A1VAR );
  A0VAR.CHAN->.TARGET:=A1VAR ;
  A1VAR.CHAN->.TARGET:=A0VAR ;
  P8PROCEDURE(P1VAR );

end;
```

ANEXO 6: Código do núcleo de simulação para o protocolo

```
program NUCTES_7(input,output);

const
  IDENTLEN = 10;
  MAXLIN = 80;

type
  DATA_TYPE      = packed array (.1..MAXLIN.) of char;
  CHANNEL        = (COUSER_PROC,COSTATION_PROC);                    /**/
  SOUSER_PROC = (R1ANY);                                          /**/
  SOSTATION_PROC = (R2ANY);                                         /**/
  S1TYPE         = ->SOTYPE;                                          /**/
  SOTYPE         = record
                     NEXT : S1TYPE;
                     case CHANNEL of
                       COUSER_PROC :                                  /**/
                         (case TOUSER_PROC : SOUSER_PROC of          /**/
                            R1ANY :                                 /**/
                              ( )                                       /**/
                          );                                            /**/
                       COSTATION_PROC :                                 /**/
                         (case TOSTATION_PROC : SOSTATION_PROC of       /**/
                            R2ANY :                                     /**/
                              ( )                                       /**/
                          );
                   end;
  I1TYPE         = ->IOTYPE;
  C1TYPE         = ->COTYPE;
  P1TYPE         = ->POTYPE;
  IOTYPE         = record
                     NEXT  : I1TYPE;
                     IDENT : integer;
                   end;
  AOTYPE         = record
                     PROC : P1TYPE;
                     CHAN : C1TYPE;
                   end;
  COTYPE         = record
                     IDENT   : integer;
                     INDLIST : I1TYPE;
                     TARGET  : AOTYPE;
                     NEXT    : C1TYPE;
                     case QUEUED : boolean of
                       false :
                         ( );
                       true :
                         (HEAD : S1TYPE);
                   end;
  P2TYPE         = packed array (.1..10.) of char;
  POTYPE         = record
                     IDENT      : P2TYPE;
```

```
                    CHANLIST    : C1TYPE;
                    NEXT,
                    REFINEMENT : P1TYPE
                  end;

var
  POVAR           : P1TYPE;
  COVAR           : C1TYPE;
  SOVAR           : S1TYPE;
  LINHA           : packed array(.1..MAXLIN.) of char;                    /**/
  CONT_TEMPO1,                                                    /**/
  CONT_TEMPO2     : integer;
  ACABOU1,                                                                /**/
  ACABOU2         : boolean;                          /**/
  PRIMITIVA : DATA_TYPE;
  RESP            : char;
  MODULO          : p2type;

procedure WRITE_DATA(D : DATA_TYPE); external; /**/
                                                                          /**/
procedure WRITE_DATA;                                                     /**/
                                                                          /**/
begin                                                                     /**/
  write(clock:9,' usuario ');                                             /**/
  if pOvar->.ident = 'user_proc1' then                                    /**/
    write('1 recebeu de 2')                                               /**/
  else                                                                    /**/
    write('2 recebeu de 1');                                              /**/
  writeln(' a mensagem ',D)                                       /**/
end;                                                                      /**/

procedure GETPRIM(var prim : DATA_TYPE);
external;

procedure GETPRIM;

begin
  writeln;
  if pOvar->.ident = 'user_proc1' then
    begin
      writeln;
      writeln('                                                LADO  1  ');
      writeln; writeln;
      writeln('USUARIO, ENTRE COM O NOME DA PRIMITIVA')
    end
  else
    begin
      writeln;
      writeln('                                                LADO  2  ');
      writeln; writeln;
      writeln('USUARIO, ENTRE COM O NOME DA PRIMITIVA')
    end;
  writeln;
  readln(PRIMITIVA);
  if (PRIMITIVA='conreq') or (PRIMITIVA='disreq')
```

