UNIVERSIDADE FEDERAL DA PARAIBA
CENTRO DE CIENCIAS E TECNOLOGIA
CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇAO

UM COMPILADOR PARA A TECNICA DE DESCRIÇAO FORMAL Estelle/83

Edilson Ferneda

CAMPINA GRANDE
MAIO - 1988

EDILSON FERNEDA

UM COMPILADOR PARA A TECNICA DE DESCRIÇAO FORMAL Estelle/83

Dissertação apresentada ao Curso de MESTRADO EM SISTEMAS E COMPUTAÇAO da Universidade Federal da Paraiba, em cumprimento às exigências para obtenção do Grau de Mestre.

AREA DE CONCENTRAÇAO: Redes de Computadores

WANDERLEY LOPES DE SOUZA

orientador

CAMPINA GRANDE
MAIO - 1988

F362c Ferneda, Edilson
Um compilador para a tecnica de descricao formal estelle/83 / Edilson Ferneda. - Campina Grande, 1988.
183 f. : il.

Dissertacao (Mestrado em Sistemas e Computacao) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Compilador 2. Compilador Estelle/83 3. Dissertacao I. Souza, Wanderley Lopes de, Dr. II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 004.4'422(043)

UM COMPILADOR PARA A TECNICA DE DESCRIÇAO FORMAL Estelle/83

EDILSON FERNEDA

Dissertação apresentada em 09/05/88

WANDERLEY LOPES DE SOUZA - Dr.
- Presidente -

HÉLIO DE MENEZES SILVA - M.Sc
- Examinador -

STEFANIA STIUBIENER - Dra.
- Examinadora -

SOLON BENAYON DA SILVA - M.Sc
- Examinador -

CAMPINA GRANDE
MAIO - 1988

- A G R A D E C I M E N T O S -

Agradeço a todos os colegas do Departamento de Sistemas e Computação da UFPb que, direta ou indiretamente, colaboraram neste trabalho.

Agradeço também aos colegas do curso de mestrado, pela amizade.

Agradeço principalmente ao meu orientador, Prof. Wanderley Lopes de Souza, pelo incentivo e pela grande paciência.

A Lúcia, Elcio, Elza, Edmir e Edberto

S u m á r i o

1. INTRODUÇAO..... 01

2. ESPECIFICAÇAO E VALIDAÇAO DE SERVIÇOS E PROTOCOLOS..... 11

2.1 TDFs e métodos de verificação associados..... 13

2.2 TDFs em vias de padronização..... 21

3. Estelle/83: UMA TDF BASEADA EM MEFE..... 23

3.1 A arquitetura de um sistema..... 23

3.2 Especificação de um protocolo em Estelle/83..... 26

4. O COMPILADOR Estelle/83..... 33

4.1 Arquitetura do compilador..... 34

4.2 A construção do compilador..... 39

4.2.1 Os analisadores léxico e sintático..... 41

4.2.2 O gerador de código..... 53

5. IMPLEMENTAÇAO E VALIDAÇAO DE UM PROTOCOLO ATRAVES DO COMPILADOR Estelle/83..... 68

6. APLICAÇOES DO COMPILADOR Estelle/83 EM SISTEMAS DE SIMULAÇAO..... 77

6.1 Simulador para a validação de especificação..... 77

6.2 Simulador para a análise de desempenho..... 82

7. CONCLUSAO..... 85

REFERENCIAS BIBLIOGRAFICAS..... 89

ANEXO A: Sintaxe da TDF Estelle/83..... 96

ANEXO B: Subconjunto da sintaxe da linguagem Pascal/VS utilizado..... 103

ANEXO C: Especificação de um protocolo em Estelle/83..... 108

ANEXO D: Implementação gerada a partir da especificação do Anexo C..... 113

ANEXO E: Núcleo de exploração para a implementação apresentada no Anexo D..... 124

ANEXO F: Especificação de um sistema para a análise do protocolo apresentado no Anexo C (meio ideal)..... 130

ANEXO G: Implementação do sistema apresentado no Anexo F..... 135

ANEXO H: Especificação de um sistema para a análise do protocolo apresentado no Anexo C (meio com perdas).. 147

ANEXO I: Implementação do sistema apresentado no Anexo H..... 152

ANEXO J: Núcleo de exploração para as implementações apresentadas nos Anexos G e I..... 164

ANEXO K: Rotinas de suporte para a criação e manipulação das estruturas de dados..... 172

ANEXO L: Resultado da análise da simulação com meio ideal.... 178

ANEXO M: Resultado da análise da simulação com meio com perdas..... 180

ANEXO N: Manual de utilização do compilador Estelle/83..... 182

Lista de figuras

1.1 : Uma rede de computadores..... 01

1.2 : Modelo Básico de Referência OSI/ISO..... 02

1.3 : Relação entre as classes do protocolo de transporte.... 06

1.4 : Especificação dos protocolos de uma camada de um sistema descrito hierarquicamente..... 07

2.1 : Ciclo de desenvolvimento de um protocolo..... 11

2.2 : Exemplo do emprego de MEF como TDF..... 14

2.3 : Exemplo do emprego de Rede de Petri como TDF..... 16

2.4 : Exemplo do emprego de Linguagem Formal como TDF..... 17

2.5 : Exemplo do emprego de linguagem de programação como TDF..... 18

2.6 : Exemplo do emprego de técnica hibrida como TDF..... 20

3.1 : Especificação de um serviço de comunicação..... 24

3.2 : Refinamento de um módulo..... 25

3.3 : Estrutura de uma transição..... 27

3.4 : Exemplo de arquitetura de um sistema de transmissão de mensagens..... 28

3.5 : Exemplo de especificação de canais..... 28

3.6 : Exemplo de especificação abstrata de módulos..... 29

3.7 : Exemplo de refinamento de um módulo em submódulos..... 29

3.8 : Exemplo de especificação de um refinamento..... 30

3.9 : Exemplo de especificação do comportamento do módulo PROTOCOLO..... 31

4.1 : Etapas e procedimentos necessários para se alcançar um código executável..... 35

4.2 : Estrutura do compilador Estelle/83..... 36

4.3 : Estrutura interna do pré-processador Estelle/83..... 37

4.4 : Subconjunto das cláusulas Prolog responsáveis pelo gerenciamento da execução do compilador Estelle/83..... 41

4.5 : Exemplo de regra de produção da BNF da TDF Estelle/83.. 42

4.6 : Cláusula, em Prolog, que traduz uma regra de produção da BNF da TDF Estelle/83..... 42

4.7 : Subconjunto das cláusulas Prolog que implementam o predicado **terminal**..... 43

4.8 : Subconjunto das cláusulas Prolog que implementam o analisador léxico do compilador Estelle/83..... 48

4.9 : Exemplo de ação semântica..... 45

4.10 : Exemplo de teste de contexto..... 46

4.11 : Exemplo de estruturas internas Estelle/83 para especificações de canais..... 47

4.12 : Exemplo de estruturas internas Estelle/83 para especificações de módulos..... 47

4.13 : Exemplo de estruturas internas Estelle/83 para especificações de refinamentos..... 49

4.14 : Exemplo de estruturas internas Estelle/83 para especificações de processos..... 51

4.15 : Subconjunto das cláusulas Prolog que implementam a função de geração de código..... 53-54

4.16 : Exemplo de estrutura interna Pascal..... 55

4.17 : Subconjunto das cláusulas Prolog responsáveis pela geração da sequência de itens léxicos da implementação Pascal..... 57

4.18 : Subconjunto das cláusulas Prolog responsáveis pela gravação da implementação Pascal..... 58

4.19 : Declarações de tipos de dados e de variáveis de um segmento Pascal gerado pelo pré-processador Estelle/83. 60

4.20 : Declarações de funções e procedimentos de um segmento Pascal gerado pelo pré-processador Estelle/83..... 62

4.21 : Estruturas de dados que representam as contruções arquiteturais do exemplo utilizado..... 64

4.22 : Corpo de um procedimento, relativo às transições de um processo, gerado pelo pré-processador Estelle/83.... 66

5.1 : Descrição dos módulos e das interações (mensagens e primitivas de baixo nivel) do protocolo EXEMPLO..... 69

5.2 : Módulos do sistema a ser simulado para a anàlise dinâmica do protocolo EXEMPLO..... 70

5.3 : Especificação do comportamento do meio de transmissão ideal..... 70

5.4 : Especificação do comportamento do meio de transmissão sujeito a perdas de informação..... 71

5.5 : Núcleo de exploração utilizado nas análises..... 72-73

6.1 : Especificação de um módulo de observação usando um método formal semelhante ao Estelle/83..... 80

6.2 : Definição dos observadores para a obtenção do "tracing"..... 81

6.3 : Sistema a ser simulado para a análise de desempenho.... 83

Lista de abreviaturas

ABNT : Associação Brasileira de Normas Técnicas.

BNF : "Backus Naur Form".

CCITT : "Comité Consultatif International Télégrafique et Téléfonique".

CCS : "Calculus of Communicating Systems".

Estelle : "Extended State Transition Language".

FTAM : "File Transfer, Access and Management".

ISO : "International Standards Organization".

Lotos : "Language of Temporal Ordering Specification".

MEF : Máquina de Estado Finita.

MEFE : Máquina de Estado Finita Estendida.

NBS : "National Bureau of Standards".

OSI : "Open System Interconnection".

PAS : Ponto de Acesso ao Serviço.

PS : Primitiva de Serviço.

RM-OSI : "Basic Reference Model - Open Systems Interconnection".

SDL : "Specification and Description Language".

TDF : Técnica de Descrição Formal.

UFPb : Universidade Federal da Paraiba.

UM COMPILADOR PARA A TECNICA DE DESCRIÇAO FORMAL Estelle/83

RESUMO

Neste trabalho é apresentado um compilador, em Prolog, para a técnica de descrição formal (TDF) "Extended state transition language (Estelle)", versão de 1983, desenvolvida pela "International Standards Organization (ISO)".

Essa ferramenta permite a obtenção semi-automática de implementações de protocolos (especificados em Estelle/83), facilitando a sua legibilidade e transportabilidade. Uma implementação semi-automática, assim obtida, é uma referência confiável tanto para a padronização de implementações (do mesmo protocolo) pertencentes a ambientes computacionais homogêneos, como para a análise do comportamento de implementações desenvolvidas manualmente (desde que elas sejam derivadas da mesma especificação).

O compilador Estelle/83 pode também ser utilizado para realizar análises estáticas e dinâmicas de especificações. Além disso, ele pode ser inserido em ambientes amigáveis de simulção, que visem a validação de especificações e/ou a análise de desempenho de sistemas descritos em Estelle/83.

A COMPILER FOR THE FORMAL DESCRIPTION TECHNIQUE Estelle/83

ABSTRACT

This work presents a compiler written in Prolog for the formal description technique (FDT) "Extended state transition language (Estelle)", 1983 version, developed by the "International Standars Organization (ISO)".

This tool permits the semi-automatic generation of protocol implementations (specified in Estelle/83), facilitating its legibility and transportability. Such an implementation, obtained in this way, is a reliable reference for normalizing implementations (from the same protocol) belonging to homogeneous computational environments, and for analizing the behavior of manually developed implementations (providing its are derived from the same specification).

The Estelle/83 compiler may also be used for performing static and dynamic analysis of specifications. Besides this, it can be inserted in simulation friendly environments for specification validations and/or for performance analysis of systems described in Estelle/83.

1. INTRODUÇÃO

A necessidade de otimizar a utilização dos recursos computacionais associada ao aumento constante dos benefícios fornecidos pelos serviços de comunicação, viabilizaram o compartilhamento desses recursos e consequentemente o acesso a informações distribuídas geograficamente, através das assim chamadas **redes de computadores** (Fig. 1.1) [Tane 81].

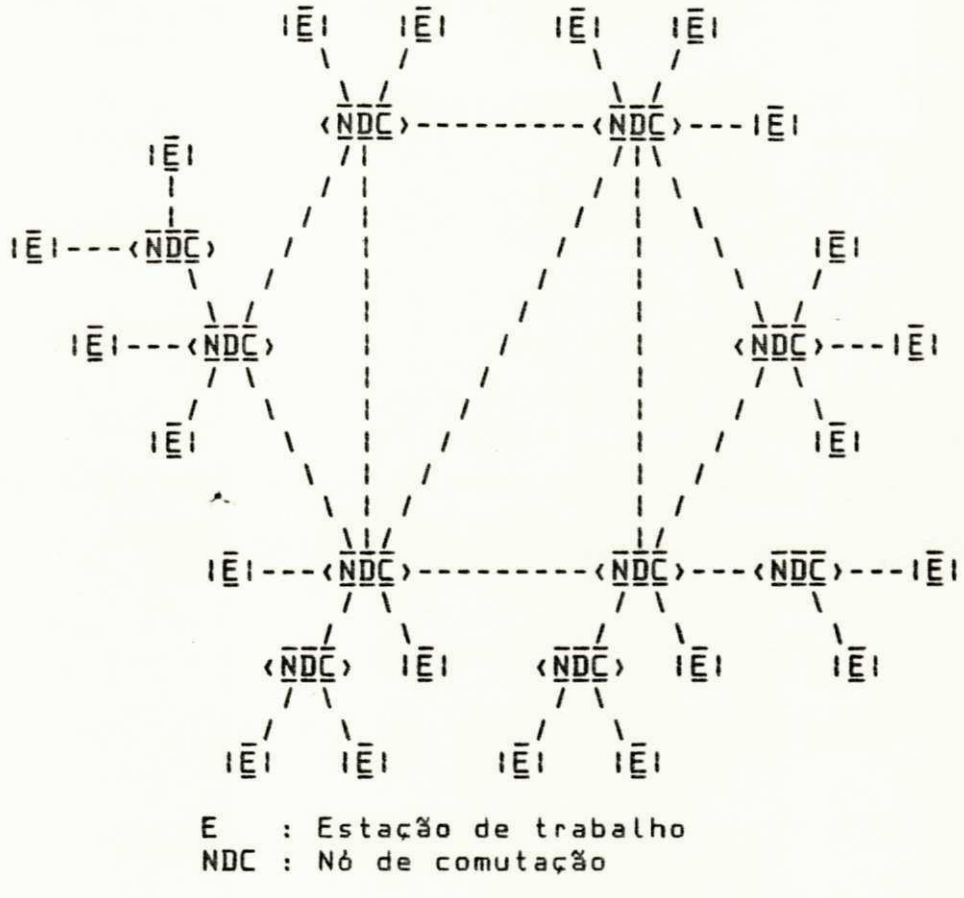

Figura 1.1: Uma rede de computadores.

Em função da sua abrangência, em termos de distância, as redes de computadores são classificadas em duas grandes categorias: **redes locais**, cuja abrangência máxima está em torno de 1 Km, e **redes a longa distância**.

A fim de facilitar a compreensão dos problemas inerentes à comunicação entre os elementos integrantes de um sistema de tal porte e a fim de facilitar a **especificação** das funções que uma rede de computadores deve desempenhar, especialistas da área propuseram uma estruturação dessas funções em camadas (ou niveis) sobrepostos hierarquicamente, dando origem ao **Modelo Básico de Referência para Interconexão de Sistemas Abertos** (RM-OSI) (Fig. 1.2) [Zimm 80].

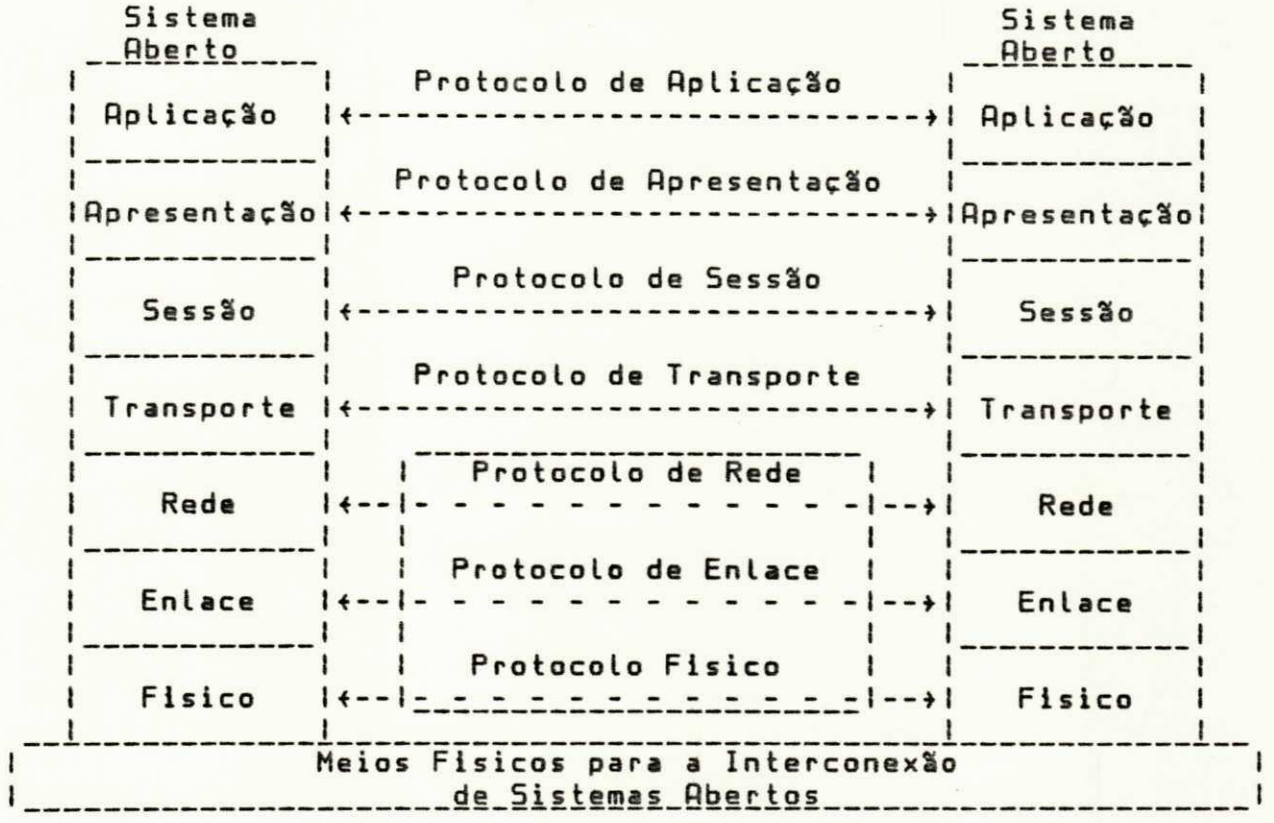

Figura 1.2: Modelo Básico de Referência OSI/ISO.

O modelo acima e as funções a serem desempenhadas por cada camada são definidos pela "International Standards Organization (ISO)" em [ISO 83a] e pelo "Comité Consultatif International Télégraphique et Téléphonique (CCITT)" em [CCIT 84]. Neste trabalho serão empregados os padrões e a nomenclatura definidos

pela ISO, utilizando-se, na medida do possível, as traduções já normalizadas pela Associação Brasileira de Normas Técnicas (ABNT).

A arquitetura do RM-OSI é dividida em sete camadas, cada uma delas com um conjunto de funções bem definidas. Essas camadas proporcionam uma melhoria progressiva dos serviços de comunicação, permitindo que os processos de aplicação, através desses serviços, possam cooperar entre si.

A seguir é dado uma descrição de cada uma das camadas do RM-OSI:

(a) **camada de aplicação**: é o nível mais alto da arquitetura OSI, sendo que as outras camadas existem apenas para lhe dar suporte. Ela serve diretamente às aplicações dos usuários finais, faz o gerenciamento dessas aplicações e o gerenciamento das atividades de comunicação do sistema. Além dos protocolos de uso geral, cada usuário pode definir nessa camada mais alguns protocolos de uso particularizado;

(b) **camada de apresentação**: visa compatibilizar as representações das estruturas de dados referenciadas pelos processos de aplicação comunicantes e dos próprios dados que são transferidos entre esses processos. Com essas transformações, puramente sintáticas, adapta-se formas particulares de manipulação de informação pertinentes a tais processos. É responsável também, quando necessário, pela segurança da informação, pela compressão de dados e pelo oferecimento de um "terminal virtual";

(c) **camada de sessão**: fornece os meios necessários para que duas entidades de apresentação pares possam organizar e sincronizar o diálogo e gerenciar a troca de dados. A camada de sessão também preocupa-se com as necessidades das aplicações, na medida em que gerencia a alternância do envio de informação entre os processos comunicantes;

(d) **camada de transporte**: faz a recuperação dos erros inerentes à perda ou desordenamento das informações, que transitam nas camadas inferiores, e a otimização da relação custo/beneficio dos serviços de comunicação. Assim, torna transparente às camadas superiores as caracteristicas da subrede de comunicação e garante a transferência de dados de maneira correta e eficiente;

(e) **camada de rede**: fornece os meios necessários para a troca de unidades de dados de serviço de rede entre entidades pares de transporte, através das conexões de rede. Isso é feito estabelecendo-se um "circuito virtual", onde transitam dados corretos e corretamente ordenados, ou é feito através de "datagrama", onde a ordem de entrega dos dados não é garantida. Executa também a função de repasse e roteamento dessas unidades de dados através da subrede;

(f) **camada de enlace**: garante a confiabilidade das unidades de dados de serviço de enlace, trocadas através do meio fisico de comunicação (seja ele qual for), e permite à camada de rede controlar a interconexão de circuitos de dados dentro da camada física. Faz também, quando necessário, o controle do compartilhamento do meio fisico de transmissão e

(g) **camada física**: provê os meios mecânicos, funcionais e os procedimentos necessários para ativar, manter e desativar conexões físicas, para a transmissão de bits entre entidades de enlace de dados.

Nem todos os sistemas abertos representam a origem ou o destino dos dados. Alguns sistemas abertos agem apenas como repetidores, repassando os dados para outros sistemas abertos.

O estabelecimento e a consolidação de padrões envolvem não só questões técnicas mas também considerações político-mercadológicas. Os padrões para protocolos de comunicação têm sido bem aceitos pelos fabricantes que atuam em teleinformática. Isso é consequência, em parte, da cooperação entre a ISO e o CCITT. Em relação aos protocolos OSI, relativos às camadas inferiores, foi aproveitada a recomendação X.25 do CCITT já existente [CCIT 77].