```
                              or (PRIMITIVA='conresp')
                              or (PRIMITIVA='datreq') then
      prim := PRIMITIVA
    else
      begin
        writeln; write('ERRO NO PREENCHIMENTO DA PRIMITIVA.');
        writeln(' TENTE NOVAMENTE.');
        GETPRIM(prim)
      end
end;

procedure ENCERRA_SISTEMA;  external;

procedure ENCERRA_SISTEMA;

begin
  writeln;writeln('** DESEJA ENCERRAR EXECUCAO DO SISTEMA? s/n **');
  writeln;
  readln(RESP);
  if (RESP = 's') or (RESP = 'S') then
    begin
      if p0var->.ident = 'user_proc1' then
        acabou1 := true
      else
        acabou2 := true
    end
  else
    begin
      ACABOU1 := false;
      ACABOU2 := false
    end
end;

procedure GETCH(var x : DATA_TYPE);       /**/
                                        external;                    /**/
procedure GETCH;                                                     /**/
                                                                     /**/
begin                                                                /**/
  if p0var->.ident = 'user_proc1' then                 /**/
    begin                                                              /**/
      writeln;
      writeln('                                          LADO  1   ');
      writeln; writeln;
      write('USUARIO, ENTRE COM SUA MENSAGEM'); writeln;               /**/
      readln(LINHA);                                          /**/
      if LINHA <> '   ' then
        writeln('USUARIO 1 ENVIOU PARA 2 A MENSAGEM',LINHA )
    end                                                              /**/
  else                              /**/
    begin                                     /**/
      writeln;
      writeln('                                          LADO   2   ');
      writeln; writeln;
      write('USUARIO, ENTRE COM SUA MENSAGEM'); writeln;               /**/
      readln(LINHA);                                          /**/
```

```
      if LINHA <> '   ' then
        writeln('USUARIO 2 ENVIOU PARA 1 A MENSAGEM',LINHA)
    end;                                                          /**/
  x:=LINHA                                    /**/
end;                                                              /**/
                                                                  /**/
procedure SETTIMER(DELAY : integer); external; /**/
                                                                  /**/
procedure SETTIMER;                                               /**/
                                                                  /**/
begin                                                             /**/
  if pOvar->.ident = 'station_p1' then                            /**/
    begin
      CONT_TEMPO1:=DELAY*100+clock;
writeln(' clock = ', clock);
writeln(' cont_tempo1 = ', cont_tempo1);
writeln(' acionado temporizador do LADO 1')                       /**/
    end
  else                                                            /**/
    begin
      CONT_TEMPO2:=DELAY*10000+clock;
writeln(' clock = ', clock);
writeln(' cont_tempo2 = ', cont_tempo2);
writeln(' acionado temporizador do LADO 2')                       /**/
    end
end;                                                              /**/
                                                                  /**/
procedure RESETTIMER; external;                 /**/
                                                                  /**/
procedure RESETTIMER;                                             /**/
                                                                  /**/
begin                                                             /**/
  if pOvar->.ident = 'station_p1' then                            /**/
    begin
     CONT_TEMPO1:=0;                                        /**/
  write(' desligado temporizador para mensagem enviada ');  /**/
    writeln('pelo LADO  1  ')                                     /**/
    end
  else                                                            /**/
    begin
      CONT_TEMPO2:=0;                                       /**/
  write(' desligado temporizador para mensagem enviada ');  /**/
    writeln('pelo LADO   2  ')                                    /**/
    end
end;                                                              /**/
                                                                  /**/
function TIMEOUT : boolean; external;
                                                                  /**/
function TIMEOUT;                                                 /**/
                                                                  /**/
begin                                                             /**/
  TIMEOUT:=true;                                                  /**/
  if pOvar->.ident = 'station_p1' then
    begin
      if (CONT_TEMPO1 = 0) or (CONT_TEMPO1 > clock) then
```

```
        TIMEOUT := false
      else
        begin
writeln(' clock = ',  clock);
writeln(' cont_tempo1 = ', cont_tempo1);
writeln(' TIMEOUT para mensagem enviada pelo LADO 1 ');
writeln(' MENSAGEM ENVIADA NOVAMENTE ')
        end
    end
  else                                                              /**/
    begin                                                           /**
      if (CONT_TEMPO2 = 0) or (CONT_TEMPO2 > clock) then
        TIMEOUT := false
      else
        begin
writeln(' clock = ',  clock);
writeln(' cont_tempo2 = ', cont_tempo2);
writeln(' TIMEOUT para mensagem enviada pelo LADO 2 ');
writeln(' MENSAGEM ENVIADA NOVAMENTE ')
        end
    end                                                             /**/
end;                                                                /**/

procedure RECUPERA_SINAL; external;

procedure REC_SINAL(d0var : s1type);

var
  s : s1type;

begin
  if d0var = nil then
    begin
      c0var->.head := s0var;
      s0var->.next := nil
    end
  else
    begin
      s := d0var;
      while s->.next <> nil do
        s := s->.next;
      s->.next := s0var;
      s0var->.next := nil
    end
end;

procedure IOABRA_SERVICE(var P1VAR : P1TYPE);                   /**/
  external;                                                         /**
                                                                    /**
procedure USER_PROC;
  external;                                                         /**
                                                                    /**
procedure CMS_PROC;                                               /**/
  external;                                                         /**
                                                                    /**
```

```
procedure STATION_PROC;                                                    /**/
  external;                                                                /**/
                                                                           /**/
procedure TRANSCOD_PROC;                                                   /**/
  external;                                                                /**/
                                                                           /**/
procedure INICIALIZACAO;                                                   /**/
                                                                           /**/
begin                                                                      /**/
  ACABOU1:=false;                                                 /**/
  ACABOU2:=false;                                                  /**/
  CONT_TEMPO1:=0;                                                 /**/
  CONT_TEMPO2:=0                                                   /**/
end;                                                                       /**/
                                                                           /**/
procedure OUT; external;

procedure OUT;

var
  mens_pending : integer;
  S : S1TYPE;

procedure RENDEZVOUS;


var
  SAVEPROCESS : P1TYPE;
  SAVECHANNEL : C1TYPE;

begin
/*  writeln('pOvar.ident = ', pOvar->.ident);*/
/*  writeln('cOvar.ident = ', cOvar->.ident);*/
  SAVEPROCESS:=POVAR;
  SAVECHANNEL:=COVAR;
  POVAR:=COVAR->.TARGET.PROC;
  COVAR:=COVAR->.TARGET.CHAN;
/*  writeln('pOvar.ident = ', pOvar->.ident);*/
/*  writeln('cOvar.ident = ', cOvar->.ident);*/
  if POVAR->.IDENT = 'USER_PROC ' then                                     /**/
    USER_PROC                                                     /**/
  else                                                                     /**/
    if POVAR->.IDENT = 'CMS_PROC  ' then                                  /**/
      CMS_PROC                                                           /**/
    else                                                                   /**/
      if POVAR->.IDENT = 'STATION_PR' then                                 /**/
        STATION_PROC                                                       /**/
      else                                                                 /**/
        if POVAR->.IDENT = 'TRANSCOD_P' then                               /**/
          TRANSCOD_PROC;                                                   /**/
  COVAR:=SAVECHANNEL;
  POVAR:=SAVEPROCESS;
/*  writeln('pOvar.ident = ', pOvar->.ident);*/
/*  writeln('cOvar.ident = ', cOvar->.ident)*/
end;
```

```
begin
  with COVAR->.TARGET.CHAN-> do
    if QUEUED then
      begin
        if head = nil then
          begin
            head := s0var;
            s0var->.next := nil
          end
        else
          begin
            s := head;
            head := s0var;
            s0var->.next := s
          end
      end
    else
      RENDEZVOUS
end;

procedure SYSTEM_INIT;

var
   px     : p1type;
   P      : P1TYPE;
   I      : I1TYPE;
   N      : integer;
   TITLE,
   FIRST,
   OK     : boolean;

begin
  P:=nil;
  IOABRA_SERVICE(P);
  POVAR:=P->.REFINEMENT;
  dispose(P);
  P:=POVAR;
  TITLE:=true;
  OK:=false;
  N:=0;
  repeat
    N:=N+1;
    COVAR:=P->.CHANLIST;
    while COVAR <> nil do
      begin
        if COVAR->.TARGET.PROC <> nil then
          if COVAR->.IDENT > 0 then
            begin
              if TITLE then
                begin
                  TITLE:=false;
/*                writeln;
                  writeln(' *** Dangling connections ***');
                  writeln;
```