A primeira padronização realmente efetuada pela ISO foi a do protocolo de transporte [ISO 84a, ISO 84b]. O primeiro objetivo do Transporte foi compensar as diferenças entre as redes desenvolvidas, a fim de fornecer um serviço homogêneo ao nível superior [LoFr 87]. Assim sendo, a ISO definiu cinco classes para esse protocolo, com diferentes níveis de complexidade (Fig. 1.3):

(a) **classe 0** - é a mais simples e deve ser utilizada com redes de alta qualidade.

(b) **classe 1** - contém as funções da classe 0 e funções adicionais de reação aos erros sinalizados pelo nível inferior.

(c) classe 2 - também deve ser utilizada com redes de alta qualidade e além das funções da classe 0, oferece, entre outros serviços, as possibilidades de multiplexação das conexões de transporte e de envio de dados expressos.

(d) classe 3 - além das funcionalidades da classe 2, oferece a detecção de erros básicos.

(e) classe 4 - além das funcionalidades oferecidas pela classe 3, oferece também a recuperação de erros relativos à transferência de dados.

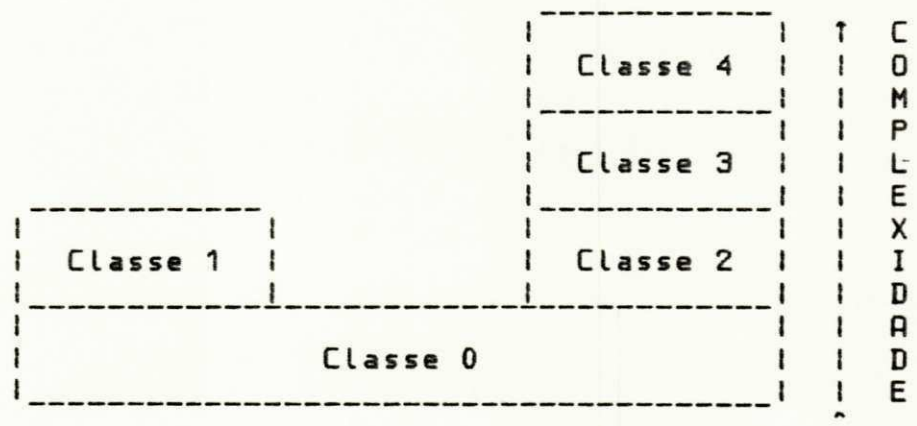

Figura 1.3: Relação entre as classes do protocolo de transporte.

Para as camadas de sessão e apresentação já existem as propostas de padronização [ISO 83b, ISO 83c] e [ISO 84c, ISO 84d] respectivamente. No caso da camada de aplicação, algumas propostas já foram efetuadas. Por exemplo, o protocolo para a transferência de arquivos FTAM [ISO 86a].

Uma vez que as redes de computadores atendem a sistemas computacionais heterogêneos e uma vez que as implementações dos

protocolos de comunicação são normalmente realizadas em equipe, é importante uma descrição clara e concisa de tais protocolos.

Na descrição de uma rede de computadores, dois conceitos são fundamentais: **serviço** e **protocolo** [Lope 87a]. Utilizando os serviços oferecidos pela camada(N-1), as entidades da camada(N) cooperam entre si, de acordo com o protocolo(N), para fornecer um serviço(N) com mais recursos à camada(N+1) (Fig. 1.4).

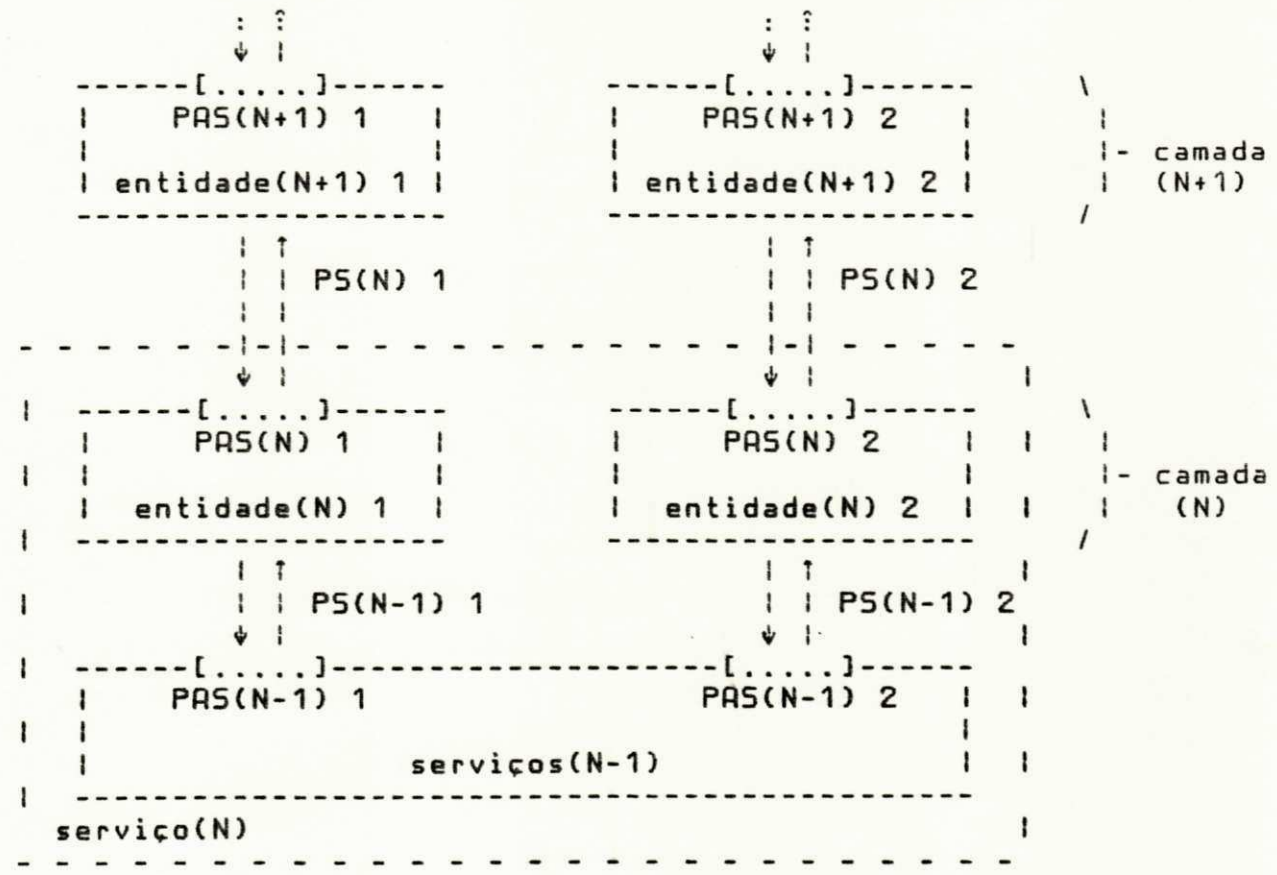

Figura 1.4: Especificação dos protocolos de uma camada de um sistema descrito hierarquicamente.

A noção de serviço(N) é abstrata, pois condensa todas as n camadas, omitindo o fluxo de dados entre elas. A especificação desse serviço é a descrição, por um observador externo, do comportamento de uma "caixa preta", sujeita às trocas de

primitivas de serviço(N) (PS(N)) com a camada superior, sendo que tais trocas são realizadas através dos **pontos de acesso ao serviço(N) (PAS(N))**.

Na especificação do protocolo(N) é descrito o comportamento das entidades que se comunicam, se sincronizam e operam de forma concorrente via os PAS(N-1) [Boch 80].

O projeto e implementação de sistemas de comunicação, além dos problemas inerentes a todo projeto de desenvolvimento de software, apresenta algumas particularidades [Boch 83]:

(a) deve-se garantir a compatibilidade entre seus diversos componentes;

(b) quase sempre tais componentes são implantados por grupos distintos de técnicos trabalhando em organizações distintas;

(c) o funcionamento do sistema como um todo é de difícil compreensão, devido ao paralelismo das operações que são executadas nos seus diferentes componentes.

A experiência tem mostrado que especificações informais de sistemas de comunicação causam ambiguidades, dificultando o seu desenvolvimento [IsRo 85]. O emprego de técnicas formais de descrição para a especificação de serviços e de protocolos de comunicação, além de gerar especificações não ambíguas, que servem como referências confiáveis para o trabalho em equipe, facilita as tarefas subsequentes de verificação, implementação e teste desses protocolos.

A ISO, interessada também na fixação de padrões para a especificação formal dos protocolos de comunicação, criou o

grupo de trabalho ISO TC97/SC16/WG1 para o desenvolvimento de técnicas de descrição formal (TDF). Esse grupo foi dividido em três subgrupos, sendo que o primeiro (subgrupo A) ficou responsável pela definição dos conceitos arquiteturais, que dariam suporte ao desenvolvimento de TDFs pelos outros subgrupos. O subgrupo B ficou responsável pelo desenvolvimento de uma TDF baseada no conceito de **máquina de estados finita estendida** (MEFE), que deu origem à técnica **"Extended state transition language (Estelle)"**. O subgrupo C, por sua vez, criou a técnica **"Language of temporal ordering specification (Lotos)"**, que é baseada em ordem temporal de primitivas de interação. Vale ressaltar que essas técnicas estão em evolução, não tendo alcançado ainda seu estágio final.

Um compilador, para uma técnica de descrição formal, permite a obtenção semi-automática de protótipos de implementações, cujas especificações são realizadas nessa técnica. Isso facilita a legibilidade, a transportabilidade e o desenvolvimento de implementações definitivas, já que elimina boa parte da codificação personalizada [LoFe 87a].

Uma implementação semi-automática pode ser também uma referência confiável para a padronização de implementações, pertencentes a ambientes computacionais homogêneos, e para a comparação com o comportamento de uma implementação desenvolvida manualmente (desde que ambas sejam derivadas da mesma especificação).

Por fim, esse compilador pode ser integrado a sistemas que visam:

(a) um certo grau de verificação (ou validação do design) para as diversas especificações (referentes aos diversos niveis de abstração) de um protocolo ou

(b) estudos de simulação para a análise de desempenho desse protocolo.

O objetivo desse trabalho é apresentar os resultados do desenvolvimento de um compilador para a TDF Estelle e sua utilidade como parte de um ambiente de simulação.

No capitulo 2 são apresentadas algumas técnicas de descrição formal para protocolos, com seus métodos de validação pertinentes e no capitulo 3 é introduzida a TDF Estelle. No capitulo 4 é descrito o compilador Estelle/83. Os resultados obtidos, com os testes desse compilador, são apresentados no capitulo 5. No capitulo 6 são sugeridas futuras direções de trabalho, que podem fazer uso desse compilador. Finalmente as conclusões são apresentadas no capitulo 7.

2. ESPECIFICAÇÃO E VALIDAÇÃO DE SERVIÇOS E PROTOCOLOS

O desenvolvimento de um protocolo de comunicação pode ser realizado através das seguintes etapas (**ciclo de desenvolvimento**) (Fig. 2.1) [LoSt 88]:

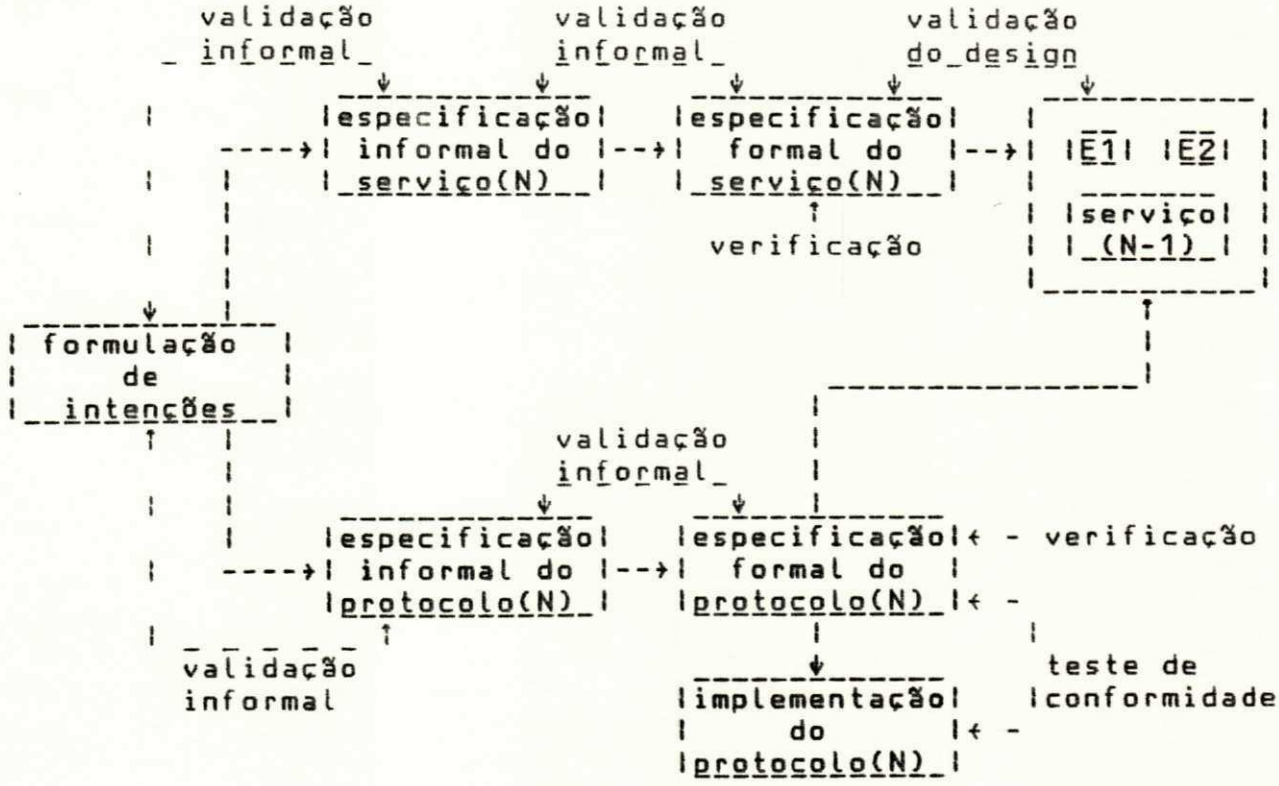

Figura 2.1: Ciclo de desenvolvimento de um protocolo.

(a) **formulação de intenções**, onde são definidas, de maneira geral, as principais caracteristicas do protocolo;

(b) **especificação informal**, escrita em linguagem natural e contendo uma descrição detalhada do software a ser produzido;

(c) **especificação formal**, realizada através de uma TDF, o que a torna uma referência confiável para o trabalho em equipe, já que elimina as ambiguidades introduzidas pela especificação

informal. Ela é útil também para o desenvolvimento da fase posterior e para as atividades de validação e

(d) **implementação**, que tem por objetivo a geração de um código executável a partir da especificação formal.

Associadas a essas etapas, pode-se também identificar algumas **atividades de validação**:

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

(b) **verificação** da ocorrência de propriedades gerais (ausência de impasses, ausência de ciclos improdutivos, vivacidade, termino apropriado, etc...) [Boch 78] e propriedades especificas que garantam a correção do protocolo;

(c) **validação do design** do protocolo, isto é, verificar 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** da implementação em relação à especificação.

Para que as especificações formais possam fornecer descrições claras e concisas, do sistema que está sendo concebido, e possam permitir uma análise rigorosa das diferentes fases do ciclo de desenvolvimento de um protocolo, é necessário que as TDFs utilizadas, para a obtenção dessas especificações formais, possuam:

(a) um alto grau de **abstração**, no sentido de independência em relação aos métodos de implementação e no sentido de omissão, em qualquer etapa da especificação, dos detalhes irrelevantes;

(b) poder de **expressão**, isto é, suas construções além de fornecer meios para exprimir comunicação, sincronização e concorrência, devem permitir a descrição dos serviços e protocolos das sete camadas OSI de forma hierárquica, modular e intuitiva e

(c) poder de **análise**, ou seja, possuam um modelo matemático, que permita a verificação formal de propriedades desejadas para os objetos que estão sendo especificados.

2.1 TDFs e métodos de verificação associados

Várias técnicas foram propostas para a especificação de serviços e protocolos de comunicação. Essas técnicas podem ser agrupadas em três grandes categorias:

(a) técnicas baseadas em modelos de transição;

(b) técnicas baseadas em linguagens de programação e

(c) técnicas hibridas.

As TDFs baseadas em modelos de transição são eficazes para a descrição dos aspectos de controle dos protocolos (fases de inicialização, de estabelecimento e de encerramento de conexão). Dentro dessa categoria, algumas das técnicas mais conhecidas são: as baseadas em **Máquina de Estados Finita** (MEF) [Boch 78], **Redes**

de Petri [Dant 80] e Linguagens Formais [TeLi 78].

Uma MEF é uma quintupla ‹X,E,S,T,A›, onde X é um conjunto finito de estados, E um conjunto finito de entradas, S um conjunto finito de saidas, T a função de transição de estado (T : ExX -› X) e A a função de saida (A : ExX -› S).

Um protocolo pode ser descrito através de uma MEF, já que a uma entidade pode ser atribuido um número finito de estados. A partir de um número finito de eventos (entradas), transições de estado, associadas (ou não) a um número finito de ações (saidas), podem ocorrer. A Figura 2.2 ilustra um sistema simplificado de transmissão de mensagens

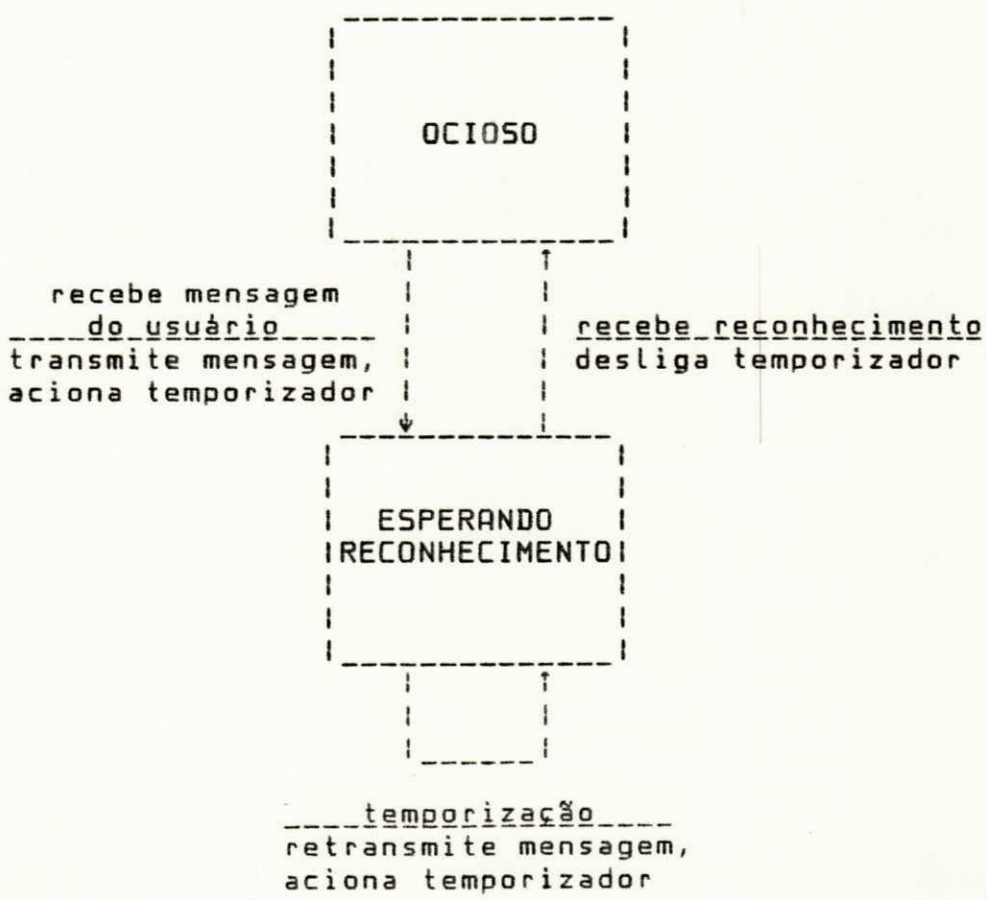

Figura 2.2: Exemplo do emprego de MEF como TDF.

onde os estados estão representados por quadrados, as transições por arcos direcionados e uma linha separa os eventos (parte superior) das ações (parte inferior). Inicialmente o estado do sistema é OCIOSO. Ao receber uma mensagem do usuário, o sistema "transitará" para o estado ESPERANDO_RECONHECIMENTO, havendo a ocorrência de duas ações: transmissão da mensagem e acionamento do temporizador. Ao receber um reconhecimento em tempo hábil, o sistema retornará ao estado OCIOSO e o temporizador será desligado. Caso ocorra o fim da temporização, a mensagem será retransmitida e o temporizador será acionado outra vez, permanecendo inalterado o estado do sistema.

Uma **Rede de Petri** é uma quádrupla ‹L,T,E,S›, onde **L** é um conjunto finito de nós (ou lugares), **T** um conjunto de transições (ou eventos), **E** a função de entrada e **S** a função de saída. Graficamente, as Redes de Petri são constituídas de nós (representando condições), barras (representando transições) e arcos direcionados (ligando nós a barras e barras a nós). Uma ficha é atribuída a um nó, desde que a condição, a ele associada, seja satisfeita. Para que uma transição possa ocorrer, é necessário que todas as condições de entrada estejam satisfeitas (pelo menos uma ficha em cada um dos nós de entrada). A ocorrência de uma transição implica na remoção de uma ficha de cada um dos nós de entrada e a alocação de uma ficha em cada um dos nós de saída. O estado da rede, num determinado instante, é dado pela distribuição de fichas em seus nós (marcação). A figura 2.3 descreve o mesmo sistema, apresentado anteriormente, através de uma Rede de Petri. Essa técnica gráfica é bastante utilizada como TDF, devido à sua simplicidade conceitual e devido à sua capacidade para a análise do fluxo de dados que ocorrem num

sistema de comunicação [CaCu 87].