```
                    writeln(' process        channel index');
                    writeln(' seqnr/id        seqnr  intval'); */
                  end;
        /*        write(N:3,'/',P->.IDENT,COVAR->.IDENT:6);  */
                  I:=COVAR->.INDLIST;
                  if I <> nil then
                    begin
              /*      write('    (');  */
                      FIRST:=true;
                      repeat
                        if FIRST then
                          FIRST:=false
                        else
/*                        write(',');
                        write(I->.IDENT:1);  */
                        I:=I->.NEXT;
                      until I = nil;
              /*      write(')')          */
                    end;
            /*  writeln    */
              end;
          COVAR:=COVAR->.NEXT
        end;
      if P->.NEXT <> nil then
        P:=P->.NEXT
      else
        OK:=true
    until OK;
/*writeln;
    writeln('-----------------------------------------------');
    writeln;    */
    if not TITLE then
      begin
        P->.NEXT:=POVAR;
        COVAR:=POVAR->.CHANLIST
      end;
writeln('pOvar.ident = ', pOvar->.ident);
pOvar := pOvar->.next;
    repeat
      if pOvar->.ident = 'USER_PROC ' then
        pOvar->.ident := 'user_proc1'
      else
        if pOvar->.ident = 'STATION_PR' then
          pOvar->.ident := 'station_p1'
        else
          if pOvar->.ident = 'TRANSCOD_P' then
            pOvar->.ident := 'transcod_1';
      pOvar := pOvar->.next
    until pOvar->.ident = 'USER_PROC ';
    repeat
      if pOvar->.ident = 'USER_PROC ' then
        pOvar->.ident := 'user_proc2'
      else
        if pOvar->.ident = 'STATION_PR' then
          pOvar->.ident := 'station_p2'
```

```
      else
        if p0var->.ident = 'TRANSCOD_P' then
          p0var->.ident := 'transcod_2';
    p0var := p0var->.next
  until p0var->.ident = 'CMS_PROC  ';
  p0var->.ident := 'cms_proc  '
/**     px := p0var;
  repeat
    p0var := p0var->.next;
    writeln('p0var.ident = ',  p0var->.ident)
  until p0var = px    **/
end;

procedure SCHEDULER;

label
  1;

procedure GETSIGNAL;

var
   s : sltype;

begin
  s0var := c0var->.head;
  s := nil;
  while s0var->.next <> nil do
    begin
      s := s0var;
      s0var := s0var->.next
    end;
  if s = nil then
    c0var->.head := nil
  else
    s->.next := nil
end;

procedure MOD_1;

begin
  c0var := p0var->.chanlist;
  c0var := c0var->.next;
  if c0var->.head = nil then
    begin
      c0var := p0var->.chanlist;
      new(s0var, COUSER_PROC,R1ANY);
      s0var->.TOUSER_PROC := R1ANY;
      USER_PROC
    end
  else
    begin
      GETSIGNAL;
      USER_PROC
    end
end;
```

```
procedure MOD_2;

var
  c1, c2 : citype;
  nm : integer;
  sm, t0var, d0var : sitype;

begin
  c0var := p0var->.chanlist;
  c1 := c0var;
  c0var := c0var->.next;
  c2    := c0var;
  c0var := c0var->.next;
  if c0var->.head <> nil then
    begin
      sm := c0var->.head;
      nm := 0;
      repeat
        nm := nm+1;
        sm := sm->.next
      until sm = nil;
      writeln;writeln('existe(m) ',nm:1,' mensagem(ns) na porta PEER');
      GETSIGNAL;
      t0var := s0var;
      d0var := c0var->.head;
      STATION_PROC;
      if t0var = s0var then
        REC_SINAL(d0var)
    end
  else
    begin
      c0var := c2;
      if c0var->.head = nil then
        begin
          c0var := c1;
          new(s0var,COSTATION_PROC,R2ANY);
          s0var->.TOSTATION_PROC := R2ANY;
          STATION_PROC
        end
      else
        begin
          sm := c0var->.head;
          nm := 0;
          repeat
            nm := nm+1;
            sm := sm->.next
          until sm = nil; writeln;
          writeln('existe(m) ',nm:1,' mensagem(ns) na porta USER');
          GETSIGNAL;
          t0var := s0var;
          d0var := c0var->.head;
          STATION_PROC;
          if t0var = s0var then
            REC_SINAL(d0var)
```