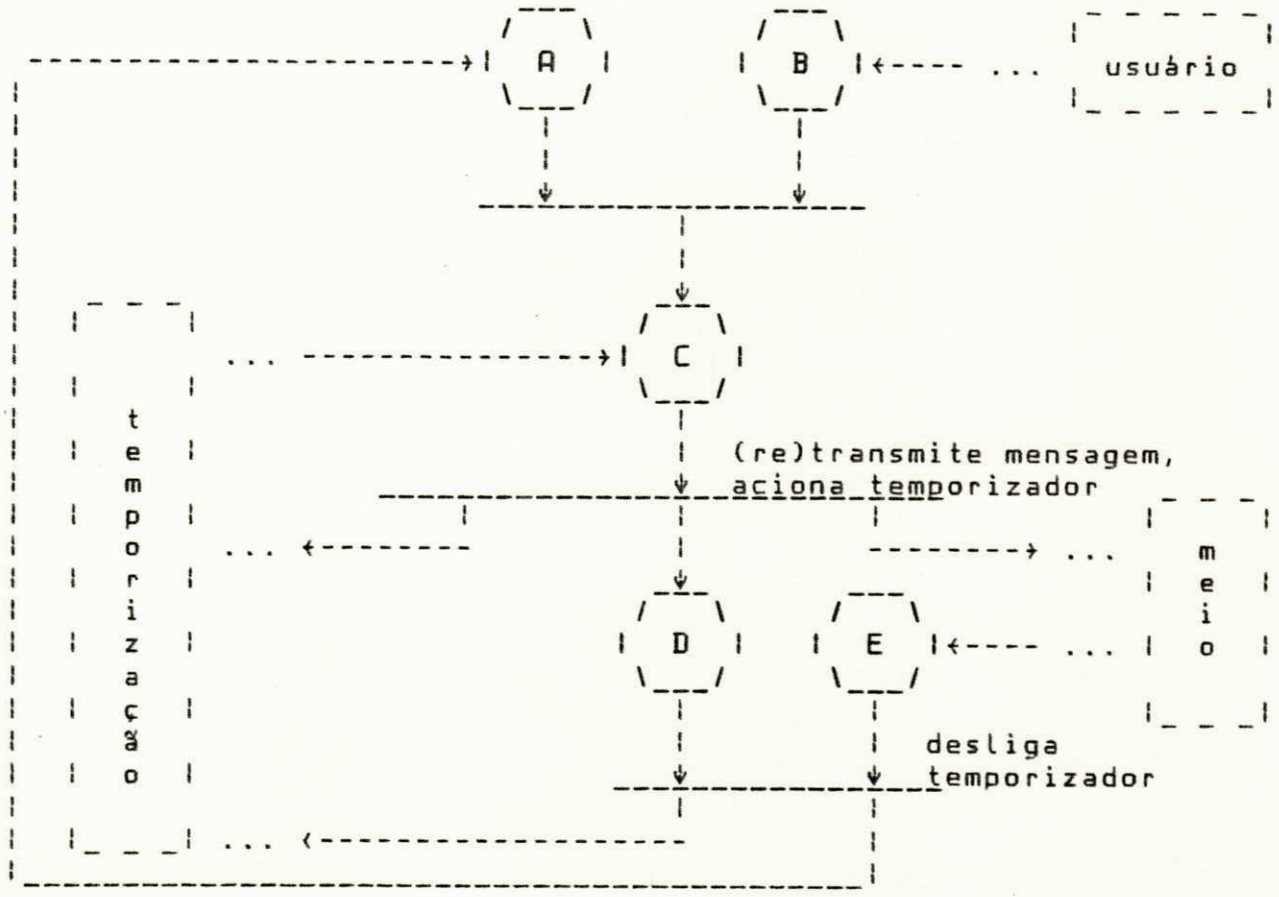

Figura 2.3: Exemplo do emprego de Rede de Petri como TDF.

A inclusão de linguagens formais, na categoria das TDFs baseadas em modelo de transição, deve-se à correspondência existente entre MEFs e gramáticas formais [Mont 82]. A representação de um modelo de transição em BNF ("Backus Naur Form") possibilita a descrição de um sistema em diversos graus de abstração, através do uso de simbolos não-terminais. As ações do sistema são definidas pela cadeia de simbolos terminais, a qual é gerada através da aplicação das regras de produção (Fig. 2.4).

```
‹OCIOSO›   ::= "recebe mensagem"
               "transminte mensagem"
               "liga temporizador"
               ‹ESPERANDO RECONHECIMENTO›.

‹ESPERANDO RECONHECIMENTO› ::= "recebe reconhecimento"
                               "desliga temporizador"
                               ‹OCIOSO›
                             | "temporização"
                               "retransmite mensagem"
                               "aciona temporizador"
                               ‹ESPERANDO RECONHECIMENTO›.
```

Figura 2.4: Exemplo do emprego de Linguagem Formal como TDF.

A técnica de verificação (TVF), a ser empregada para a validação de um sistema de comunicação, depende da TDF utilizada para a sua especificação.

Para as TDFs baseadas em modelos de transição, as TVFs mais empregadas são baseadas em métodos de exploração de estados. O grande inconveniente desse tipo de método está na possibilidade de um crescimento excessivo do número de estados (explosão de estados), inviabilizando qualquer tipo de análise [BoSu 80].

Linguagens de Programação podem também ser utilizadas para a especificação de serviços e protocolos [Boch 75]. Isto porque protocolos podem ser representados através de processos concorrentes, os quais podem ser descritos utilizando-se linguagens de programação de alto nivel (Fig. 2.5). Esse tipo de TDF é eficaz para a descrição dos aspectos relativos à transferência e estrutura de dados dos protocolos. Tais métodos permitem também a obtenção de especificações próximas às suas implementações. Por outro lado, o alto grau de detalhamento presente nas especificações restringe as alternativas de implementação.

```
const PARA_SEMPRE = false;
type MENSAGEM = ...;

procedure RECEBE_MENSAGEM(var M : MENSAGEM);
begin
   { espera o recebimento de mensagem do usuário }
end;

function TEMPORIZACAO : boolean;
begin
   { retorna TRUE se o tempo de reconhecimento expirou-se }
end;

procedure LIGA_TEMPORIZACAO;
begin
   { inicia a temporização da mensagem enviada }
end;

procedure DESLIGA_TEMPORIZACAO;
begin
   { desliga a temporização da mensagem enviada }
end;

procedure TRANSMITE(M : MENSAGEM);
begin
   { envia mensagem }
end;

procedure RETRANSMITE;
begin
   { retransmite a mensagem e reinicializa a temporização }
end;

function RECEBE_RECONHECIMENTO : boolean;
begin
   { retorna TRUE se um reconhecimento foi recebido }
end;

procedure PROCESSO;
var M : MENSAGEM;
begin
   RECEBE_MENSAGEM(M);
   repeat
      TRANSMITE(M);
      LIGA_TEMPORIZACAO;
      while not (TEMPORIZACAO or RECEBE_RECONHECIMENTO) do;
      if TEMPORIZACAO then
         RETRANSMITE
      else
         begin
            DESLIGA_TEMPORIZACAO;
            RECEBE_MENSAGEM(M)
         end;
   until PARA_SEMPRE
end;
```

Figura 2.5: Exemplo do emprego de linguagem de programação como TDF.

Ainda dentro dessa categoria, técnicas que empregam notações mais abstratas têm sido utilizadas para a especificação de protocolos. Entre elas, destacam-se: **lógica temporal** [Lamp 80] e **tipos abstratos de dados** [SuTh 81].

Para as TDFs baseadas em linguagem de programação, as TVFs mais empregadas são baseadas em **provas de programa**. Em tais métodos podem ser distinguidas duas fases: numa primeira etapa são formuladas asserções, relativas a determinadas propriedades requeridas aos protocolos, e numa segunda etapa são verificadas essas asserções. O grande inconveniente desse tipo de TVF é que ela exige uma certa dose de criatividade para a formulação e prova das asserções.

As técnicas híbridas combinam as vantagens das duas categorias anteriores. Por exemplo, o controle das transições pode ser especificado empregando-se um modelo de transição, ao qual é adicionado variáveis de contexto e procedimentos descritos numa linguagem de programação de alto nível [ScRo 80] (Fig. 2.6).

Para esse tipo de TDF híbrida, as TVFs associadas geralmente utilizam exploração de estados no modelo de transição implícito e provas de programa nas variáveis de contexto e procedimentos [BoGe 77].

```
type
   MENSAGEM : ...;

var
   M : MENSAGEM;

interface
   [from U : MENS(M)];
   [from I : RECONHECIMENTO];
   [to   I : MENS(M)];
   [from S : TEMPORIZACAO];
   [to   S : LIGA_TEMPORIZACAO];
   [to   S : DESLIGA_TEMPORIZACAO];

procedure RETRANSMISSAO;
begin
   { retransmite mensagem e reinicializa a temporização }
end;

‹ESPERANDO_RECONHECIMENTO› ‹-- ‹OCIOSO›
   [from U : MENS(M)]
   begin
      [to I : MENS(M)];
      [to S : TEMPORIZACAO]
   end;

‹ESPERANDO_RECONHECIMENTO› ‹-- ‹ESPERANDO_RECONHECIMENTO›
   [from S : TEMPORIZACAO]
   begin
      RETRANSMISSAO
   end;

‹OCIOSO› ‹-- ‹ESPERANDO_RECONHECIMENTO›
   [from I : RECONHECIMENTO]
   begin
      [to S : DESLIGA_TEMPORIZACAO]
   end;
```

Figura 2.6: Exemplo do emprego de técnica hibrida como TDF.

As TDFs apresentadas permitem, teoricamente, fornecer resultados definitivos a respeito dos protocolos analisados. Entretanto, as restrições que são impostas, para a aplicação dessas TVFs em protocolos reais, podem levar a conclusões limitadas.

A **simulação** é um método de verificação aplicável a várias categorias de TDFs. Embora não possa, teoricamente, fornecer

resultados definitivos, ela pode outorgar um certo grau de confiabilidade aos protocolos reais. Maiores detalhes sobre o uso da simulação, na atividade de validação de especificações de serviços e protocolos, será mostrado no capitulo 6.

Uma outra atividade de validação importante é o **teste de conformidade**. Ele é feito a partir do estudo do comportamento de uma implementação, frente a uma gama de situações possiveis, definidas à priori. Esses **cenários** devem conter um conjunto de estimulos externos, aos quais a implementação será submetida, e um conjunto de resultados esperados. Os estudos nessa área têm se concentrado na definição de arquiteturas de teste e na obtenção de cenários de teste [Sari 87].

2.2 TDFs em vias de padronização

A preocupação de órgãos internacionais de padronização, tais como o "National Bureau of Standards (NBS)", o CCITT e a ISO, com os problemas causados pelas especificações informais de serviços e protocolos de comunicação, levou-os à busca de TDFs padrões.

Em [BlTe 81] é apresentada a proposta da NBS para uma TDF baseada em autômato finito acrescido de variáveis, onde as transições são vinculadas a segmentos de programa. O exemplo mostrado na Figura 2.6 foi definido segundo essa técnica.

A técnica "Specification and Description Language (SDL)" foi desenvolvida, pelo CCITT, para descrever sistemas de chaveamento [CCIT 83a, CCIT 83b, CCIT 83c, CCIT 83d, CCIT 83e]. Ela é baseada em um modelo MEFE e era voltada, originalmente, para a representação gráfica (SDL/GR - "Graphical Representation"). Nos

últimos anos, porém, foram incluidos recursos para a definição de estruturas de dados e tipos abstratos de dados (SDL/PR - "Programming Representation").

Lotos (definida pela ISO) é uma técnica de especificação algébrica [ISO 86b]. E uma combinação de "Calculus of Communicating Systems (CCS)" [Miln 80], usado para definir o comportamento dinâmico de processos, com ACT ONE [EhMa 85], que é uma linguagem para a especificação de tipos abstratos de dados. Em Lotos, um sistema é descrito através de um processo ou um conjunto de processos. Cada processo é representado por uma caixa preta [Lope 87b].

Estelle (também desenvolvida pela ISO) [ISO 85] é uma TDF híbrida baseada em MEFE, que tem como suporte a linguagem de programação Pascal [ISO 83d]. A TDF Estelle, por ser o objeto deste trabalho, será vista detalhadamente no próximo capitulo.

Uma comparação entre as TDFs SDL, Lotos e Estelle pode ser encontrada em [Boch 87a].

3. Estelle/83: UMA TDF BASEADA EM MEFE

Estelle foi fruto, principalmente, da cooperação entre a ISO e o CCITT. Ela surgiu do aprimoramento dos trabalhos, relativos à utilização de MEFEs como TDFs, desenvolvidos por vários pesquisadores (entre eles, Ansart [AnRa 82], Bochmann [BoGe 77, Boch 78] e Tenney [BlTe 81]).

Estelle utiliza os conceitos arquiteturais definidos pelo ISO TC97/SC16/WG1 subgroup A: **módulo**, **refinamentos** de um módulo em submódulos, **canal**, **porta**, **interação** e **conexão** de módulos [ISO 82].

Neste capítulo serão introduzidos os conceitos acima mencionados, conjuntamente com suas representações em Estelle/83 [ISO 83e] (uma das primeiras versões dessa TDF). A BNF de Estelle/83 é apresentada no Anexo A.

3.1 Arquitetura de um sistema

A arquitetura de um sistema é definida por um conjunto de **módulos**, interconectados através de **canais**. Um módulo é visto pelo seu ambiente (demais módulos que compõem o sistema) como um conjunto de **portas**. Esse tipo de estrutura permite especificar módulos e conexões separadamente (Fig. 3.1) [Lope 85].

Um canal define um conjunto de **mensagens** (que podem ser invocadas através desse tipo de canal), os valores possíveis dos parâmetros associados a essas mensagens e os papéis que os módulos, conectados às extremidades desse canal, deverão desempenhar (e.g. "usuário" e "provedor").

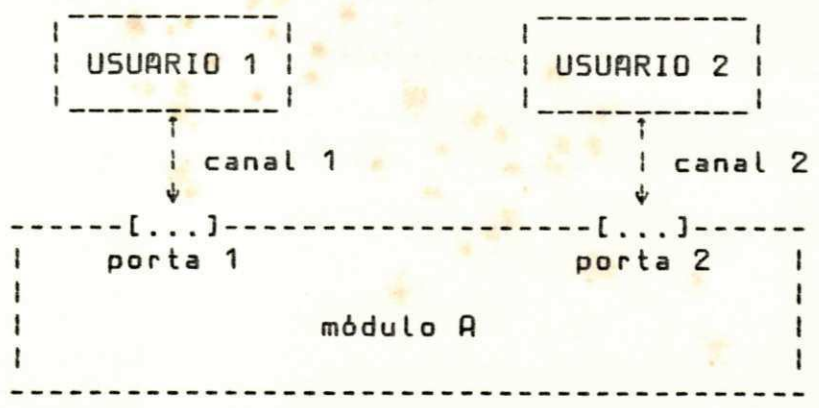

Figura 3.1: Especificação de um serviço de comunicação.

Um módulo é especificado, no nivel mais alto de abstração, descrevendo-se suas portas. Uma porta pode ser **elementar** ou **estruturada** ("array" de portas). Cada porta é caracterizada pelo tipo de canal ao qual ela está associada.

A especificação completa de um módulo pode ser realizada através do seu refinamento em submódulos (Fig. 3.2) ou através da definição do seu comportamento, em termos de um processo.

No caso da linguagem Estelle/83, a descrição do comportamento de cada módulo é baseado numa MEFE. As transições de uma MEFE ocorrem em função das interações produzidas pelo ambiente e recebidas pelo módulo (**entradas**), ou ocorrem em função dos eventos internos (**transições espontâneas**). Ao executar uma transição, o módulo pode gerar interações (**saidas**). As interações de um módulo com o seu ambiente são consideradas atômicas, ou seja, uma única entrada de um único módulo é tratada por vez. Como a partir de uma interação de entrada é possivel a existência de várias transições executáveis e como a partir de um determinado estado é possivel a existência de transições espontâneas, a especificação de um módulo é **não deterministica**.

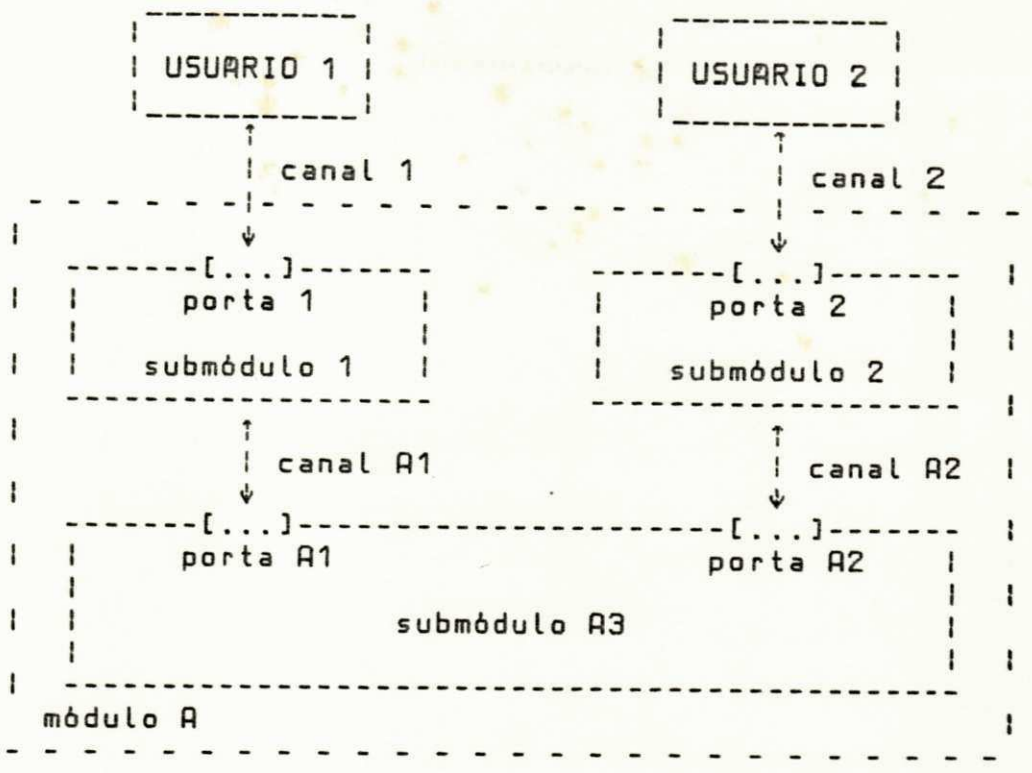

Figura 3.2: Refinamento de um módulo.

A descrição de um módulo contém uma variável, que define o **estado principal** do módulo, e um conjunto de **variáveis de estado adicionais**. Simultaneamente com os parâmetros de interação de entrada, elas determinam a transição a executar no módulo. A cada transição está associada uma **condição**, que deve ser satisfeita para a execução da transição, e uma **ação**, que atualiza os valores das variáveis e eventualmente gera interações de saída.

A disciplina de comunicação entre os módulos pode ser feita através de:

(a) **rendez-vouz**. Nesse caso a comunicação só ocorrerá quando um dos módulos estiver apto a oferecer uma interação e o outro módulo estiver apto a aceitar essa mesma interação ou

(b) filas. Nesse caso o módulo emissor coloca a interação na fila de espera do módulo receptor, liberando-se para outras atividades.

3.2 Especificação de um protocolo em Estelle/83

Uma vez definida a arquitetura de um sistema a ser especificado, a descrição dos componentes desse sistema, em Estelle/83, assemelha-se a um programa Pascal. Inicialmente são declarados os elementos globais: constantes e tipos. Simbolos do tipo "..." ou palavras do tipo "primitive" indicam que a escolha é deixada a cargo do implementador. Em seguida são definidos os canais, conjuntamente com as primitivas a serem iniciadas pelo usuário e pelo provedor de serviços. Finalmente os módulos são declarados.

Após o cabeçalho de um módulo, que é a declaração de seu nome e de suas portas (associadas ou não a filas - "queued"), são definidos os seus elementos locais: constantes, tipos, rótulos, variáveis (entre as quais uma representando o estado principal), funções e procedimentos. O módulo é então inicializado ("initialize") e as transições ("trans") são descritas.

Uma transição é declarada através das seguintes cláusulas: "from" (seguida do estado principal vigente), "to" (seguida do novo estado principal), "when" (seguida da interação de entrada) e "provided" (seguida das condições habilitadoras da transição). A cláusula "with" também pode ser empregada e sua semântica é idêntica à utilizada em Pascal. A fim de condensar a descrição das transições, a ordem dessas cláusulas é irrelevante e pode

haver uma combinação das mesmas. As transições espontâneas podem conter a cláusula "any", que é executada quando as condições a ela associadas são satisfeitas. Para a execução das transições, uma "priority" pode ser estabelecida (quanto maior é o valor do número inteiro, menor é a prioridade). Após a declaração das cláusulas de uma transição, são atualizadas as variáveis de estado adicionais, podendo ocorrer ou não uma interação de saída (instrução "out"). A estrutura de uma transição, com suas possíveis cláusulas, é apresentada na Figura 3.3.

```
trans
  priority ‹expressão›
  from ‹estado principal vigente›
  to ‹novo estado principal›
  when ‹interação de entrada›
  provided ‹condições habilitadoras›
  with ‹variável› do
    begin . . .
      {atualização das variáveis de estado adicionais }
      . . .
      out ‹interação de saída›
      . . .
    end;
```

Figura 3.3: Estrutura de uma transição.

Obs.: em Estelle/83, pode ser definido um conjunto de estados e esse conjunto ser referenciado após a cláusula **from**.

Para o exemplo usado no capítulo 2, pode-se considerar a arquitetura apresentada na Figura 3.4. Nessa arquitetura, três módulos são definidos: EMISSOR, RECEPTOR e MEIO_DE_TRANSMISSAO. O módulo EMISSOR fornece mensagens de dado para o módulo MEIO_DE_TRANSMISSAO, que se encarrega de encaminhar essas mensagens para o módulo RECEPTOR. Esse último, por sua vez, responde enviando mensagens de reconhecimento ao módulo EMISSOR,

via o MEIO_DE_TRANSMISSAO. É suposto que o MEIO_DE_TRANSMISSAO pode perder mensagens.