```
        end
    end
end;

procedure MOD_3;

var
   c3 : c1type;
   sm : s1type;
   nm : integer;

begin
  c0var := p0var->.chanlist;
  c3 := c0var;
  c0var := c0var->.next;
  if c0var->.head <> nil then
    begin
      sm := c0var->.head;
      nm := 0;
      repeat
        nm := nm+1;
        sm := sm->.next
      until sm = nil; writeln;
      writeln('existe(m) ',nm:1,' mensagem(ns) na porta Down');
      GETSIGNAL;
      TRANSCOD_PROC
    end
  else
    begin
      c0var := c3;
      if c0var->.head <> nil then
        begin
          sm := c0var->.head;
          nm := 0;
          repeat
            nm := nm+1;
            sm := sm->.next
          until sm = nil; writeln;
          writeln('existe(m) ',nm:1,' mensagem(ns) na porta Up');
          GETSIGNAL;
          TRANSCOD_PROC
        end
      else
        writeln('** NAO HA MENSAGENS NAS PORTAS **')
    end
end;

procedure MOD_4;

var
  sm : s1type;
  nm : integer;

begin
  c0var := p0var->.chanlist;
```

```
  if c0var->.head = nil then
    begin
      c0var := c0var->.next;
      c0var := c0var->.next;
      if c0var->.head <> nil then
        begin
          sm := c0var->.head;
          nm := 0;
          repeat
            nm := nm+1;
            sm := sm->.next
          until sm = nil; writeln;
          writeln('existe(m) ',nm:1,' mensagem(ns) na porta A');
          GETSIGNAL;
          CMS_PROC
        end
      else
        writeln('** NAO HA MENSAGENS NAS PORTAS **')
    end
  else
    begin
      sm := c0var->.head;
      nm := 0;
      repeat
        nm := nm+1;
        sm := sm->.next
      until sm = nil; writeln;
      writeln('existe(m) ',nm:1,' mensagem(ns) na porta B');
      GETSIGNAL;
      CMS_PROC
    end
end;

begin
  writeln;writeln;writeln;writeln;writeln;writeln;writeln;
  writeln('                              ***********************************');
  writeln;
  writeln('                              SIMULACAO DO PROTOCOLO ABRACADABRA');
  writeln;
  writeln('                              ***********************************');
  writeln; writeln;writeln;writeln;writeln;writeln;writeln;writeln;
  repeat
1:  writeln;
    writeln('                         ** OS NOMES DOS PROCESSOS SAO: ** ');
    writeln('                         - user_proc1;        - user_proc2; ');
    writeln('                         - station_p1;        - station_p2; ');
    writeln('                         - transcod_1;        - transcod_2; ');
    writeln('                                    - cms_proc              ');
    writeln;
    writeln(' ** DIGITE O NOME DO PROCESSO A SER EXECUTADO ** ');
    readln(MODULO);
    if (MODULO <> 'user_proc1') AND (MODULO <> 'user_proc2') AND
       (MODULO <> 'station_p1') AND (MODULO <> 'station_p2') AND
       (MODULO <> 'transcod_1') AND (MODULO <> 'transcod_2') AND
       (MODULO <> 'cms_proc  ') then
```

```
    begin
      writeln(' ** NOME DO PROCESSO EMITIDO INCORRETAMENTE ** ');
      writeln; writeln(' ** TENTE NOVAMENTE ** ');
      goto 1
    end;
  if MODULO = 'user_proc1' then
    begin
      if p0var->.ident = 'user_proc1' then
        MOD_1
      else
        begin
          repeat
            p0var := p0var->.next
          until p0var->.ident = 'user_proc1';
          MOD_1
        end
    end;
  if MODULO = 'user_proc2' then
    begin
      if p0var->.ident = 'user_proc2' then
        MOD_1
      else
        begin
          repeat
            p0var := p0var->.next
          until p0var->.ident = 'user_proc2';
          MOD_1
        end
    end;
  if MODULO = 'station_p1' then
    begin
      if p0var->.ident = 'station_p1' then
        MOD_2
      else
        begin
          repeat
            p0var := p0var->.next
          until p0var->.ident = 'station_p1';
          MOD_2
        end
    end;
  if MODULO = 'station_p2' then
    begin
      if p0var->.ident = 'station_p2' then
        MOD_2
      else
        begin
          repeat
            p0var := p0var->.next
          until p0var->.ident = 'station_p2';
          MOD_2
        end
    end;
  if MODULO = 'transcod_1' then
    begin
```

```
        if p0var->.ident = 'transcod_1' then
          MOD_3
        else
          begin
            repeat
              p0var := p0var->.next
            until p0var->.ident = 'transcod_1';
            MOD_3
          end
      end;
    if MODULO = 'transcod_2' then
      begin
        if p0var->.ident = 'transcod_2' then
          MOD_3
        else
          begin
            repeat
              p0var := p0var->.next
            until p0var->.ident = 'transcod_2';
            MOD_3
          end
      end;
    if MODULO = 'cms_proc  ' then
      begin
        if p0var->.ident = 'cms_proc  ' then
          MOD_4
        else
          begin
            repeat
              p0var := p0var->.next
            until p0var->.ident = 'cms_proc  ';
            MOD_4
          end
      end;

    writeln('acabou1 = ', ACABOU1);
    writeln('acabou2 = ', ACABOU2)
  until ACABOU1 and ACABOU2;                              /**/
writeln;
writeln('                              O SISTEMA FOI DESCONECTADO');
writeln;writeln
end;

begin
  TERMIN(INPUT);
  TERMOUT(OUTPUT);
  INICIALIZACAO;                                                  /**/
  SYSTEM_INIT;
  SCHEDULER
end.
```

ANEXO 7: Código Pascal das rotinas de suporte

```
segment PFAUX;

const
  IDENTLEN = 10;

type
  S1TYPE          = ->SOTYPE;
  SOTYPE          = record
                    end;
  I1TYPE          = ->IOTYPE;
  C1TYPE          = ->COTYPE;
  P1TYPE          = ->POTYPE;
  IOTYPE          = record
                      NEXT  : I1TYPE;
                      IDENT : integer;
                    end;
  AOTYPE          = record
                      PROC : P1TYPE;
                      CHAN : C1TYPE;
                    end;
  COTYPE          = record
                      IDENT   : integer;
                      INDLIST : I1TYPE;
                      TARGET  : AOTYPE;
                      NEXT    : C1TYPE;
                      case QUEUED : boolean of
                        false :
                          ( );
                        true :
                          (HEAD : S1TYPE);
                    end;
  P2TYPE          = packed array (.1..10.) of char;
  POTYPE          = record
                      IDENT       : P2TYPE;
                      CHANLIST    : C1TYPE;
                      NEXT,
                      REFINEMENT : P1TYPE
                    end;


var
  POVAR           : P1TYPE;
  COVAR           : C1TYPE;

function FOFUNCTION(INDPOS : integer) : integer; external;

function FOFUNCTION;

var
  P : I1TYPE;
```

```
begin
  P:=COVAR->.INDLIST;
  while INDPOS > 0 do
    begin
      P:=P->.NEXT;
      INDPOS:=INDPOS-1
    end;
  FOFUNCTION:=P->.IDENT
end;

procedure P0PROCEDURE(IDENT : integer); external;

procedure P0PROCEDURE;

begin
  COVAR:=P0VAR->.CHANLIST;
  while COVAR->.IDENT < IDENT do
    COVAR:=COVAR->.NEXT
end;

procedure P1PROCEDURE(IDENT : P2TYPE; var P : P1TYPE); external;

procedure P1PROCEDURE;

begin
  P0VAR->.IDENT:=IDENT;
  P0VAR->.CHANLIST:=nil;
  P0VAR->.NEXT:=P;
  P:=P0VAR;
  COVAR:=nil
end;

procedure P2PROCEDURE(IDENT : integer); external;

procedure P2PROCEDURE;

var
  P : I1TYPE;

begin
  new(P);
  P->.IDENT:=IDENT;
  P->.NEXT:=COVAR->.INDLIST;
  COVAR->.INDLIST:=P
end;

procedure P3PROCEDURE(Q : boolean ; IDENT : integer); external;

procedure P3PROCEDURE;

var
  P : C1TYPE;

begin
  if Q then
```

```
      begin
        new(P,true);
        P->.HEAD:=nil
      end
    else
      new(P,false);
    P->.QUEUED:=Q;
    P->.IDENT:=IDENT;
    P->.INDLIST:=nil;
    P->.NEXT:=nil;
    P->.TARGET.PROC:=nil;
    if COVAR <> nil then
      COVAR->.NEXT:=P
    else
      POVAR->.CHANLIST:=P;
    COVAR:=P
end;

procedure P4PROCEDURE(INDVAL,INDPOS : integer); external;

procedure P4PROCEDURE;

var
    I,
    J  : integer;
    P,
    P1 : I1TYPE;
    Q : C1TYPE;