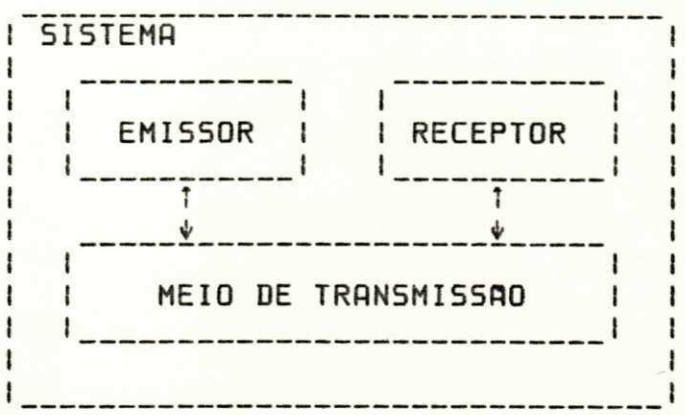

Figura 3.4: Exemplo de arquitetura de um sistema de transmissão de mensagens.

Para a descrição das interações que ocorrem entre esses módulos podem ser especificados, em Estelle/83, os seguintes canais: EMISSOR_MEIO e RECEPTOR_MEIO (Fig. 3.5).

```
type MSG_TYPE = ...;

channel EMISSOR_MEIO(USUARIO,PROVEDOR);
  by USUARIO : MENSAGEM(M : MSG_TYPE);
  by PROVEDOR : RECONHECIMENTO;
end EMISSOR_MEIO;

channel RECEPTOR_MEIO(USUARIO,PROVEDOR);
  by USUARIO : RECONHECIMENTO;
  by PROVEDOR : MENSAGEM(M : MSG_TYPE);
end RECEPTOR_MEIO;
```

Figura 3.5: Exemplo de especificação de canais.

A especificação abstrata dos módulos EMISSOR, RECEPTOR e MEIO_DE_TRANSMISSAO é apresentada na figura 3.6.

```
module EMISSOR;
  PORTA_E : EMISSOR_MEIO(USUARIO);
end EMISSOR;

module RECEPTOR;
  PORTA_R : RECEPTOR_MEIO_R(USUARIO);
end RECEPTOR;

module MEIO_DE_TRANSMISSAO;
  PORTA_E : EMISSOR_MEIO(PROVEDOR);
  PORTA_R : RECEPTOR_MEIO(PROVEDOR);
end MEIO_DE_TRANSMISSAO;
```

Figura 3.6: Exemplo de especificação abstrata de módulos.

Esses módulos, por sua vez, podem ser refinados até alcançar um nivel de detalhamento conveniente. Por exemplo, o módulo EMISSOR pode ser subdividido em três submódulos: USUARIO, PROTOCOLO e TEMPORIZADOR (Fig. 3.7). A especificação desse refinamento é apresentada na figura 3.8.

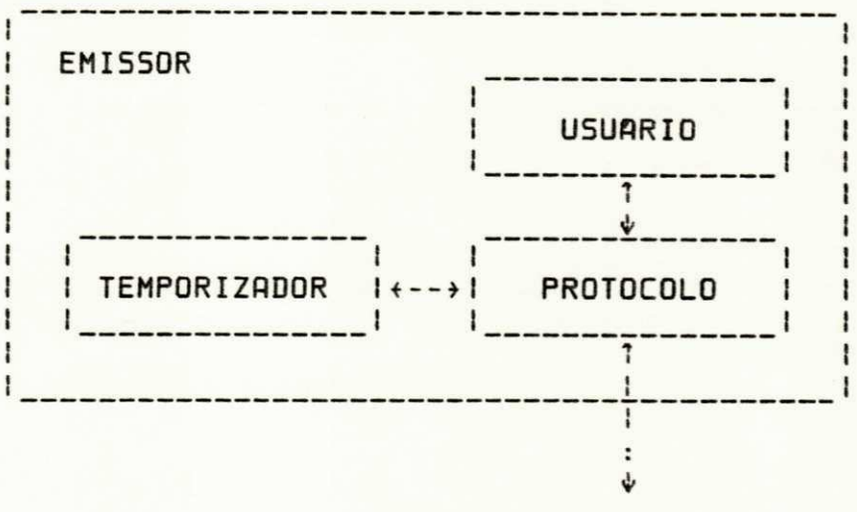

Figura 3.7: Exemplo de refinamento de um módulo em submódulos.

Uma vez especificados os objetos que compõem o sistema, deve-se especificar as ocorrências dos módulos

```
‹nome da ocorrência› : ‹nome do módulo›
                       with ‹corpo do módulo›;
```

onde ‹corpo do módulo› é o nome do processo ou do refinamento que especifica o módulo referenciado.

```
refinement REF_EMISSOR for EMISSOR;
  . . .
  channel PROTOCOLO_USUARIO(AMBOS);
    by AMBOS : MENSAGEM(M : MSG_TYPE);
  end PROTOCOLO_USUARIO;

  channel PROTOCOLO_TEMPORIZADOR(USUARIO,PROVEDOR);
    by USUARIO : LIGA_TEMPORIZADOR;
                 DESLIGA_TEMPORIZADOR;
    by PROVEDOR : TEMPORIZACAO;
  end PROTOCOLO_TEMPORIZADOR;
  . . .
  module USUARIO;
    PORTA_U : PROTOCOLO_USUARIO(AMBOS);
  end USUARIO;

  process PROC_USUARIO for USUARIO;
    . . .
  end PROC_USUARIO;

  module TEMPORIZADOR;
    PORT_T : PROTOCOLO_TEMPORIZADOR(PROVEDOR);
  end TEMPORIZADOR;

  process PROC_TEMPORIZADOR for TEMPORIZADOR;
    . . .
  end PROC_TEMPORIZADOR;

  module PROTOCOLO;
    PORTA_U : PROTOCOLO_USUARIO(AMBOS);
    PORTA_M : EMISSOR_MEIO(USUARIO);
    PORTA_T : PROTOCOLO_TEMPORIZADOR(USUARIO);
  end PROTOCOLO;

  process PROC_PROTOCOLO for PROTOCOLO;
    . . .
  end PROC_PROTOCOLO;

  T : TEMPORIZADOR with PROC_TEMPORIZADOR;
  U : USUARIO with PROC_USUARIO;
  P : PROTOCOLO with PROC_PROTOCOLO;

  connect T.PORTA_T to P.PORTA_T;
          U.PORTA_U to P.PORTA_U;

  replace PORTA_E by P.PORTA_M;

end REF_EMISSOR;
```

Figura 3.8: Exemplo de especificação de um refinamento.

Em seguida, as ocorrências dos módulos devem ser conectadas, através de suas portas (declaração "**connect**"). As portas dos módulos refinados (módulos pai), que correspondem às portas das ocorrências dos submódulos gerados (módulos filho), devem ser vinculadas (declaração "**replace**").

```
connect
  ‹nome da ocorrência›.‹nome da porta›
  to ‹nome da ocorrência›.‹nome da porta›;

replace
  ‹nome da porta› by ‹nome da ocorrência›.‹nome da porta›;
```

Os comportamentos dos submódulos USUARIO, TEMPORIZADOR e PROTOCOLO são descritos pelos seus respectivos processos. Por exemplo, o processo que descreve o comportamento do submódulo PROTOCOLO é apresentado na figura 3.9.

```
process PROC_PROTOCOLO for PROTOCOLO;

  queued PORTA_U,PORTA_M;

  var state : (OCIOSO,ESPERANDO_RECONHECIMENTO);

  initialize begin state:=OCIOSO end;

  trans when PORTA_U.MENSAGEM
        from OCIOSO to ESPERANDO_RECONHECIMENTO
          begin
            out PORTA_M.MENSAGEM(M);
            out PORTA_T.LIGA_TEMPORIZADOR
          end;
        when PORTA_M.RECONHECIMENTO
        from ESPERANDO_RECONHECIMENTO to OCIOSO
          begin
            out PORTA_T.DESLIGA_TEMPORIZADOR
          end;
        when PORTA_T.TEMPORIZADOR
          begin
            out PORTA_M.MENSAGEM(M);
            out PORTA_T.LIGA_TEMPORIZADOR
          end;

  end PROC_PROTOCOLO;
```

Figura 3.9 : Exemplo de especificação do comportamento do módulo PROTOCOLO.

Os Anexos C, F e H apresentam exemplos de especificações completas, em Estelle/83, para protocolos do tipo bit-alternante. Detalhes sobre essas especificações serão vistas no capitulo 5.

4. O COMPILADOR Estelle/83

O objetivo maior de um compilador, para uma TDF, é possibilitar implementações semi-automáticas de sistemas especificados formalmente nessa TDF.

Pode-se definir três etapas para a construção de uma implementação completa, a partir de sua especificação em Estelle/83 [BoGe 84]:

(a) compilação da especificação formal;

(b) desenvolvimento de um núcleo de exploração e

(c) codificação das rotinas que implementam as primitivas de baixo nivel, responsáveis pela utilização dos recursos oferecidos pelo ambiente computacional, no qual o protocolo será inserido.

As implementações geradas, a partir de uma especificação em Estelle/83, são constituidas de estruturas de dados e de procedimentos que deveriam, em principio, operar simultaneamente. Na maioria das vezes, porém, esse paralelismo não é real mas simulado. Deve haver, portanto, uma parte da implementação que seja responsável pela execução pseudo-paralela dos processos: o núcleo. Esse núcleo deve [FeLo 86]:

(a) definir uma politica de escalonamento para a recepção/emissão de mensagens;

(b) identificar e acionar os processos envolvidos no intercâmbio de mensagens e

(c) gerir as transições espontâneas (internas).

Uma vez que a implementação das primitivas de baixo nivel depende quase que exclusivamente do ambiente onde elas irão atuar, elas devem ser codificadas, a parte, numa linguagem apropriada.

Além disso, rotinas de suporte, que auxiliam a manipulação das estruturas de dados geradas pelo compilador Estelle/83, devem ser consideradas.

A Figura 4.1 ilustra essas etapas e os procedimentos necessários para alcançar um código executável.

4.1 Arquitetura do compilador

A especificação completa, em Estelle/83, de um sistema, é geralmente composta de vários módulos "fontes", que interagem através de mensagens. Portanto, o compilador deve permitir o tratamento desses módulos separadamente.

O compilador Estelle/83 utiliza a sintaxe apresentada em [ISO 83e] e gera, como um passo intermediário, um programa numa linguagem de alto nivel. Esse programa é constituido de um conjunto de estruturas de dados e de um conjunto de procedimentos, que refletem os conceitos próprios da TDF Estelle/83. Uma vez que essa TDF tem como suporte a linguagem de programação Pascal, torna-se natural o uso de Pascal (ou outra linguagem correlata) como linguagem de implementação.

Figura 4.1: Etapas e procedimentos necessários para se alcançar um código executável.

O compilador Estelle/83 é, portanto, constituido de um **pré-processador** e de um compilador Pascal. A função desse pré-processador é traduzir as construções de nivel mais alto da TDF Estelle/83, produzindo um programa totalmente em Pascal (Fig. 4.2).

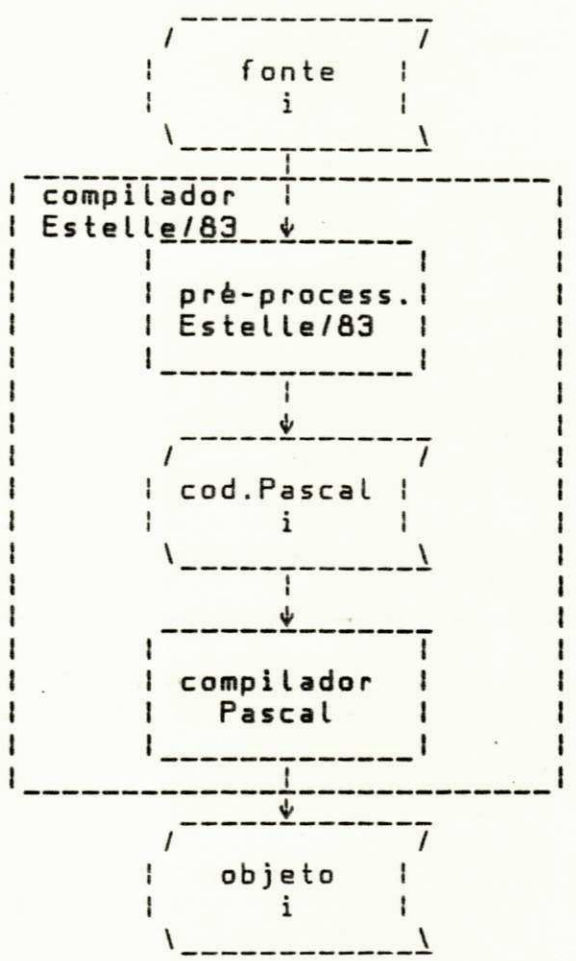

Figura 4.2: Estrutura do compilador Estelle/83.

O pré-processador Estelle/83 é responsável pela análise de um código fonte Estelle/83 e pela geração do correspondente código de implementação Pascal.

Na atividade de análise, podem ser distinguidas quatro funções principais (Fig. 4.3) [Grie 71]:

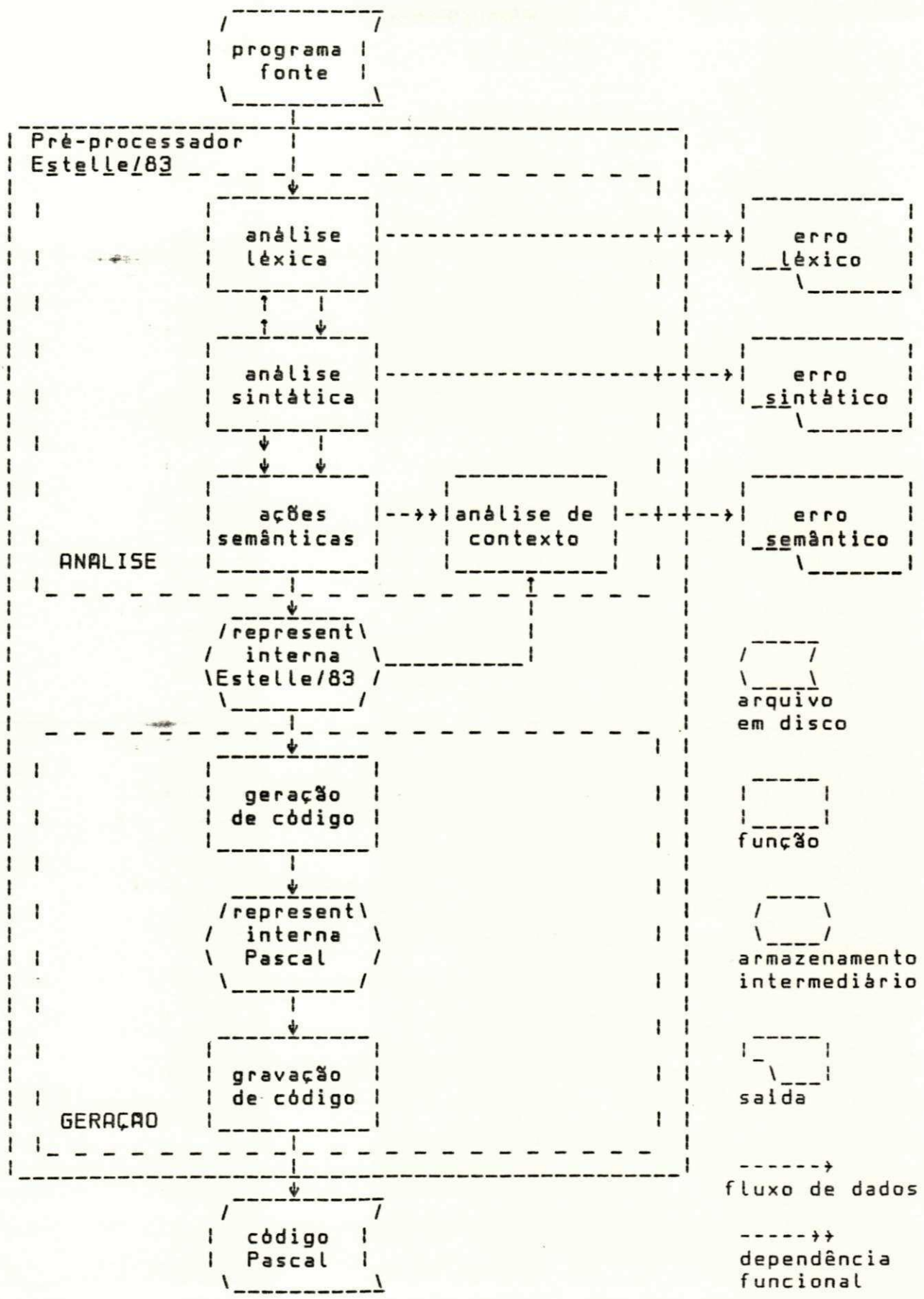

Figura 4.3: Estrutura interna do pré-processador Estelle/83.

(a) a **analise léxica** é responsável pela leitura da especificação, em Estelle/83, e pelo reconhecimento de cada **item léxico** dentro dessa especificação;

(b) a **analise sintática** consome os itens léxicos gerados pelo analisador léxico, agrupando-os em diversas **unidades sintáticas**. A verificação da pertinência (ou não) dessas unidades sintáticas é realizada com base na BNF de Estelle/83;

(c) as **ações semânticas** são acionadas pelo analisador sintático e criam uma representação interna para as estruturas sintáticas Estelle/83 e

(d) a **análise de contexto**, acionada pelas ações semânticas, verifica a correção semântica dessas estruturas internas.

As funções (a), (b) e (d) verificam, respectivamente, os erros léxicos, sintáticos e de contexto. Na ocorrência de qualquer tipo de erro, uma mensagem apropriada é emitida ao usuário e a execução do compilador é interrompida. A metodologia escolhida, para o tratamento de erros, é a mais simples possível: a cada erro sinalizado, cabe ao usuário efetuar as correções apropriadas na especificação fonte, reiniciando em seguida todo o procedimento de compilação.

Uma vez terminada a atividade de análise, tem início a atividade de geração de código. Nessa etapa, duas funções principais podem ser distinguidas (Fig.4.3):

(a) a **geração de código** transforma a representação interna, gerada durante a atividade de análise, numa estrutura próxima

à linguagem de implementação (Pascal) e

(b) a **gravação de código** gera um segmento de programa Pascal livre de erros léxicos e sintáticos.

O pré-processador realiza somente a análise de contexto, relativa às construções introduzidas pela TDF Estelle/83. A análise de contexto, relativa às construções da linguagem Pascal, é responsabilidade do compilador Pascal. A detecção desse tipo de erro, no código de implementação, não cria maiores dificuldades para a sua correção, a nivel da especificação fonte, já que esta última está refletida na implementação gerada.

4.2 A construção do compilador

A linguagem Prolog ("Programming in Logic") [ClMe 81] tem sido utilizada na construção de ferramentas que auxiliam a especificação formal, a verificação, a implementação e os testes de protocolos. Exemplos dessas ferramentas podem ser encontrados em [BoDs 85], [JaMo 85] e [BaSa 87].

O pré-processador Estelle/83 foi totalmente desenvolvido em Prolog. As vantagens do emprego dessa linguagem, nesse tipo de aplicação, em lugar de ferramentas usuais para a construção de compiladores, são [Warr 80]:

(a) a atividade de programação em Prolog é simples, não sendo necessário muito tempo para a obtenção de um produto final;

(b) a manutenção e/ou extensão de um programa escrito em Prolog pode ser também muito simples;

(c) há um mapeamento direto entre a função de análise sintática e a sua implementação em Prolog e

(d) A linguagem Prolog tem sido usada satisfatoriamente na construção de ferramentas de simulação para protocolos de comunicação. Como será visto no Capitulo 6, o compilador Estelle/83 será, futuramente, integrado a um sistema desse tipo.

A fim de verificar a viabilidade do uso do Prolog, para a construção do pré-processador Estelle/83, o analisador sintático foi desenvolvido, inicialmente, em micro-Prolog [Clar 84]. Apesar da confirmação, nesse trabalho [LoFe 86], das potencialidades do uso de Prolog, o micro-Prolog não atendeu algumas caracteristicas desejadas. O maior problema encontrado foi a limitação, imposta pelo micro-Prolog, em relação à memória reservada para área de trabalho (64K). Optou-se, então, pelo interpretador Waterloo-Prolog [Wate ..], que é usado em ambiente CMS, num computador IBM 4341, em um ambiente com 3Mb de memória disponivel para cada máquina virtual.

A Figura 4.4 apresenta um subconjunto das cláusulas Prolog responsável pelo gerenciamento da execução do compilador Estelle/83.

A implementação do protocolo, especificado em Estelle/83, é gerada em Pascal/VS [IBM ..a, IBM ..b], que atua também no ambiente CMS do IBM 4341. Para tornar esse código o mais próximo possivel das construções do Pascal padrão [ISO 83d], apenas um subconjunto das estruturas do Pascal/VS foi utilizado. A BNF desse subconjunto é apresentado no Anexo B.

```
estelle <-
   nome_arquivo_fonte &
   analise_sintatica &
   geracao_de_codigo &
   nome_arquivo_saida &
   imprime_codigo &
   compila_codigo.

nome_arquivo_fonte <-
   read(X) &
   addax(arquivo(X)).

analise_sintatica <-
   . . .

geracao_de_codigo <-
   . . .

nome_arquivo_saida <-
   read(X) &
   addax(saida(X)).

imprime_codigo <-
   . . .

compila_codigo <-
   saida(X) &
   system(pascalvs.X.nil).
```

Figura 4.4: Subconjunto das cláusulas Prolog responsáveis pelo gerenciamento da execução do compilador Estelle/83.