begin
    J:=FOFUNCTION(INDPOS);
    case J <= INDVAL of
      true :
        while J < INDVAL do
          begin
            Q:=COVAR;
            COVAR:=COVAR->.NEXT;
            case Q->.IDENT <> COVAR->.IDENT of
              false :
                begin
                  P:=Q->.INDLIST;
                  P1:=COVAR->.INDLIST;
                  I:=INDPOS;
                  while I > 0 do
                    begin
                      I:=I-1;
                      case P->.IDENT <> P1->.IDENT of
                        false :
                          begin
                            P:=P->.NEXT;
                            P1:=P1->.NEXT
                          end;
                      end
                    end;
                  J:=J+1;
```

```
                case J <> P1->.IDENT of
                  false :
                end
              end
            end
          end
  end
end;

procedure P5PROCEDURE(I,J : integer ; P : P1TYPE); external;

procedure P5PROCEDURE;

begin
  if I <= 0 then
    POVAR:=P
  else
    begin
      POVAR:=P->.REFINEMENT;
      while I > 1 do
        begin
          I:=I-1;
          POVAR:=POVAR->.NEXT
        end
    end;
  P0PROCEDURE(J)
end;

procedure P6PROCEDURE(var X : A0TYPE); external;

procedure P6PROCEDURE;

begin
  if POVAR->.IDENT = 'REFINEMENT' then
    X:=COVAR->.TARGET
  else
    begin
      X.PROC:=POVAR;
      X.CHAN:=COVAR
    end
end;

procedure P7PROCEDURE(P : P1TYPE; var Q : P1TYPE); external;

procedure P7PROCEDURE;

begin
  new(POVAR);
  P1PROCEDURE('REFINEMENT',Q);
  POVAR->.REFINEMENT:=P
end;

procedure P8PROCEDURE(P : P1TYPE); external;

procedure P8PROCEDURE;
```

```
var
  Q,
  R,
  S : P1TYPE;

procedure PURGE(var P : P1TYPE);

var
  Q : P1TYPE;
  R,
  S : C1TYPE;
  T,
  U : I1TYPE;

begin
  R:=P->.CHANLIST;
  while R <> nil do
    begin
      T:=R->.INDLIST;
      while T <> nil do
        begin
          U:=T;
          T:=T->.NEXT;
          dispose(U)
        end;
      S:=R;
      R:=R->.NEXT;
      dispose(S,false)
    end;
  Q:=P;
  P:=P->.NEXT;
  dispose(Q)
end;

begin
  Q:=P->.REFINEMENT;
  R:=nil;
  S:=nil;
  while Q <> nil do
    if Q->.IDENT <> 'REFINEMENT' then
      begin
        if S = nil then
          S:=Q;
        R:=Q;
        Q:=Q->.NEXT
      end
    else
      begin
        if R <> nil then
          R->.NEXT:=Q->.REFINEMENT
        else
          R:=Q->.REFINEMENT;
        PURGE(Q);
        if R <> nil then
```