4.2.1 Os analisadores léxico e sintático

Em [Moni 84] são apresentadas várias técnicas para o desenvolvimento de analisadores léxicos e sintáticos, todas elas baseadas em **Gramáticas de Cláusulas Definidas** ("Definite Clause Grammars - DCGs") [StSh 86]. As DCGs são uma generalização das **Gramáticas livres de contexto**.

A TDF Estelle/83 é definida através de uma gramática do tipo LL(1). Nesse tipo de gramática, não há ambiguidades na escolha das **regras de produção**, que gerarão a árvore sintática do programa que está sendo compilado [AhUl 79]. Isso facilita a

construção, em Prolog, do analisador sintático para essa TDF [LoFe 87b], não sendo preciso o emprego de todo o poder das DCGs. Por exemplo, a regra de produção mostrada na Figura 4.5 pode ser traduzida, em Prolog, na forma apresentada na Figura 4.6.

```
. . .
‹module› ::= 'module' ‹IDENT› ';' ‹portlist› 'end' ‹IDENT›.
. . .
‹portlist› ::= . . .
. . .
```

Figura 4.5: Exemplo de regra de produção da BNF da TDF Estelle/83.

```
. . .
module <-
  terminal('module',n) &
  terminal('IDENT',s) &          as(15) &
  terminal(';',s) &
  portlist &                     as(16) &
  terminal('end',s) &
  terminal('IDENT',s) &          as(17).
. . .
```

Figura 4.6: Cláusula, em Prolog, que traduz a regra de produção da BNF da TDF Estelle/83.

O tratamento dos símbolos terminais é feito pelo predicado **terminal(x,y)**, onde x indica o item léxico esperado e y indica se a "não ocorrência" do item léxico deve (s) ou não deve (n) ser considerado um erro sintático. A Figura 4.7 apresenta um subconjunto das cláusulas Prolog que implementam o predicado **terminal**. A cláusula definida nas linhas 14-18 efetuam o reconhecimento de um dos tipos de itens léxicos válidos: o IDENTIFICADOR. Os predicados pré-definidos para a manutenção do banco de dados (**addax** e **delax**) são relativamente lentos, mas são os únicos disponíveis no Waterloo-Prolog.

```
01 terminal(X,A) <-
02     le_token(Y) &
03     / &
04     testa(X,Y,A).
05
06 le_token(X) <-
07     token(X) &
08     ne(X,'') &
09     /.
10 le_token(X) <-
11     item_lexico(X) &
12     addax(token(X),1).
13 . . .
14 testa('IDENT',X,*) <-
15     string(X,'D'.Y) &
16     string(Z,Y) &
17     grava_token(Z) &
18     /.
19 . . .
20 testa(X,*,'s') <-
21     erro_sintatico(X).
22 . . .
23 grava_token(X) <-
24     delax(tk(Y)) &
25     append(Y,X.nil,Z) &
26     addax(tk(Z)) &
27     delax(token(*)).
28 . . .
29 erro_sintatico(X) <-
30     write('*** ERRO SINTATICO *** '.X.' ESPERADO') &
31     stop.
```

Figura 4.7: Subconjunto das cláusulas Prolog que implementam o predicado **terminal**.

Neste trabalho, o analisador léxico foi definido como um autômato finito [SeMe 83]. Os itens léxicos são gerados pelo analisador léxico, a partir de solicitações do analisador sintático. Os tipos de itens léxicos da TDF Estelle/83 são os mesmos da linguagem Pascal, acrescidos das palavras reservadas próprias a essa TDF. A Figura 4.8 apresenta um subconjunto das cláusulas Prolog que implementam o analisador léxico do compilador Estelle/83. As linhas 08-41 representam a parte do autômato finito responsável pela construção dos itens léxicos do tipo IDENTIFICADOR.

```
01 item_lexico(X) <-
02     repeat &
03         le_caracter(V) &
04         ne(V,' ') &
05         estado_0(V,X) &
06     ne(X,'&').
07 . . .
08 estado_0(X,Y) <-
09     letter(X) &
10     le_caracter(V) &
11     estado_1(V,X.nil,Y) &
12     /.
13 . . .
14 estado_1(X,Y,Z) <-
15     (letter(X) | digit(X)) &
16     le_caracter(V) &
17     append(Y,X.nil,W) &
18     / &
19     estado_1(V,W,Z).
20 estado_1('_',Y,Z) <-
21     le_caracter(V) &
22     append(Y,'_'.nil,W) &
23     estado_2(V,W,Z) &
24     /.
25 estado_1(X,Y,Z) <-
26     palavra_reservada(Y) &
27     string(X,'R'.Y) &
28     devolve(X) &
29     /.
30 estado_1(X,Y,Z) <-
31     string(X,'D'.Y) &
32     devolve(X).
33
34 estado_2(X,Y,Z) <-
35     (letter(X) | digit(X)) &
36     le_caracter(V) &
37     append(Y,X.nil,W) &
38     estado_1(V,W,Z) &
39     /.
40 estado_2(*,*,*) <-
41     erro_lexico('IDENTIFICADOR INVALIDO').
42 . . .
43 le_caracter(X) <-
44     delax(arqlex(X)) &
45     /.
46 le_caracter(X) <-
47     arquivo(A) &
48     readch(X,A).
49
50 devolve(X) <-
51     addax(arqlex(X)).
52
53 erro_lexico(X) <-
54     write('*** ERRO LEXICO *** '.X).
55 . . .
```

Figura 4.8: Subconjunto das cláusulas Prolog que implementam o analisador léxico do compilador Estelle/83.

Em alguns pontos do analisador sintático são inseridas algumas **ações semânticas (as)**. Essas ações são responsáveis pela geração da representação interna Estelle/83, em termos de cláusulas Prolog, da especificação de entrada. A Figura 4.9 mostra um conjunto dessas ações semânticas.

```
as(1) <-
   limpa_tk &
   /.
. . .
as(15) <-
   tk(X.nil) &
   addax(modulo(X,nil,nil)) &
   addax(portlist(nil)) &
   delax(refinamento(Y)) &
   append(Y,('module'.X).nil,Z) &
   addax(refinamento(Z),1) &
   /.
as(16) <-
   delax(portlist(X)) &
   delax(modulo(Y,nil,nil)) &
   addax(modulo(Y,X,nil)) &
   /.
as(17) <-
   tk(X.nil) &
   delax(modulo(Y,Z,nil)) &
   addax(modulo(Y,Z,X)) &          s04(X) &
   limpa_tk &
   /.
. . .
limpa_tk <-
   delax(tk(*)) &
   addax(tk(nil)).
. . .
```

Figura 4.9: Exemplos de ação semântica.

A cláusula tk contém a lista de itens léxicos, acumulados no decorrer do desenvolvimento da árvore sintática, que serão utilizados na construção da representação interna Estelle/83. As estruturas **modulo** e **refinamento** serão detalhados adiante.

As ações semânticas acionam o analisador de contexto, que é responsável pelos testes semânticos das estruturas próprias à TDF

Estelle/83 (módulo, canal, processo e refinamento). Os testes, considerados nesse trabalho, são os mesmos efetuados em [Gerb 83]. A Figura 4.10 mostra um desses testes.

```
. . .
s04(X) <-
   modulo(X,*,X) &
   /.
s04(*) <-
   erro_semantico(1).
. . .
```

Figura 4.10: Exemplo de teste de contexto.

As estruturas internas Estelle/83, geradas pelas ações semânticas, devem ser maleáveis, facilitando a sua manipulação pelo gerador de código.

Para a representação de canais é usada uma estrutura **canal** com quatro termos:

(a) o nome do canal;

(b) uma lista de papéis que esse canal deve desempenhar;

(c) uma lista com os papéis desse canal, associados às suas respectivas mensagens e

(d) um identificador, indicando o encerramento da especificação do canal, que deve ser o próprio nome do canal.

A Figura 4.11 apresenta a estrutura interna Estelle/83, gerada a partir da especificação de alguns canais, relativa ao exemplo usado até agora.

```
. . .
canal(EMISSOR_MEIO,
      USUARIO.PROVEDOR.nil,
      (USUARIO.MENSAGEM.(.M.:.MSG_TYPE.).nil)
      .(PROVEDOR.RECONHECIMENTO.nil).nil,
      EMISSOR_MEIO).
canal(RECEPTOR_USUARIO,
      USUARIO.PROVEDOR.nil,
      (USUARIO.RECONHECIMENTO.nil)
      .(PROVEDOR.MENSAGEM.(.M.:.MSG_TYPE.).nil).nil,
      RECEPTOR_USUARIO).
...
```

Figura 4.11: Exemplo de estruturas internas Estelle/83 para especificações de canais.

Para a representação de módulos é usada uma estrutura modulo com os seguintes termos:

(a) o nome do módulo;

(b) uma lista de portas com os canais (estruturados ou não) associados e

(c) um identificador, indicando o encerramento da especificação do módulo, que deve ser o próprio nome do módulo.

Para os módulos do exemplo considerado, algumas de suas estruturas internas Estelle/83 são apresentadas na Figura 4.12.

```
. . .
modulo(EMISSOR,
       (PORTA_E.nil.EMISSOR_MEIO.(USUARIO).nil).nil,
       EMISSOR).
modulo(RECEPTOR,
       (PORTA_R.nil.RECEPTOR_MEIO.(USUARIO).nil).nil,
       RECEPTOR).
. . .
```

Figura 4.12: Exemplo de estruturas internas Estelle/83 para especificações de módulos.

Para a representação de refinamentos são usados quatro tipos de cláusula: **refinamento**, **instance_ref**, **connect_ref** e **replace_ref** (Fig. 4.13).

A cláusula **refinamento** contém um único termo, que é uma lista. A cabeça dessa lista é uma estrutura **ref** que contém:

(a) o nome do refinamento;

(b) os parâmetros do refinamento;

(c) o nome do módulo cujo comportamento é definido por esse refinamento e

(d) um identificador, indicando o encerramento da especificação do refinamento, que deve ser o próprio nome do refinamento.

A **cauda** da lista contém as especificações dos canais, dos módulos, dos processos e de novos refinamentos (internos a esse refinamento).

A estrutura **instance_ref** contém dois termos:

(a) o nome do refinamento do qual ela faz parte e

(b) uma lista de ocorrências ("instance") de módulos, acompanhados de seus respectivos processos (ou refinamentos).

A estrutura **connect_ref** contém dois termos:

(a) o nome do refinamento do qual ela faz parte e

(b) uma lista de conexões entre ocorrências de módulos, sendo que cada ocorrência de módulo é acompanhada da porta através da qual se dará a conexão.

```
. . .

refinamento(ref(nil,nil,nil,nil)
            .(module.SISTEMA)
            .(refinement.REF_SISTEMA).nil).
refinamento(ref(REF_SISTEMA,nil,SISTEMA,REF_SISTEMA)
            .(channel.EMISSOR_MEIO)
            .(channel.RECEPTOR_MEIO)
            .(module.EMISSOR)
            .(refinement.REF_EMISSOR)
            .(module.RECEPTOR)
            .(module.MEIO_DE_TRANSMISSAO)
            . . . . .nil).
refinamento(ref(REF_EMISSOR,nil,EMISSOR,REF_EMISSOR)
            .(channel.PROTOCOLO_USUARIO)
            .(channel.PROTOCOLO_TEMPORIZADOR)
            .(module.USUARIO)
            .(process.PROC_USUARIO)
            .(module.TEMPORIZADOR)
            .(process.PROC_TEMPORIZADOR)
            .(module.PROTOCOLO)
            .(process.PROC_PROTOCOLO).nil).
. . .
instance_ref(REF_EMISSOR,
             ((T.nil).TEMPORIZADOR.PROC_TEMPORIZADOR.nil)
             .((U.nil).USUARIO.PROC_USUARIO.nil)
             .((P.nil).PROTOCOLO.PROC_PROTOCOLO.nil).nil).
. . .
connect_ref(REF_EMISSOR,
            ((T.PORTA_T.nil).P.PORTA_T.nil)
            .((U.PORTA_U.nil).P.PORTA_U.nil).nil).
. . .
replace_ref(REF_EMISSOR,((PORTA_E.nil).P.PORTA_M.nil).nil).

. . .
```

Figura 4.13: Exemplo de estruturas internas Estelle/83 para especificações de refinamentos.

A estrutura **replace_ref** também contém dois termos:

(a) o nome do refinamento do qual ela faz parte e

(b) uma lista de vinculações entre as portas do módulo pai com as portas das ocorrências dos módulos filhos (submódulos).

Para a representação de processos são usadas estruturas que descrevem:

(a) o cabeçalho da declaração;

(b) as portas, do módulo que o processo define, que são associadas a filas (caso a comunicação seja através de filas);

(c) os rótulos;

(d) os tipos de dados;

(e) as variáveis;

(f) os procedimentos e funções;

(g) as primitivas de baixo nível;

(h) a inicialização do módulo e

(i) as transições.

A Figura 4.14 ilustra a representação interna Estelle/83, para um dos processos do exemplo até aqui considerado.

A cláusula **process** contém quatro termos:

(a) o nome do processo;

(b) os parâmetros do processo;

(c) o nome do módulo cujo comportamento é definido por esse processo e

(d) um identificador, indicando o encerramento da especificação do processo, que deve ser o próprio nome do processo.

```
. . .
process(PROC_PROTOCOLO,
        nil,
        PROTOCOLO,
        PROC_PROTOCOLO).
. . .
var_processo(PROC_PROTOCOLO,
             state.:.(.OCIOSO.,.ESPERANDO_RECONHECIMENTO.).;
             .nil).
. . .
init_body(PROC_PROTOCOLO,begin.state.:=.OCIOSO.end.;.nil).
. . .
trans(PROC_PROTOCOLO,
      (when.PORTA_U.'.'.MENSAGEM.nil)
      .(from.OCIOSO.nil)
      .(to.ESPERANDO_RECONHECIMENTO.nil).nil,
      nil,
      4).
trans(PROC_PROTOCOLO,
      (when.PORTA_M.'.'.RECONHECIMENTO.nil)
      .(from.ESPERANDO_RECONHECIMENTO.nil).
      .(to.OCIOSO.nil).nil,
      nil,
      5).
trans(PROC_PROTOCOLO,
      (when.PORTA_T.'.'.TEMPORIZADOR.nil).nil,
      nil,
      6).
. . .
trans_body(4,begin.nil).
trans_body(4,out.PORTA_M.'.'.MENSAGEM.(.M.).nil).
trans_body(4,;.out.PORTA_T.'.'.LIGA_TEMPORIZADOR.nil).
trans_body(4,end.nil).
trans_body(5,begin.nil).
trans_body(5,out.PORTA_T.'.'.DESLIGA_TEMPORIZADOR.nil).
trans_body(5,end.nil).
trans_body(6,begin.nil).
trans_body(6,out.PORTA_M.'.'.MENSAGEM.(.M.).nil).
trans_body(6,;.out.PORTA_T.'.'.LIGA_TEMPORIZADOR.nil).
trans_body(6,end.nil).
. . .
```

Figura 4.14: Exemplo de estruturas internas Estelle/83 para a especificação de um processo.

A cláusula queued contém dois termos:

(a) o nome do processo do qual ela faz parte e

(b) uma lista dos nomes das portas associadas às filas.

As primitivas de baixo nivel, que são definidas como procedimentos, são representadas por uma cláusula **procfunc_processo** com três termos:

(a) o nome do processo do qual ela faz parte;

(b) o cabeçalho do procedimento que a define e

(c) o identificador **primitive**, que indica a natureza desse procedimento.

A cláusula **initialize** contém dois termos:

(a) o nome do processo do qual ela faz parte e

(b) uma lista de instruções referente ao bloco que define essa inicialização.

As transições são representadas por dois tipos de estruturas. O primeiro tipo (**trans**) contém:

(a) o nome do processo do qual ela faz parte;

(b) as cláusulas da transição e

(c) um número inteiro que identifica as ações associadas à transição.

A cada estrutura do tipo **trans** está associado um conjunto de estruturas do tipo **trans_body**, que contém dois termos:

(a) o número de identificação da ação e

(b) as instruções dessa ação.

Para as declarações de rótulos, tipos de dados, funções e procedimentos, definidos como parte de um processo, são criadas estruturas que já estão na forma apropriada para a geração de código. Essas declarações serão consideradas como declarações globais.

4.2.2 Geração de código

Uma vez terminada a análise sintática, a especificação de entrada estará representada através de um conjunto de cláusulas Prolog. Essas cláusulas devem ser transformadas, a fim de representar a implementação, em Pascal, da especificação de entrada. As regras que definem as representações, em estruturas internas Pascal, das construções próprias à TDF Estelle/83, são apresentadas em [Gerb 83]. A Figura 4.15 apresenta um subconjunto das cláusulas que implementam essa função.

```
geracao_de_codigo <-
   gera_cabecalho &
   gera_declaracoes &
   gera_procedimentos_process &
   gera_procedimentos_estruturas_de_dados.

gera_cabecalho <-
   saida(X) &
   addax(segment(X.';'.nil).

gera_declaracoes <-
   gera_type &
   gera_var &
   gera_procfunc.

gera_type <-
   gera_CHANNEL &
   gera_sinais &
   gera_SOTYPE &
   gera_estruturas_auxiliares &
   gera_PROCESS &
   gera_POTYPE.

. . .                                                    (cont.)
```

```
gera_var <-
   addax(var_procfunc(nil,'P0VAR'.':'.'P1TYPE'.';'.nil)) &
   addax(var_procfunc(nil,'C0VAR'.':'.'C1TYPE'.';'.nil)) &
   addax(var_procfunc(nil,'S0VAR'.':'.'S1TYPE'.';'.nil)).

gera_procfunc <-
   gera_procfunc_auxiliares &
   gera_procfunc_especificadas.

. . .

gera_PROCESS <-
   addax(type_procfunc(nil,'PROCESS'.'='.'('.nil)) &
   process(X,*,*,*) &
   string(X,Y) &
   string(V,'P'.'0'.Y) &
   delax(type_procfunc(nil,'PROCESS'.P)) &
   append(P,V.',',nil,Q) &
   addax(type_procfunc(nil,'PROCESS'.Q)) &
   addax(npme_process(X,V)) &
   fail.
gera_PROCESS <-
   delax(type_procfunc(nil,'PROCESS'.X)) &
   append(Y,',',nil,X) &
   append(Y,')'.';'.nil,Z) &
   addax(type_procfunc(nil,'PROCESS'.Z)).

. . .
```

Figura 4.15: Subconjunto das cláusulas Prolog que implementam a função de geração de código.

O código gerado pelo pré-processador Estelle/83 não representa completamente o sistema especificado. Esse código deverá ser ligado ("linking") a um núcleo de exploração, aos procedimentos que implementam as primitivas de baixo nivel e aos procedimentos que auxiliam na manipulação das estruturas de dados geradas pelo pré-processador Estelle/83. Em Pascal/VS, cada uma dessas partes da implementação total é representada por um segmento.

Para a representação do segmento Pascal/VS, relativo à especificação de entrada Estelle/83, são usadas sete tipos de cláusulas (Fig. 4.16):

```
segment(EXEMPLO).
. . .
type_procfunc(nil,MSG_TYPE.=.'...'.;.nil).
. . .
var_procfunc(nil,P0VAR.:.P1TYPE.;.nil).
. . .
procfunc_processo(nil,
                  function.F0FUNCTION.(.INDPOS.:.integer.).:
                  .integer.;.nil,
                  extern.nil).
. . .
procfunc_processo(nil,
                  procedure.IOSISTEMA.(.var.P1VAR.:.P1TYPE.).;
                  .nil,
                  nil).
. . .
var_procfunc(IOSISTEMA,P2VAR.:.P1TYPE.;.nil).
. . .
pf_body_ok(IOSISTEMA,begin.nil).
pf_body_ok(IOSISTEMA,new.(.P0VAR.,.P0PROC_PROTOCOLO.).;.nil).
. . .
pf_body_ok(IOSISTEMA,end.nil).
. . .
```

Figura 4.16: Exemplo de estrutura interna Pascal.

(a) uma cláusula **segment** com o nome do segmento que está sendo gerado;

(b) uma cláusula **label_procfunc**, cujo primeiro termo é o nome do procedimento, ao qual o conjunto de rótulos pertence, e um outro termo que representa o conjunto de rótulos propriamente dito;

(c) um conjunto de cláusulas **const_procfunc**, cujo primeiro termo é o nome do procedimento, ao qual a constante pertence, e um outro termo que representa a constante propriamente dita;

(d) um conjunto de cláusulas **type_procfunc**, cujo primeiro termo é o nome do procedimento, ao qual o tipo de dado pertence, e um outro termo que representa o tipo de dado propriamente dito;

(e) um conjunto de cláusulas **var_procfunc**, cujo primeiro termo é o nome do procedimento, ao qual a variável pertence, e um outro termo que representa a variável propriamente dita;

(f) um conjunto de cláusulas **procfunc_processo**, cujo primeiro termo representa o nome da rotina, a qual o procedimento pertence; um segundo termo que representa o cabeçalho desse procedimento e um útimo termo que representa a presença (**nil**) ou a auxência (**extern**) ou a definição futura (**forward**) do corpo desse procedimento na especificação de entrada Estelle/83 e

(g) para cada cláusula do tipo **procfunc_processo**, de corpo presente, há um conjunto de cláusulas **pf_body_ok**. Cada cláusula **pf_body_ok** contém dois termos: o nome do procedimento e uma instrução relativa ao corpo desse procedimento.

Com exceção da cláusula **segment**, em todas as demais o uso de **nil**, como primeiro termo, indica que essa estrutura faz parte de uma declaração global.

Uma vez transformada a representação interna Estelle/83 na representação interna Pascal, um programa Pascal correspondente deve ser gerado. Essa função é efetuada em duas etapas:

(a) a geração de uma sequência de itens léxicos Pascal, a partir da representação interna Pascal e

(b) a gravação desses itens léxicos, que constituem a implementação do protocolo propriamente dita.