```
            begin
              while R->.NEXT <> nil do
                R:=R->.NEXT;
              R->.NEXT:=Q
            end
        end;
    P->.REFINEMENT:=S
end;
```

ANEXO 8: alguns traços verificados na simulação

PROTOCOLO

FASE DE CONEXÃO

(a) **ConReq**(1-->2), CR(2-->3), UnitReq(3-->4), UnitInd(4-->5), CR(5-->6), **ConInd**(6-->7), **ConResp**(7-->6), CC(6-->5), UnitReq(5-->4),UnitInd(4-->3), CC(3-->2),**ConConf**(2-->1);

(b) **ConReq**(1-->2), CR(2-->3), UnitReq(3-->4), UnitInd(4-->5), CR(5-->6), **ConInd**(6-->7), **DisReq**(7-->6), DR(6-->5), UnitReq(5-->4), UnitInd(4-->3), DR(3-->2), **DisInd**(2-->1, DC(2-->3), UnitReq(3-->4), UnitInd(4-->5), DC(5-->6);

(c) **ConReq**(1-->2), **ConReq**(7-->6), CR(6-->5), CR(2-->3), UnitReq(3-->4),UnitReq(5-->4),UnitInd(4-->3), UnitInd(4-->5), CR(3-->2), CR(5-->6), **ConConf**(2-->1), **ConConf**(6-->7);

(d) **ConReq**(1-->2), CR(2-->3), UnitReq(3-->4), (meio perde o 1o CR),..., CR(2-->3), UnitReq(3-->4), (meio perde o Nésimo CR), **DisInd**(2-->1), DR(2-->3), UnitReq(3-->4), UnitInd(4-->5), DR(5-->6),DC(6-->5),UnitReq(5-->4),UnitInd(4-->3),DC(3-->2);

FASE DE TRANSFERENCIA DE DADOS

(a) **DatReq**(1-->2), DT(2-->3), UnitReq(3-->4), UnitInd(4-->5), DT(5-->6), **DatInd**(6-->7), AK(6-->5), UnitReq(5-->4), UnitInd(4-->3), AK(3-->2);

(b) **DatReq**(1-->2), **DatReq**(7-->6), DT(2-->3), DT(6-->5), UnitReq(3-->4),UnitReq(5-->4), UnitInd(4-->3),UnitInd(4-->5), DT(3-->2), DT(5-->6), **DatInd**(2-->1), AK(2-->3),**DatInd**(6-->7), AK(6-->5), UnitReq(3-->4), UnitReq(5-->4), UnitInd(4-->3), UnitInd(4-->5), AK(3-->2), AK(5-->6);

(c) **DatReq**(1-->2), DT(2-->3), UnitReq(3-->4), UnitInd(4-->5), DT(5-->6), **DatInd**(6-->7), AK(6-->5), UnitReq(5-->4), (meio perde o AK), DT(2-->3) (retransmissão do DT), UnitReq(3-->4), UnitInd(4-->5), DT(5-->6), AK(6-->5) (DT duplicado), UnitReq(5-->4), UnitInd(4-->3), AK(3-->2);

FASE DE DESCONEXÃO

(a) **DisReq**(1-->2), DR(2-->3), UnitReq(3-->4), UnitInd(4-->5), DR(5-->6), **DisInd**(6-->7), DC(6-->5), UnitReq(5-->4), UnitInd(4-->3), DC3-->2);

(b) **DisReq**(1-->2), **DisReq**(7-->6), DR(2-->3), DR(6-->5), UnitReq(3-->4),UnitReq(5-->4), UnitInd(4-->3),UnitInd(4-->5), DR(3-->2), DR(5-->6);

(c) **DisReq**(1-->2), DR(2-->3), UnitReq(3-->4), UnitInd(4-->5), DR(5-->6), **DisInd**(6-->7), DC(6-->5), UnitReq(5-->4), (meio perde DC), DR(2-->3) (retransmissão do DR), UnitReq(3-->4), UnitInd(4-->5), DR(5-->6), DC(6-->5) (DR duplicado), UnitReq(5-->4), UnitInd(4-->3), DC(3-->2);

SERVIÇO

FASE DE CONEXÃO

(a) **ConReq**(1-->2),IConReqInd(2-->3),**ConInd**(3-->4),**ConResp**(4-->3), IConRespConf(3-->2), **ConConf**(2-->1);

(b) **ConReq**(1-->2),IConReqInd(2-->3),**ConInd**(3-->4), **DisReq**(4-->3), IDis(3-->2), **DisInd**(2-->1);

(c) **ConReq**(1-->2), **ConReq**(4-->3), IConReqInd(2-->3), IConReqInd (3-->2), **ConConf**(2-->1), IConRespConf(2-->3), **ConConf**(3-->4), IConRespConf(3-->2));

(d) **ConReq**(1-->2), **DisInd**(2-->1);

FASE DE TRANSFERÊNCIA DE DADOS

(a) **DatReq**(1-->2), IDat(2-->3), **DatInd**(3-->4);

(b) **DatReq**(1-->2), **DatReq**(3-->4), IDat(2-->3), IDat(3-->2), **DatInd**(2-->1), **DatInd**(3-->4);

(c) **DatReq**(1-->2), IDat(2-->3), **DatInd**(3-->4);

FASE DE DESCONEXÃO

(a) **DisReq**(1-->2), IDis(2-->3), **DisInd**(3-->4);

(b) **DisReq**(1-->2), **DisReq**(4-->3), IDis(3-->3), IDis(3-->2);

(c) **DisReq**(1-->2), IDis(2-->3), **DisInd**(3-->4);