A Figura 4.17 apresenta um subconjunto de cláusulas Prolog que implementam a primeira etapa. As cláusulas apresentadas nas linhas 09-21 executam a transformação da representação interna Pascal, referente às declarações de tipos de dados, numa sequência de itens léxicos Pascal correspondente.

```
01 imprime_codigo <-
02    imprime_cabecalho &
03    imprime_label(nil) &
04    imprime_const(nil) &
05    imprime_type(nil) &
06    imprime_var(nil) &
07    imprime_procfunc(nil).
08 . . .
09 imprime_type(X) <-
10    delax(type_procfunc(X,Y.Z)) &
11    imprime_pascal('type'.Y.Z) &
12    imp_type(X) &
13   /.
14 imprime_type(*).
15
16 imp_type(X) <-
17    delax(type_procfunc(X,Z)) &
18    imprime_pascal(Z) &
19   / &
20   imp_type(X).
21 imp_type(*).
22 . . .
23 imprime_pascal(nil).
24 imprime_pascal(X.Y) <-
25    write(X,arqaux) &
26   / &
27   imprime_pascal(Y).
```

Figura 4.17: Subconjunto das cláusulas Prolog responsáveis pela geração da sequência de itens léxicos da implementação Pascal.

A Figura 4.18 apresenta um subconjunto das cláusulas responsáveis pela gravação da implementação Pascal. Para essa segunda etapa, a exemplo da análise sintática, é utilizada a BNF do subconjunto do Pascal empregado (01-22). As cláusulas apresentadas nas linhas 23-42 efetuam a identação do código Pascal gerado.

```
01 e_program <-
02    e_proghead &
03    e_seqblock.
04
05 e_proghead <-
06    e_terminal('segment') &          coluna &
07    e_terminal('IDENT') &
08    e_terminal(';') &                linha & linha.
09
10 e_seqblock <-
11    e_labeld &
12    e_constd &
13    e_typed &
14    e_vard &
15    e_procfuncd.
16
17 e_labeld <-
18    e_terminal('label') &            mais_2 & linha &
19    e_terminal('INTEGER') &
20    e_seqinteger &                   menos_2 & linha &
21    e_terminal(';').
22 . . .
23 coluna <-
24    e_grava_token(' ').
25
26 linha <-
27    n_tab(N) &
28    delax(col_imp(*)) &
29    addax(col_imp(N)) &
30    saida(A) &
31    newline(arquivo_de_saida) &
32    tab(N,A).
33
34 mais_2 <-
35    delax(n_tab(X)) &
36    Y is X + 2 &
37    addax(n_tab(Y)).
38
39 menos_2 <-
40    delax(n_tab(X)) &
41    Y is X - 2 &
42    addax(n_tab(Y)).
43
44 e_terminal(X) <-
45    . . . { semelhante ao "terminal" do analisador sintático }
46
47 e_grava_token(X) <-
48    saida(S) &
49    write(X,S) &
50    tamanho(X,N) &
51    delax(col_imp(A)) &
52    B is A + N &
53    addax(coluna_imp(B)).
54 . . .
```

Figura 4.18: Subconjunto das cláusulas Prolog responsáveis pela gravação da implementação Pascal.

No exemplo que está sendo utilizado, as declarações de tipos e variáveis, relativas ao segmento que implementa a especificação de entrada, são mostradas na Figura 4.19. Nas linhas 5-13 são declarados os canais (5-7) e suas respectivas mensagens (8-13). Nas linhas 16-27 é declarado o registro SOTYPE, no qual são definidas as estruturas de todas as mensagens trocadas entre os módulos do sistema, relativas a todos os canais (por exemplo, nas linhas 18-22). Nas linhas 41-54 é declarado o registro POTYPE, que representa todos os processos do sistema. Para cada processo representado, estão associadas suas variáveis locais (49-52). Finalmente, na linha 55 são declaradas as variáveis globais POVAR (que contém o endereço do processo corrente), COVAR (canal corrente) e SOVAR (mensagem sendo tratada).

Obs.: todo canal identificado pelo prefixo **CO**, seguido do nome de um processo, refere-se às transições espontâneas relativas a esse processo. Isto é, **CO** representa um canal que não veicula nenhum tipo de mensagem. Entretanto, é necessário criar-se uma pseudo-mensagem (RnANY), para cada transição espontânea, que seja responsável pelo acionamento do teste das condições habilitadoras dessa transição. No exemplo que está sendo desenvolvido não há transições desse tipo. Canais do tipo **CO** são declaradas nos anexos D, G e H.

```
01 . . .
02 type
03   MSG_TYPE = ...;
04   . . .
05   CHANNEL = (C1EMISSOR_MEIO,C2RECEPTOR_MEIO,
06              C3PROTOCOLO_USUARIO,C4PROTOCOLO_TEMPORIZADOR,
07              . . . );
08   S1EMISSOR_MEIO = (S1MENSAGEM,S1RECONHECIMENTO);
09   S2RECEPTOR_MEIO = (S2MENSAGEM,S2RECONHECIMENTO);
10   S3PROTOCOLO_USUARIO = (S3MENSAGEM);
11   S4PROTOCOLO_TEMPORIZADOR = (S4LIGA_TEMPORIZADOR,
12                               S4DESLIGA_TEMPORIZADOR,
13                               S4TEMPORIZACAO);
14   . . .
15   S1TYPE = ->S0TYPE;
16   S0TYPE = record NEXT : S1TYPE;
17              case CHANNEL of
18                C1EMISSOR_MEIO :
19                  (case T1EMISSOR_MEIO : S1EMISSOR_MEIO of
20                     S1MENSAGEM :
21                       (D1MENSAGEM : record M : MSG_TYPE end);
22                     S1RECONHECIMENTO : (); );
23                C2RECEPTOR_MEIO : ( . . . );
24                C3PROTOCOLO_USUARIO : ( . . . );
25                C4PROTOCOLO_TEMPORIZADOR : ( . . . );
26                . . .
27            end;
28   I1TYPE = ->I0TYPE;
29   C1TYPE = ->C0TYPE;
30   P1TYPE = ->P0TYPE;
31   I0TYPE = record NEXT : I0TYPE; IDENT : integer end;
32   A0TYPE = record PROC : P1TYPE; CHAN : C1TYPE end;
33   C0TYPE = record
34              IDENT : integer;
35              INDLIST : I1TYPE;
36              TARGET : A0TYPE
37            end;
38   P2TYPE = packed array (.1..10.) of char;
39   PROCESS = (P0PROC_USUARIO,P0PROC_TEMPORIZADOR,
40              P0PROC_PROTOCOLO);
41   P0TYPE = record
42              IDENT : P2TYPE;
43              CHANLIST : C1TYPE;
44              NEXT : P1TYPE;
45              case PROCESS of
46                P0PROC_USUARIO : ( . . . );
47                P0PROC_TEMPORIZADOR : ( . . . );
48                P0PROC_PROTOCOLO :
49                  (D0PROC_PROTOCOLO :
50                     record
51                       state:(OCIOSO,ESPERANDO_RECONHECIMENTO);
52                     end; );
53                . . .
54            end;
55 var P0VAR : P1TYPE; C0VAR : C1TYPE; S0VAR : S0TYPE;
```

Figura 4.19: Declarações de tipos de dados e de variàveis de um segmento Pascal gerado pelo pré-proces. Estelle/83.

Para cada processo da especificação de entrada, dois tipos de procedimentos são gerados (Fig. 4.20): o primeiro será utilizado na construção da estrutura de dados que refletirá a arquitetura do sistema (módulos, portas, canais e conexões - linhas 21-56), enquanto que o segundo implementa as transições do processo (17-19). As rotinas de suporte, que manipulam a estrutura de dados, estão definidas em outro segmento e, por isso, são declaradas como **external** (5-16).

Obs.: procedimentos que são declarados nas especificações de refinamentos ou nas especificações de processos, na implementação são declarados de forma global, conjuntamente com as declarações relativas às rotinas de suporte (ver anexos D, G e H).

A construção da estrutura de dados de um módulo refinado é realizada através das ligações das estruturas de dados de seus submódulos. Esses submódulos, por sua vez, podem ser refinados ou não. No primeiro caso, a construção de sua estrutura de dados é realizada de maneira análoga à do módulo. No segundo caso, a construção da estrutura de dados será composta de:

(a) uma estrutura **POTYPE**, que representa o submódulo;

(b) uma estrutura **COTYPE** para cada porta do submódulo, sendo que uma variável lógica indica se a porta está associada a uma fila (**true**) ou não (**false**) e

(c) uma inicialização das variáveis de entrada do submódulo.

Por exemplo, o submódulo PROTOCOLO não é refinado e a construção de sua estrutura de dados está descrita nas linhas 28-37 da Figura 4.20.

```
01 segment EXEMPLO;
02   . . .
03   type . . .
04   var . . .
05   procedure OUT; external;
06   function F0FUNCTION(INDPOS : integer) : integer; external;
07   procedure P0PROCEDURE(IDENT : integer); external;
08   procedure P1PROCEDURE(IDENT : P2TYPE; var P : P1TYPE);
09     external;
10   procedure P2PROCEDURE(IDENT : integer); external;
11   procedure P3PROCEDURE(Q : boolean; IDENT : integer);
12     external;
13   procedure P4PROCEDURE(INDVAL,INDPOS : integer); external;
14   procedure P6PROCEDURE(var X : A0TYPE); external;
15   procedure P7PROCEDURE( P : P1TYPE; var Q : P1TYPE);external;
16   procedure P8PROCEDURE( P : P1TYPE); external; . . .
17   procedure PROC_USUARIO; . . .
18   procedure PROC_TEMPORIZADOR; . . .
19   procedure PROC_PROTOCOLO; . . .
20   . . .
21   procedure IOSISTEMA(var P1VAR : P1TYPE);
22     var P2VAR : P1TYPE; A0VAR,A1VAR : A0TYPE;
23     . . .
24     procedure IOREF_EMISSOR(var P1VAR : P1TYPE);
25       var P2VAR : P1TYPE; A0VAR,A1VAR : A0TYPE;
26       procedure IOPROC_USUARIO(var P1VAR : P1TYPE); . . .
27       procedure IOPROC_TEMPORIZADOR(var P1VAR : P1TYPE); . . .
28       procedure IOPROC_PROTOCOLO(var P1VAR : P1TYPE);
29         begin
30           new(P0VAR,P0PROC_PROTOCOLO);
31           P1PROCEDURE('PROC_PROTO',P1VAR);
32           P3PROCEDURE(true,1);
33           P3PROCEDURE(true,2);
34           P3PROCEDURE(false,3);
35           with P0VAR->.D0PROC_PROTOCOLO do
36             begin state:=OCIOSO end
37         end;
38       begin P2VAR:=nil; IOPROC_USUARIO(P2VAR);
39         IOPROC_TEMPORIZADOR(P2VAR); IOPROC_PROTOCOLO(P2VAR);
40         P7PROCEDURE(P2VAR,P1VAR);
41         P5PROCEDURE(3,3,P1VAR); P6PROCEDURE(A0VAR);
42         P5PROCEDURE(1,1,P1VAR); P6PROCEDURE(A1VAR);
43         A0VAR.CHAN->.TARGET:=A1VAR;
44         A0VAR.CHAN->.TARGET:=A1VAR;
45         P5PROCEDURE(3,1,P1VAR); P6PROCEDURE(A0VAR);
46         P5PROCEDURE(2,1,P1VAR); P6PROCEDURE(A1VAR);
47         A0VAR.CHAN->.TARGET:=A1VAR;
48         A0VAR.CHAN->.TARGET:=A1VAR;
49         A0VAR.CHAN:=C0VAR;
50         P5PROCEDURE(3,2,P1VAR);
51         P6PROCEDURE(A0VAR.CHAN->.TARGET);
52         P8PROCEDURE(P1VAR)
53       end;
54     begin . . . end;
```

Figura 4.20: Declarações de funções e procedimentos de um segmento Pascal gerado pelo pré-processador Estelle/83.

No caso de um módulo (ou submódulo) refinado, as estruturas de dados de seus submódulos são ligadas da seguinte forma:

(a) para cada submódulo é realizada uma chamada ao procedimento que cria a sua estrutura de dados;

(b) é criado um cabeçalho temporário para o módulo refinado;

(c) para cada conexão, entre submódulos especificada, é criado um conjunto de instruções responsável por essa conexão;

(d) para cada vinculação, entre portas especificada, é criado um conjunto de instruções responsável por essa vinculação e

(e) são eliminadas as estruturas temporárias, que auxiliaram a construção da estrutura de dados do módulo.

Por exemplo, a construção das estruturas de dados do módulo refinado EMISSOR está descrita nas linhas 40-55 da Figura 4.20. A Figura 4.21 mostra algumas estruturas de dados, que representam as construções arquiteturais do módulo principal SISTEMA com seus processos, canais e portas internas.

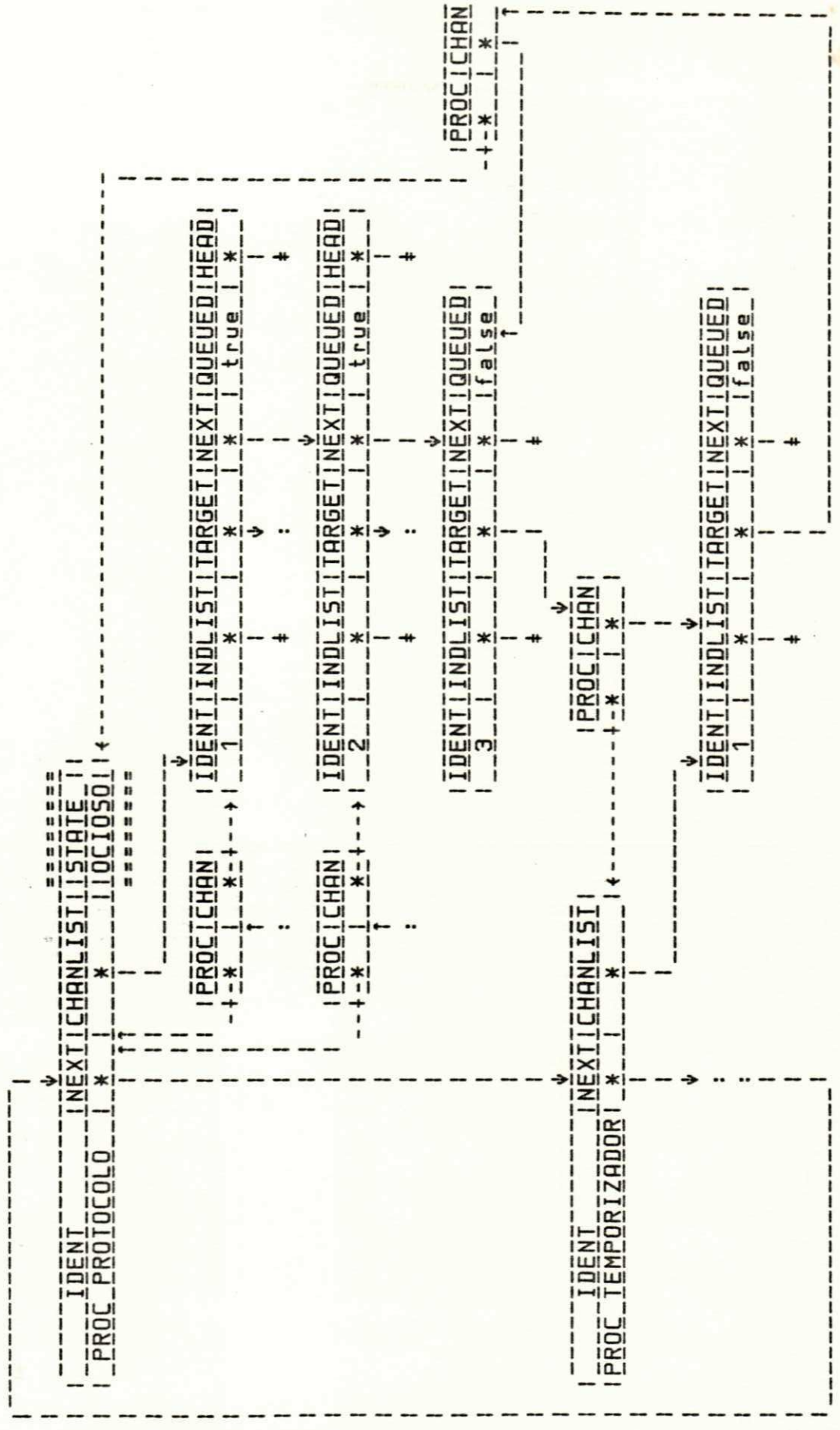

Figura 4.21: Estruturas de dados que representam as construções arquiteturais do exemplo utilizado.

O procedimento, que implementa as transições especificadas de um processo, é composto de um conjunto de instruções que:

(a) salvaguardam as estruturas de dados, referentes ao canal e a mensagem que estão sendo tratados, prevenindo-se, assim, contra uma possivel perda dos localizadores (ponteiros) dessas estruturas (o que acontece quando há uma interação de saida - **out**);

(b) implementam as transições propriamente ditas, cujo escopo é limitado, caso o processo possua variáveis, por uma instrução **with** e

(c) liberam a estrutura de dados, referente à mensagem que foi tratada.

Na Figura 4.22, o item (a) está descrito na linha 06, o item (b) nas linhas 07-26 e o item (c) nas linhas 27-33.

Em relação ao item (b), cabe salientar:

(a) a cláusula **provided** é implementada por uma instrução **if**, acompanhada da mesma expressão booleana presente nessa cláusula;

(b) a cláusula **when** é implementada por duas instruções **if**: a primeira verifica se o identificador do canal vigente é o mesmo identificador especificado na cláusula e a segunda verifica se os tipos, da mensagem que está sendo tratada e da mensagem especificada na cláusula, são os mesmos. Caso a mensagem especificada contenha parâmetros, uma instrução **with** é acrescentada, a fim de definir a estrutura dessa mensagem dentro do registro SOTYPE.

```
01 procedure PROC_PROTOCOLO;
02   label 1;
03   var C1VAR : C0VAR;
04       S1VAR : S0VAR;
05   begin
06     C1VAR:=C0VAR; S1VAR:=S0VAR;
07     with P0VAR->.D0PROC_PROTOCOLO do
08       begin
09         if C0VAR->.IDENT = 3 then
10           if S0VAR->.T3PROTOCOLO_USUARIO = S3MENSAGEM then
11             with S0VAR ->.D3MENSAGEM do
12               begin
13                 begin
14                   P0PROCEDURE(2);
15                   new(S0VAR,C3PROTOCOLO_USUARIO,S3MENSAGEM);
16                   S0VAR->.D3MENSAGEM:=M
17                 end;
18                 begin
19                   P0PROCEDURE(3);
20                   new(S0VAR,C1PROTOCOLO_USUARIO,S1MENSAGEM);
21                   S0VAR->.D1MENSAGEM:=M
22                 end;
23                 goto 1
24               end;
25         . . .
26       end;
27   1 : case C1VAR->.IDENT of
28         1 : . . .
29         3 : case S1VAR->.T3PROTOCOLO_USUARIO of
30               S3MENSAGEM :
31                 dispose(S1VAR,C3PROTOCOLO_USUARIO,S3MENSAGEM);
32             end;
33       end
34   end;
```

Figura 4.22: Corpo de um procedimento relativo às transições de um processo, gerado pelo pré-processador Estelle/83.

(c) a cláusula **from** é implementada por uma instrução **if**, que verifica se o estado vigente é o mesmo estado (ou se pertence ao conjunto de estados) definido na cláusula;

(d) a cláusula **to** é implementada atribuindo-se, à variável **state**, o valor referente ao novo estado principal presente na cláusula. Essa instrução é realizada no começo do bloco das instruções, que manipulam as variáveis de estado adicionais;

(e) a cláusula **with** de Estelle/83 e a instrução **with** do Pascal possuem a mesma semântica e, portanto, sua implementação é realizada diretamente e

(f) a cláusula **any** é implementada da mesma forma que a cláusula **when**, sendo que a especificação da variável, presente na cláusula, é considerada como um parâmetro da pseudo-mensagem RnANY, gerada para a transição espontânea.

Obs.: a cláusula **priority** não foi implementada neste trabalho.

Uma instrução **out** gera um bloco de instruções (13-17, 18-22), onde:

(a) é realizada uma chamada à rotina externa **POPROCEDURE**, que localiza o canal que será usado;

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

(c) é chamada a rotina **OUT**, que coloca a mensagem no local de recepção pertinente, ou diretamente na.

Ao final de cada uma das transições (normal ou espontânea) é feita a liberação dos registros de dados usados na transição (26-32).

O nosso exemplo é bem simples e não usa todos os recursos oferecidos pela TDF Estelle/83. Em casos mais próximos da realidade, esse código Pascal gerado pode tornar-se bem mais complexo. Os anexos D, G e I mostram implementações geradas, pelo compilador Estelle/83, das especificações apresentadas nos anexos C, F e H, respectivamente. Tais especificações, junto com suas respectivas implementações, serão abordados no próximo capitulo.

5. IMPLEMENTAÇAO E VALIDAÇAO DE UM PROTOCOLO ATRAVES DO COMPILADOR Estelle/83

Dois tipos de análise podem ser realizados com as especificações de protocolos [Boch 87b]:

(a) **estática**, que permite a validação léxica e sintática da especificação. Permite também a validação de alguns aspectos semânticos, tais como erros de contexto, relativos aos conceitos arquiteturais da TDF utilizada (**análise semântica estática**). No compilador Estelle/83, parte da análise semântica estática é feita pelo compilador Pascal e

(b) **dinâmica**, que permite investigar o comportamento da especificação, submetendo-a a um conjunto de cenários possiveis. A análise dinâmica pode ser realizada de forma **exaustiva** ou através de **simulação**. No primeiro caso, são considerados todos os cenários possiveis. Na simulação, apenas alguns desses cenários são considerados.

Cabe salientar que os tipos de erros, detectáveis através da análise dinâmica, não podem ser percebidos na análise estática, pois esta última não leva em consideração o ambiente distribuido no qual o protocolo irá atuar.

Neste trabalho, foram efetuadas uma análise estática e duas análises dinâmicas, por simulação, da especificação do protocolo (do tipo "bit-alternante"), apresentada no Anexo C. A Figura 5.1 descreve os módulos, utilizados na especificação desse protocolo, as mensagens, utilizadas para as interações entre os módulos, e as primitivas de baixo nivel, utilizadas para as interações entre esses módulos e o ambiente no qual esse protocolo será inserido.

A análise estática da especificação do protocolo foi feita através de sua compilação sucessiva, até que todos os erros léxicos, sintáticos e de contexto foram erradicados. Um código executável foi alcançado através da ligação ("linking") dos seguintes objetos:

(a) o código gerado pelo compilador Pascal, a partir da implementação do protocolo (Anexo D);

(b) o código do um núcleo de exploração, adaptado para essa implementação (Anexo E);

(c) o código das rotinas de suporte (Anexo K), que auxiliam na construção e manipulação das estruturas de dados, que por sua vez foram definidas pela implementação e

(d) o código das primitivas de baixo nivel.

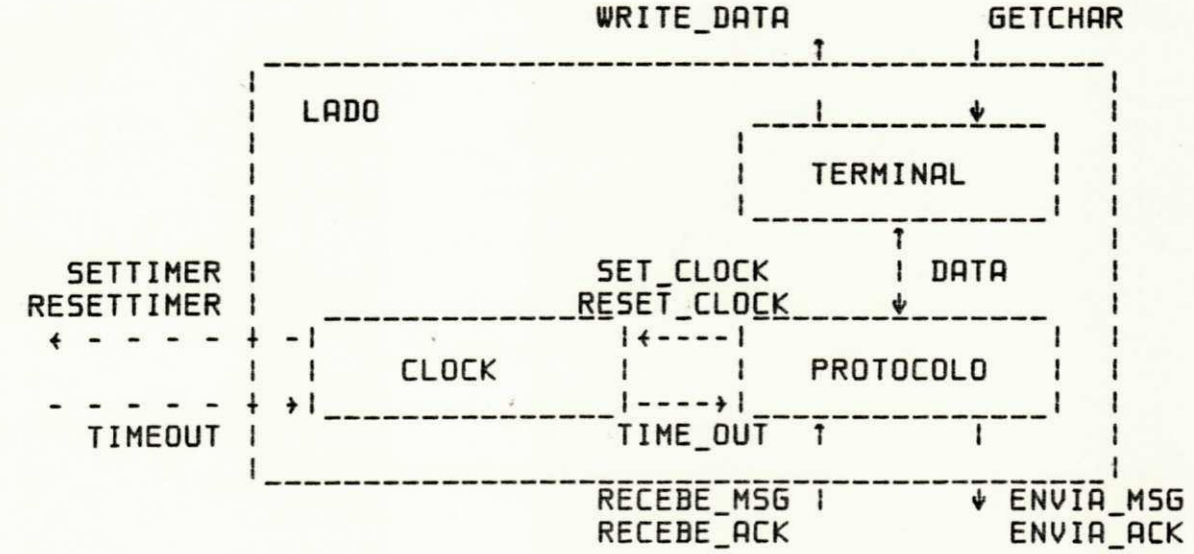

Figura 5.1: Descrição dos módulos e das interações (mensagens e primitivas de baixo nivel) do protocolo exemplo.

Para a análise dinâmica, optou-se pela especificação de um sistema composto de duas ocorrências desse protocolo, que irão interagir através de um módulo, que simula o meio de comunicação (Fig. 5.2).

Figura 5.2: Módulos do sistema a ser simulado para a análise dinâmica do protocolo exemplo.

Inicialmente, esse meio foi considerado livre de perdas e de atrasos de transmissão. O seu comportamento é especificado na Figura 5.3.

```
process EX_MEIO for MEIO;

  trans
    when PORT(.B.).MSG
      begin
        out PORT(.not B.).MSG(M)
      end;
    when PORT(.B.).ACK
      begin
        out PORT(.not B.).ACK
      end;

end EX_MEIO;
```

Figura 5.3: Especificação do comportamento do meio de transmissão ideal.

A especificação do sistema completo é apresentada no Anexo F. Nessa especificação, as primitivas de baixo nivel, trocadas entre os submódulos PROTOCOLO e o ambiente computacional, foram substituidas por mensagens, agora trocadas entre os submódulos PROTOCOLO e o módulo MEIO_DE_TRANSMISSAO. O código Pascal, gerado a partir dessa especificação, é mostrado no Anexo G.

Na segunda análise dinâmica, foi considerado um meio de transmissão sujeito a perdas de informação. O comportamento desse novo módulo MEIO_DE_TRANSMISSAO é mostrado na Figura 5.4. A especificação completa desse sistema é mostrado no Anexo H e sua implementação, em Pascal, é mostrada no Anexo I.

```
process EX_MEIO for MEIO;

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

  trans
    when PORT(.B.).MSG
    provided not PERDA_NO_MEIO
      begin
        out PORT(.not B.).MSG(M)
      end;
    when PORT(.B.).ACK
    provided not PERDA_NO_MEIO
      begin
        out PORT(.not B.).ACK
      end;

end EX_MEIO;
```

Figura 5.4: Especificação do comportamento do meio de transmissão sujeito a perdas de informação.

Nas análises realisadas foi utilizado o núcleo de exploração apresentado na Figura 5.5. Esse núcleo contém:

```
001 program NUCLEO(input,output);
002   . . .
003   type . . .
004        CHANNEL = ( . . . );
005        . . .
006        S1TYPE = ->S0TYPE;
007        S0TYPE = record NEXT : S1TYPE;
008                   case CHANNEL of
009                     . . .
010                 end;
011        I1TYPE = ->I0TYPE;
012        C1TYPE = ->C0TYPE;
013        P1TYPE = ->P0TYPE;
014        I0TYPE = record NEXT : I1TYPE; IDENT : integer end;
015        A0TYPE = record PROC : P1TYPE; CHAN : C1TYPE end;
016        C0TYPE = record IDENT : integer; INDLIST : I1TYPE;
017                   TARGET : A0TYPE;
018                   case QUEUED : boolean of
019                     false : ();
020                     true : (HEAD : S1TYPE)
021                 end;
022        P2TYPE = packed array (.1..10.);
023        P0TYPE = record IDENT : P2TYPE; CHANLIST : C1TYPE;
024                   NEXT,REFINEMENT : P1TYPE
025                 end;
026   var P0VAR : P1TYPE; C0VAR : C1TYPE; S0VAR : S1TYPE;
027       SIGNAL_PENDING : integer;
028       . . .
029   . . .
030   procedure OUT : external; procedure OUT;
031     procedure RENDEZVOUS;
032       var SAVEPROCESS : P1TYPE; SAVECHANNEL : C1TYPE;
033       begin
034         SAVEPROCESS:=P0VAR; SAVECHANNEL:=C0VAR;
035         P0VAR:=C0VAR->.TARGET.PROC;
036         C0VAR:=C0VAR->.TARGET.CHAN;
037         . . .
038         C0VAR:=SAVECHANNEL; P0VAR:=SAVEPROCESS
039       end;
040     begin
041       with C0VAR->.TARGET.CHAN-> do
042         if QUEUED then begin
043           SIGNAL_PENDING:=SIGNAL_PENDING+1;
044           S0VAR->.NEXT:=HEAD; HEAD:=S0VAR end
045         else RENDEZVOUS
046     end;
047   procedure SYSTEM_INIT;
048     var P : P1TYPE; I : I1TYPE; N : integer;
049         TITLE,FIRST,OK : boolean;
050     begin
051       P:=nil;
052       . . .
053       P0VAR->.REFINEMENT; dispose(P); P:=P0VAR;
054       SIGNAL_PENDING:=0; TITLE:=true; OK:=false; N:=0;
055       repeat
056         N:=N+1; C0VAR:=P->.CHANLIST;                    (cont.)
```

```
057         while COVAR <> nil do begin
058           if COVAR->.TARGET.PROC <> nil then
059             if COVAR->.IDENT > 0 then begin
060               if TITLE then begin
061                 TITLE:=false; writeln;
062                 writeln(' *** DANGLING CONNECTIONS ***');
063                 writeln;
064                 writeln(' process       channel index');
065                 writeln(' seqnr/id       seqnr  intval') end;
066               write(N:3,'/',P->.IDENT,COVAR->.IDENT:6);
067               I:=COVAR->.INDLIST;
068               if I <> nil then begin
069                 write('    ('); FIRST:=true;
070                 repeat
071                   if FIRST then FIRST:=false else write(',');
072                   write(I->.IDENT:1); I:=I->.NEXT
073                 until I = nil;
074                 write(')') end;
075               writeln end
076             COVAR:=COVAR->.NEXT end;
077         if P->.NEXT <> nil then P:=P->.NEXT else OK:=true
078       until OK;
079       writeln; writeln('--------------------------------');
080       writeln;
081       if not TITLE then begin
082         P->.NEXT:=POVAR; COVAR:=POVAR->.CHANLIST end
083     end;
084   procedure SCHEDULER;
085     procedure GETSIGNAL;
086       var FOUND : boolean; S : S1TYPE;
087       begin
088         FOUND:= false; SIGNAL_PENDING:=SIGNAL_PENDING-1;
089         while COVAR = nil do begin
090           POVAR:=POVAR->.NEXT; COVAR:=POVAR->.CHANLIST end;
091         if COVAR->.QUEUED then begin
092           SOVAR:=COVAR->.HEAD;
093           if SOVAR <> nil then begin
094             FOUND:=true; S:=nil;
095             while SOVAR->.NEXT <> nil do begin
096               S:=SOVAR; SOVAR:=SOVAR->.NEXT end end end;
097         if not FOUND then CO->.HEAD:=nil else S->.NEXT:=nil
098       end;
099     begin
100       repeat
101         if SIGNAL_PENDING > 0 then begin
102           GETSIGNAL;
103           . . . end
104         else begin
105           POVAR:=POVAR->.NEXT; COVAR:=POVAR->.CHANLIST;
106           . . . end
107       until . . .
108     end;
109   begin
110     SISTEM_INIT; SCHEDULER
111   end.
```

Figura 5.5: Núcleo de exploração utilizado nas análises.

(a) um conjunto de declarações de rótulos, constantes, tipos de dados e variáveis referentes a esse módulo (002-028);

(b) um procedimento que implementa a interação de saida (mensagem), executada pela instrução **out** (030-046);

(c) um procedimento SYSTEM_INIT responsável pela chamada das rotinas da implementação, que constroem as estruturas de dados do sistema (047-083);

(d) um procedimento SCHEDULER responsável pelo gerenciamento das interações entre módulos e das interações dos módulos com o ambiente computacional (084-108) e

(e) um bloco de instruções, declarado a nivel global, que constroe as estruturas de dados do sistema, através do procedimento SYSTEM_INIT, e que dá inicio às interações entre os módulos, através do procedimento SCHEDULER.

Uma vez de posse do código Pascal, que implementa a especificação do protocolo considerado, algumas adaptações são feitas no núcleo de exploração:

(a) são declarados, se necessários os rótulos e as constantes auxiliares (002);

(b) são declarados os canais e suas respectivas mensagens, que representam, na especificação, as transições espontâneas (004-005);

(c) são declarados, como externos, os procedimentos gerados pelo pré-processador Estelle/83 e, quando necessário, são declaradas as rotinas auxiliares (029);

(d) são chamados, dentro do corpo do procedimento OUT, para a execução das interações entre os módulos, os procedimentos que, em função do processo corrente, traduzem o comportamento dos módulos (037);

(e) é chamado, dentro do procedimento SYSTEM_INIT, o procedimento que cria as estruturas de dados do sistema (052);

(f) são chamados, dentro do procedimento SCHEDULER, quando há mensagens pendentes entre módulos, os procedimentos que, em função do processo corrente, traduzem o comportamento dos módulos (103);

(g) é criada, para cada transição espontânea do processo corrente, uma estrutura de dados que representa uma pseudo-mensagem, que ativará o procedimento referente a esse processo e

(h) é definida uma expressão booleana que indica a condição de parada da execução da simulação.

Em ambas as simulações efetuadas, o núcleo de exploração utilizado foi o apresentado no Anexo J. Essas análises, para simplificar a implementação das primitivas de baixo nivel, foram desenvolvidas em Pascal e foram inseridas no segmento que contém o núcleo de exploração.

Os resultados da execução das duas análises dinâmicas estão listados nos Anexos L e M. O Anexo N apresenta o manual de utilização do compilador Estelle/83.

Para que se possa realizar uma avaliação mais contundente, do compilador Estelle/83, é necessário especificar e analisar um

protocolo mais complexo (por exemplo, Transporte). Entretanto, o tempo para se atingir uma especificação completa, em Estelle/83, e para se obter uma implementação semi-automática confiável de um protocolo desse porte, é relativamente longo. Essa avaliação deverá ser realizada durante os trabalhos de desenvolvimento dos protocolos do modelo OSI/ISO, que estão em curso junto ao Grupo de Redes de Computadores da Universidade Federal da Paraiba (GRC/UFPb).

6. APLICAÇOES DO COMPILADOR Estelle/83 EM SISTEMAS DE SIMULAÇAO

O compilador Estelle/83 pode ser integrado a um sistema mais amplo, que propicie um ambiente, para a simulação de especificações de protocolos, mais amigável. Essa simulação pode visar a validação das próprias especificações e do design do protocolo, assim como pode visar a análise do desempenho dessas especificações [LoFe 87a].

6.1 Validação de especificações através de simulação

Os métodos utilizados para a validação de protocolos estão normalmente relacionados às técnicas empregadas para a especificação dos mesmos. Tais métodos podem ser classificados em três categorias: análise lógica, simulação e teste [Boch 86].

Na primeira categoria encontram-se métodos (análise de alcançabilidade, derivação de invariantes, provas indutivas, execução simbólica, etc.) que utilizam um certo raciocínio lógico para verificar determinadas propriedades requeridas aos protocolos. Embora produzam resultados conclusivos, a aplicação exclusiva de tais métodos, em protocolos complexos, tem-se mostrado impraticável.

Em contrapartida, simulação e teste não fornecem resultados definitivos, já que não podem ser aplicados exaustivamente. Entretanto, eles podem ser utilizados em protocolos reais, dando-lhes uma certa credibilidade, que pode ser aumentada com análise lógica (ou vice-versa).

Se o objetivo é utilizar o compilador Estelle/83 para simular a execução de uma especificação, visando fornecer um certo grau de validação para a mesma, ao usuário cabe definir as propriedades a serem verificadas, ou os eventos a serem retidos para análise [JaMo 85]. Portanto, é importante que o sistema de simulação seja suficientemente flexível, para que o usuário possa descrever as diferentes propriedades e possa ter acesso às diferentes informações durante a simulação [Groz 86].

O sistema todo, composto da descrição de vários módulos, é então compilado e o resultado será um conjunto de estruturas de dados e procedimentos. Dentre os procedimentos alguns indicarão o estado do sistema e possíveis alternativas para o progresso do mesmo, enquanto que outros poderão executar ações atômicas, produzindo mudanças de estado.

As decisões a serem tomadas, no que diz respeito à execução do sistema, podem ser conduzidas segundo:

(a) uma estratégia de testes pré-definida pelo usuário. Tais estratégias podem ser programadas em Prolog e as ações associadas podem ser descritas através de um predicado Prolog;

(b) uma sequência de testes gerados automaticamente. Esta geração automática pode ser feita usando-se um sistema, capaz de encontrar algumas sequências de testes a partir de uma especificação de um protocolo em Estelle/83 [FaLi 86].

Quanto à verificação propriamente dita, duas alternativas são possíveis:

(a) "**on line**", através da execução em "paralelo" de **observadores**, que captam informações do sistema simulado, sem interferir na sua execução ou

(b) "**off line**", através da análise, por um verificador externo, dos eventos retidos.

Em geral, observadores são processos executados conjuntamente com o sistema alvo e que utilizam **sondas** para observar os pontos de acesso aos módulos, que compõem o sistema, captando as informações relativas às interações e ao estado do sistema global. A fim de facilitar a programação de observadores, podemos usar os conceitos de **observador local**, que observa parte do sistema composto, e de **observador global**, que reúne o resultado de cada uma das observações locais para a avaliação do estado global do sistema [DsBo 86].

Como ilustração, observadores serão utilizados no exemplo desenvolvido nos capítulos anteriores. Para observar a evolução da execução do sistema (elementos internos do módulo e interações), é definido um observador global, que recolhe as informações captadas por dois tipos de observadores locais, que utilizam sondas distintas (Fig. 6.1).

Inicialmente, é definido o objeto de observação (3). Em seguida são definidos os módulos locais, com as suas respectivas sondas (7-13), e o módulo global (15-18). Nesta etapa são utilizadas certas construções não inerentes a Estelle/83 e, portanto, não aceitas na versão atual do compilador. Posteriormente, os comportamentos desses módulos são descritos através de processos, que efetuam a observação propriamente dita

(20-30). Finalmente, são definidas as ocorrências dos observadores (32-34) e são realizadas as conexões entre as sondas dos observadores locais e os elementos do sistema a ser observado (37-38) e as conexões entre o observador global e os observadores locais (39-40).

```
01 simulation SIM;
02
03   import EXEMPLO;  (* sistema compilado separadamente *)
04
05   observation OBSVTN;
06
07     module OBS_M(OPC : boolean);
08       M : module EX_PROTOCOLO;
09     end OBS_M;
10
11     module OBS_C;
12       C : channel EMISSOR_MEIO;
13     end OBS_C;
14
15     module OBS;
16       O1 : module OBS_M;
17       O2 : module OBS_C;
18     end OBS;
19
20     local observer OBS_LOCAL_M for OBS_M;
21       ...
22     end OBS_LOCAL_M;
23
24     local observer OBS_LOCAL_C for OBS_C;
25       ...
26     end OBS_LOCAL_C;
27
28     global observer OBS_GLOBAL for OBS;
29       ...
30     end OBS_GLOBAL;
31
32     O_M : OBS_M with OBS_LOCAL_M;
33     O_C : OBS_C with OBS_LOCAL_C;
34     O_G : OBS   with OBS_GLOBAL;
35
36     probe
37       O_M.M  to ST1;
38       O_C    to ST1.R;
39       O_G.O1 to O_M;
40       O_G.O2 to O_C;
41
42   end OBSVTN;
43
44 end SIM;
```

Figura 6.1: Especificação de um módulo de observação usando um método formal do mesmo nivel do Estelle/83.

Dois tipos de observação são possíveis:

(a) **passiva**, onde os observadores fornecem um "tracing", a um observador externo, do sistema observado e não interfem na evolução da execução desse sistema e

(b) **ativa**, quando os observadores, em função das informações recolhidas e analisadas, escolhe alternativas que guiarão a execução do sistema.

Na Figura 6.2 são definidos o comportamento do observador local OBS_LOCAL_M e um trecho (relacionado a OBS_LOCAL_M) do comportamento do observador global OBS_GLOBAL. Essas definições visam a obtenção de um "tracing" para o sistema EXEMPLO.

```
local observer OBS_LOCAL_M for OBS_M;

  var  A : MSG_TYPE;
       T : 1..3;

  trans when PORT_U.MENSAGEM
        from OCIOSO
          begin A:=M; T:=1 end;
        when PORT_M.RECONHECIMENTO
        from ESPERANDO_RECONHECIMENTO
          begin T:=2 end;
        when PORT_T.TEMPORIZADOR
          begin T:=3 end;

end OBS_LOCAL_M;

global observer OBS_GLOBAL for OBS;

  trans with OBS_LOCAL_M
          begin
            case T of
              1 : writeln('enviou mensagem ',A);
              2 : writeln('recebeu reconhecimento');
              3 : writeln('estourou temporizador');
            end
         end;

end OBS_GLOBAL;
```

Figura 6.2: Definição dos observadores para a obtenção do "tracing".

6.2 Análise de desempenho através de simulação

Em função da escolha dos parâmetros, da análise e confiabilidade desejadas, é que se decide, normalmente, o tipo de metodologia a ser empregada para verificar a performance de um sistema.

Modelos analíticos (possivelmente baseados em cadeias de Markov ou redes de filas) fornecem uma boa interpretação dos parâmetros de performance (correspondentes às diferentes partes do sistema modelado) e de suas relações. Entretanto, as limitações impostas pelo modelo, em muitos casos, não correspondem à realidade.

Os modelos que podem ser empregados em simulação geralmente são mais próximos da realidade. Infelizmente, análises baseadas em simulação requerem uma grande quantidade de recursos computacionais e os resultados obtidos são, frequentemente, de difícil compreensão.

Medidas em sistemas reais podem ser realizadas com os mesmos objetivos da simulação, ou seja, estudar os casos onde as suposições efetuadas pelos modelos analíticos não são válidas, ou para determinar se tais modelos fornecem resultados realísticos.

Se o objetivo é utilizar o compilador Estelle/83 para simular a execução de um sistema (do tipo apresentado na Figura 6.3), visando fornecer um certo grau de análise de desempenho para o mesmo, é necessário antes desenvolver uma notação, que permita a inclusão à especificação do sistema, de declarações relacionadas à performance do sistema.

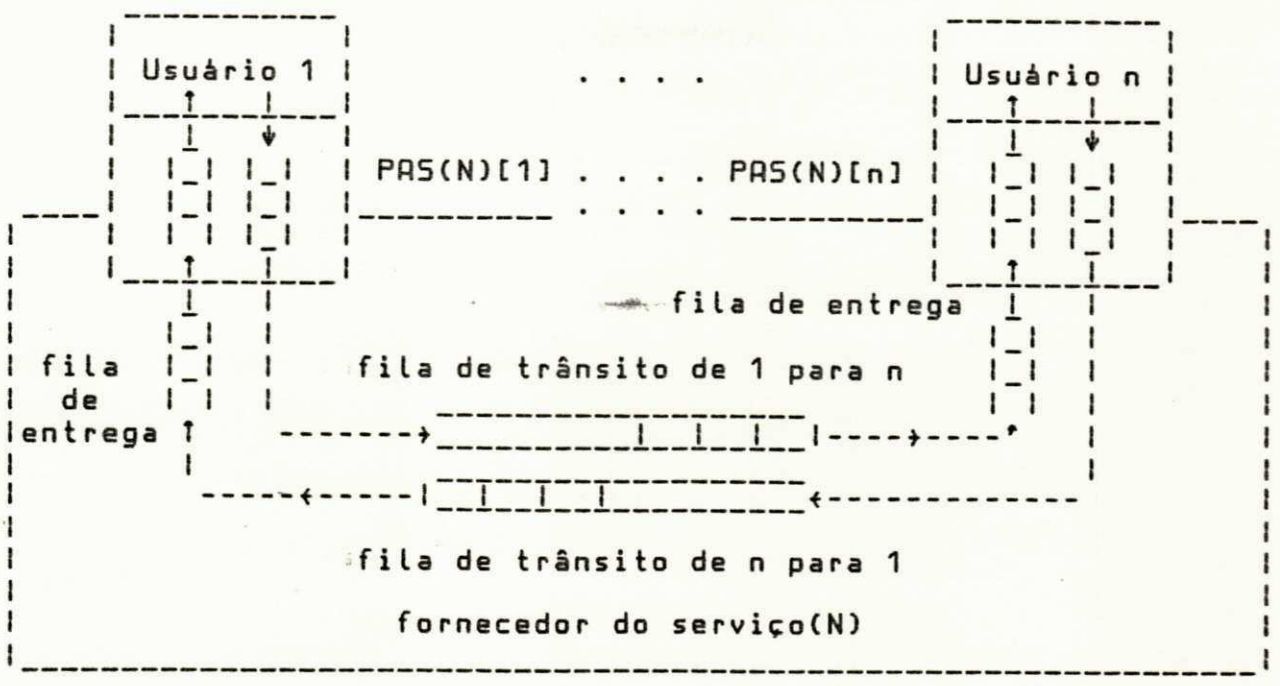

Figura 6.3: Sistema a ser simulado para a análise de desempenho.

Em [VaBo 84] algumas sugestões são propostas em relação à linguagem Estelle, a fim de permitir a inclusão de parâmetros de performance:

(a) o conceito de TIME, para que a duração das ações ou a duração de ocupação dos recursos sejam expressos. A inclusão desse conceito não acarreta nenhuma alteração sintática na linguagem, mas acarreta a inclusão da variável TIME ao código da implementação. Isso implica que não é possivel referenciá-la diretamente em uma expressão da especificação. A cada transição essa variável é atualizada pelo escalonador, ou pelo executor da simulação. Essa variável representa o relógio do sistema;

(b) a solicitação de recursos via cláusula HOLD (relacionada às transições) e que deve bloquear uma transição enquanto o recurso solicitado não estiver disponível. A clausula HOLD

tem a seguinte forma:

hold ‹núm. de unidades›
units ‹recurso›
of ‹intervalo de tempo›

Isto significa que o recurso requerido deve estar disponivel para se efetuar a transição. Caso não esteja, a transição é bloqueada até que os recursos se tornem disponiveis. A implementação desta cláusula pode ser feita com uma função ALOCA_RECURSOS(r,u,t), onde r é o ponteiro para o recurso desejado, u é a quantidade de recursos requerida e t o tempo necessário na utilização desses recursos;

(c) a especificação de **delays**, através de funções aleatórias, para modelar não determinismos e o uso de **delays** de transmissão nos canais (que modelam as filas) e

(d) o emprego da cláusula **DELAY**, pelas transições espontâneas, para indicar que a condição deve permanecer verdadeira durante um certo tempo, antes que a transição ocorra. Pode ser empregada também para modelar o ambiente, onde **delay** corresponderá ao intervalo de tempo entre requisições externas de serviço. A cláusula **DELAY ‹expressão›** pode ser implementada através de uma função que retorna **TRUE** ao expirar o intervalo de tempo estipulado quando da chamada dessa função.

Além dessas sugestões, que indicam uma linha de desenvolvimento em direção à simulação, trabalhos que visem uma melhor performance do compilador Estelle/83 podem ser realizados.

7. CONCLUSÃO

A partir da experiência obtida, durante o desenvolvimento do compilador Estelle/83 e durante as análises efetuadas, pode-se levantar algumas questões e algumas sugestões podem ser apresentadas para o prosseguimento deste trabalho.

O programa, em Prolog, do pré-processador Estelle/83, contém, aproximadamente, cinco mil linhas de código, distribuidos da seguinte forma:

função	linhas de código	número de cláusulas	número de procedimentos
gerenciamento.....	80	09	07
análise sintática.....	1200	288	124
análise léxica.....	400	86	34
ações semânticas.....	650	102	76
análise de contexto.....	500	137	45
geração de código Pascal.....	1200	227	109
gravação de código Pascal.....	1000	228	98
	~ 5000	1077	493

Deve-se salientar, porém, que 1050 linhas da análise sintática e 650 linhas da gravação de código são a transcrição direta, para o Prolog, das regras da BNF da TDF Estelle/83 e da BNF do subconjunto do Pascal, respectivamente.

Existe pouca bibliografia referente ao desenvolvimento, em Prolog, de compiladores da magnitude do Estelle/83 [Moni 84]. A facilidade encontrada na transcrição, em Prolog, das funções do pré-processador Estelle/83 pode ser contraposta ao tempo,

relativamente longo, dispensado, pelo autor, na busca de soluções para os seguintes problemas:

(a) encontrar um tratamento adequado, para o analisador sintático, dos nós terminais das regras de produção;

(b) encontrar uma estrutura adequada para as representações internas Estelle/83 e Pascal e, principalmente,

(c) encontrar um conjunto de regras adequado para a transformação da representação interna Estelle/83 na representação interna Pascal (função geração de código).

O tempo gasto na obtenção de cada código executável, referente às especificações apresentadas no capitulo 5, foi em torno de três minutos, atuando em sistema IBM 4341, com uso exclusivo. Esse tempo, excessivamente longo, constitui-se principalmente de tempo de entrada/saida e deve-se, em grande parte, ao uso de um interpretador Waterloo-Prolog (na época, única ferramenta adequada e disponivel na UFPb) para essa tarefa.

Outro problema, apresentado pelo interpretador Waterloo-Prolog é a necessidade excessiva de memória para a compilação. Para se ter uma idéia da gravidade desse problema, os exemplos apresentados no Capitulo 5 (Anexos G e I) necessitam quase que a totalidade dos recursos de memória disponiveis. Isso ocorre devido a ausência de um "coletor de lixo", responsável pela liberação das áreas de memória que já não estão sendo mais utilizadas.

Os "benchmarks" comparando os interpretadores Prolog de primeira geração, como o Waterloo-Prolog, com os mais recentes e

sofisticados compiladores Prolog, indicam que o emprego de um desses compiladores tornará o pré-processador Estelle/83 consideravelmente mais eficiente (por exemplo, o Prolog da IBM [IBM ..c]).

Em relação à performance do programa Prolog, propriamente dita, acredita-se que a implementação, em uma linguagem procedural, de certas funções do pré-processador Estelle/83 (por exemplo, a análise léxica), também reduzirá o tempo gasto na obtenção do código executável.

Foi deixado para o futuro a avaliação do código Pascal gerado, quando comparado com uma implementação desenvolvida manualmente.

Outro trabalho que pode ser desenvolvido, visando melhorar a produtividade do pré-processador Estelle/83, é a definição de um estratégia mais eficiente para a recuperação de erros [FiMi 80] e a consequente inclusão, ao pré-processador, dos procedimentos (em Prolog) que implementam essa estratégia.

A razão maior para o emprego de Prolog, na construção do pré-processador Estelle/83, foi facilitar a sua adaptação ao futuro padrão da TDF Estelle. Uma versão, mais recente de Estelle, é apresentada em [ISO 85]. A partir da análise, realizada em [Linn 85], dessa versão intermediária, pode-se já constatar grandes diferenças em relação a Estelle/83 (por exemplo, a possibilidade de criação dinâmica de módulos). Pode-se preconizar, então, que a adaptação do pré-processador Estelle/83 à versão padrão do Estelle não será trivial. Esse trabalho seria ainda mais oneroso, se uma linguagem procedural tivesse sido

escolhida para a construção do compilador Estelle/83.

Estudos preliminares levam a crer que o esforço dispendido no desenvolvimento de uma implementação completa, a partir da compilação de uma especificação formal, em Estelle/83, é cerca de 50% menor que o esforço dispendido no desenvolvimento de uma implementação manual, a partir da mesma especificação.

Além de fornecer implementações semi-automáticas, o compilador Estelle/83, devido às caracteristicas intrincecas da TDF Estelle/83, também fornece meios para a especificação de ambientes de simulação. Portanto, esse compilador pode ser a base para o desenvolvimento de simuladores amigáveis, que visem a validação e/ou análise de desempenho de especificações Estelle/83 de protocolos.

Referências bibliográficas

[AhUl 79] A.V. Aho, J.D. Ullmann, "Principles of Compiler Design", Addison-Wesley Publishing Company, Menlo Park (USA), abril 1979.

[AnRa 82] J.P. Ansart, O. Rafiq, V. Chari, "Protocol Description and Implemmentation Language", Anais do II IFIP WG6.1 International Workshop on Protocol Specification, Testing, and Verification, Idyllwild (EUA), maio 1982, pp. 17-20. (Também publicado em Protocol Specification, Testing and Verification, II, editado por C. Sunshine, Noth-Holland, Amsterdam (Holanda), 1983)

[BaSa 87] M. Barbeau, B. Sarikaya, "CAD-PT: A computer-aided design tool for protocol testing", Relatório técnico, Concordia University, Faculty of Engineering and Computer Science, Department of Electrical Engineering, Montréal (Canadá), 1987.

[BlTe 81] T.P. Blumer, R.L. Tenney, "A Formal Specification Technique and Implementation Method for Protocol", National Bureau of Standards, Draft Report ICST/HLNP-81-85, Washington (EUA), julho 1981. (Publicado em Computer Networks vol. 6, 1982, pp.201-217)

[Boch 75] G.v. Bochmann, "Logical verification and implementation of protocols", anais do 4th Data Communications Symposiom, Quebec (Canadá), 1975, pp. 8.15-8.20.

[Boch 78] G.v. Bochmann, "Finite State Description of Communication Protocols", Computer Networks, vol. 2, outubro 1978, pp. 361-372.

[Boch 80] G.v. Bochmann, "A General Transition Model for Protocols and Communication Services", IEEE Transactions on Communications, vol. COM-28(4), abril 1980, pp. 643-650.

[Boch 83] G.v. Bochmann, "Concepts of Distributed Systems Design", Springer-Verlag, Berlin (Alemanha), 1983.

[Boch 86] G.v. Bochmann, "Recent Developments in Protocol Specification, Validation and Testing", Anais do IV Simpósio Brasileiro de Redes de Computadores, Recife (PE), março 1986, pp.354-368.

[Boch 87a] G.v. Bochmann, "Specifications of a Simplified Transport Protocol Using Differents Formal Description Techniques", Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal (Canadá), abril 1987. ("Draft")

[Boch 87b] G.v. Bochmann, "Usage of Protocol Development Tools: The Results of a Survey", IFIP Protocol Specification,

Testing and Verification: VII - An International Symposium, Participant's Proceedings, Zurique (Suiça), maio 1987.

[BoDs 85] G.v. Bochmann, R.Dssouli, W. Lopes de Souza, B. Sarikaya, H. Ural, "Use of Prolog for Building Protocol Design Tools", anais do V IFIP WG6.1 International Workshop on Protocol Specification, Verification, and Testing, Toulouse-Moissac (França), junho 1985, pp. 131-147. (Também publicado em Protocol Specification, Testing, and Verification, V, editado por M. Diaz, North Holland, Amterdam (Holanda), 1986)

[BoGe 77] G.v. Bochmann, J. Gecsei, "A Unified Model for the Specification and Verification of Protocols", Anais do IFIP Congress, Toronto (Canadá), 1977, pp. 229-234.

[BoGe 84] G.v. Bochmann, G. Gerber, J.M. Serre, "Semi-automatic Implemmentation of Communication Protocols", Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal (Canadá), publicação n. 518, Dezembro 1984.

[BoSu 80] G.v. Bochmann, C.A. Sunshine, "Formal Methods in Communication Protocol Design", IEEE Transactions on Communications, vol. COM-28(4), abril 1980, pp. 624-631.

[CaCu 87] J.F.B. de Castro, P.R.F. Cunha, "Especificando Protocolos Através de uma Metodologia Baseada em Redes de Petri", Anais do V Simpósio Brasileiro de Redes de Computadores, São Paulo (SP), abril 1987, pp. 56-82.

[CCIT 77] CCITT Recommendation X.25, "Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment for Terminals Operating in the Packet Node on Public Data Networks", Orange Book, vol. VIII.2, Genebra (Suiça), 1977, pp. 70-108.

[CCIT 83a] CCITT SDL Z.100, "Introduction to SDL", Genebra (Suiça), maio 1983.

[CCIT 83b] CCITT SDL Z.101, "Basic SDL", Genebra (Suiça), maio 1983.

[CCIT 83c] CCITT SDL Z.102, "Structural Concepts in SDL", Genebra (Suiça), maio 1983.

[CCIT 83d] CCITT SDL Z.103, "Functional Extensions to SDL", Genebra (Suiça), maio 1983.

[CCIT 83e] CCITT SDL Z.104, "Data in SDL", Genebra (Suiça), maio 1983.

[CCIT 84] CCITT Recommendation X.200, "Reference Model of Open Systems Interconnection for CCITT Applications", Genebra (Suiça), outubro 1984.

[Clar 84] K.L. Clark, "micro-PROLOG: Programming in Logic", Prentice-Hall International, 1984.

[ClMe 81] W.F. Clocksin, C.S. Mellish, "Programming in Prolog", Springer-Verlag, Berlin (Alemanha), 1981.

[Dant 80] A.A.S. Danthine, "Protocol Representation With Finite State Models", IEEE Transactions on Communications, vol. COM-28(4), abril 1980, pp. 632,642.

[DsBo 86] R. Dssouli, G.v. Bochmann, "Conformance testing with multiple observers", anais do VI IFIP WG6.1 International Workshop on Protocol Specification, Verification, and Testing, Montréal (Canadá), junho 1986, pp. 6.35-6.55.

[EhMa 85] H. Ehrig, B. Mahr, "Fundamentals of Algebric Specification 1 - Equations and Initial Semantics", Springer-Verlag, Berlin (Alemanha), 1985.

[FaLi 86] J.-P. Favreau, R.J. Linn Jr, "Automatic generation of test scenario skeletons from protocol specifications written in Estelle", anais do VI IFIP WG6.1 International Workshop on Protocol Specification, Verification, and Testing, Montréal (Canadá), junho 1986, pp. 6.1-6.12.

[FeLo 86] E. Ferneda, W. Lopes de Souza, "Compilador para a Linguagem de Especificação de Protocolos Estelle", Anais do XIII Seminário Integrado de Software e Hardware do VI Congresso da Sociedade Brasileira de Computação, Recife (PE), julho 1986, Vol. 1, pp. 420-428.

[FiMi 80] C.N. Fisher, D.R. Milton, S.B. Quiring, "Eficient LL(1) Error Correction and Recovery Using Only Insertions", Acta Informática, vol. 13, 1980, pp. 141-154.

[Gerb 83] G.W. Gerber, "Une Methode d'Implantation Automatisée de Systemes Specifies Formellement", Dissertação de mestrado, Département d'Informatique et de Recherche Opérationelle, Université de Montréal, Montréal (Canadá), doc. de travail #142, outubro 1983.

[Grie 71] D. Gries, "Compiler Construction for Digital Computers", John Wiley & Sons, Inc., New York (EUA), 1971.

[Groz 86] R. Groz, "Unrestricted verification of protocol properties on a simulation using an observer approach", anais do VI IFIP WG6.1 International Workshop on Protocol Specification, Verification, and Testing, Montréal (Canadá), junho 1986, pp. 7.25-7.36.

[IBM ..a] "Pascal/VS Language Reference Manual", IBM, Manual n. SH20-6168-1.

[IBM ..b] "Pascal/VS Programmer's Guide", IBM, Manual n. SH20-6162-2.

[IBM ..c] "MZS/Program Logic - Program Description Operation Manual", IBM, Manual n. SH40-0030.

[ISO 82] ISO TC97/SC16/WG1 subgroup A, "Concepts for Describing the OSI Architecture", Working Document N1346, novembro 1982.

[ISO 83a] ISO IS 7498, "Information Processing Systems - Basic Reference Model for Open Systems Interconnections", 1983.

[ISO 83b] ISO DP 8326, "Open Systems Interconnection - Basic Connection Oriented Section - Service Definition", março 1983.

[ISO 83c] ISO DP 8327, "Open Systems Interconnection - Basic Connection Oriented Section - Protocol Specification", março 1983.

[ISO 83d] ISO IS 7185, "Programming Language Pascal", 1a. edição, dezembro 1983.

[ISO 83e] ISO TC97/SC16/WG1 subgroup B, "A FDT Based on an Extended State Transition Model", Working Document, maio 1983.

[ISO 84a] ISO DIS 8072, "Informations Processing Systems - Open Systems Interconnection - Transport Service Definition", maio 1984.

[ISO 84b] ISO DIS 8073, "Informations Processing Systems - Open Systems Interconnection - Connection Oriented Transport Protocol Specification", maio 1984.

[ISO 84c] ISO DP 8822, "Informations Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Service Definition", outubro 1984.

[ISO 84d] ISO DP 8822, "Informations Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Protocol Definition", outubro 1984.

[ISO 85] ISO DP 9074, "Information Process Systems - Open Systems Interconnection - Estelle - A Formal Description Technique Based on an Extended State Transition Model", setembro 1985.

[ISO 86a] ISO DIS 8571/1/2/3/4, "Information Processing Systems - Open Systems Interconnection - File Transfer, Access and Management, agosto 1986.

[ISO 86b] ISO DP 8807, "Lotos - A formal Description Technique Based on the Temporal Ordering of Observational Behavior", março 1986.

[IsRo 85] M.M. Isoni, J.A. Rodrigues, J.P.Sauvé, J.A.B. Moura, "Uma Experiência em Implementação e Testes de Protocolos para Redes de Computadores", Anais do III Simpósio Brasileiro de Redes de Computadores, Rio de Janeiro (RJ), abril 1985, pp. 12.1-12.16.

[JaMo 85] C. Jard, J.F. Monin, R. Groz, "Experience in Implementing X.250 (a CCITT subset of Estelle) in Veda", anais do V IFIP WG6.1 Workshop on Protocol Specification, Verification, and Testing, Toulouse-Moinsac (França), junho 1985, pp. 315-331. (Também publicado em Protocol Specification, Testing, and Verification, V, editado por M. Diaz, North-Holland, Amterdam (Holanda), 1986)

[Lamp 80] L. Lamport, "'Sometime' Is Sometimes Not Never: On the Temporal Logic Program", Anais do VII annual ACM Symposium on Principles of Programming Languages, Las Vegas (EUA), 1980, pp. 174-185.

[Linn 85] R.J. Linn Jr., "The Features and Facilities of Estelle", anais do V IFIP WG6.1 International Workshop on Protocol Specification, Verification, and Testing, Toulouse-Moinsac (França), junho 1985, pp. 271-296. (Também publicado em Protocol Specification, Testing, and Verification, V, editado por M. Diaz, North-Holland, Amterdam (Holanda), 1986)

[LoDs 85] W. Lopes de Souza, R.Dssouli, G.v.Bochmann, "Ambiente de Teste para Protocolos de Comunicação", Anais do III Simpósio Brasileiro de Redes de Computadores, Rio de Janeiro (RJ), março 1985, pp. 24.1-24.12.

[LoFe 86] W. Lopes de Souza, E. Ferneda, "Analisador Sintático em Prolog para a Linguagem de Especificação de Protocolos Estelle", Anais do IV Simpósio Brasileiro de Redes de Computadores, Recife (PE), março 1986, pp. 325-353.

[LoFe 87a] W. Lopes de Souza, E. Ferneda, "Aplicações do Compilador Estelle", Anais do V Simpósio Brasileiro de Redes de Computadores, São Paulo (SP), abril 1987, pp. 107-120.

[LoFe 87b] W. Lopes de Souza, E. Ferneda, "A Compiler for a Formal Description Technique", Anais do I Iberican Conference on Data Communications, Lisboa (Portugal), maio 1987, pp. 275-286. (Também publicado em Computer Communications Systems, editado por A. Cerveira, North-Holland, Amterdam (Holanda), janeiro 1988)

[LoFr 87] W. Lopes de Souza, J.U. Ferreira Jr., "Especificações Informais e Formais do Serviço e Protocolo de Transporte - Classe 2", Iteel, Relatório Técnico contrato Telebrás/Iteel n. 190/87, Campina Grande (PB), 1987.

[Lope 85] W. Lopes de Souza, "Utilização dos Conceitos de Módulo, Porta e Canal em Especificações Formais de Serviços, Protocolos e Interfaces de Comunicação", Anais do III Simpósio Brasileiro de Redes de Computadores, Rio de Janeiro (RJ), abril 1985, pp. 25.1-25.23.

[Lope 87a] W. Lopes de Souza, "Uma Proposta para o Desenvolvimento Sistemático do Software de Comunicação para Rede Local", Anais do V Simpósio Brasileiro de Telecomunicações, UNICAMP, Campinas, São Paulo (SP), setembro 1987, pp. 92-99.

[Lope 87b] W. Lopes de Souza, "LOTOS: Uma técnica para a Descrição Formal de Serviços e Protocolos", Anais do V Simpósio Brasileiro de Redes de Computadores, São Paulo (SP), abril 1987, pp. 121-144.

[LoSt 88] W. Lopes de Souza, S. Stiubiener, "Especificação, verificação e testes de protocolos", Relatório Técnico GRC/UFBb n. TR-01/88, fevereiro 1988. (Também submetido à Revista Brasileira de Telecomunicações)

[Miln 80] R. Milner, "A Calculus of Communicating Systems", editado por G.Goos e J. Hartmanis, Springer-Verlag, Berlin (Alemanha), 1980.

[Moni 84] J.F. Monin, "Ecriture d'un Compilateur 'reel' en Prolog", Centre National d'Etudes de Télécommunications, Note Technique NT/LAA/SLC/179, Lannion (França), julho 1984. (Também publicado em Journées sur la Programmation en Logique, Plestin (França), abril 1984)

[Mont 82] J.A.S. Monteiro, "Descrição, Validação e Geração Automática de Implementações de Protocolos", Dissertação de mestrado, Escola Politécnica, Universidade de São Paulo, São Paulo (SP), 1982.

[Sari 87] B. Sarikaya, "Recent Developments in Protocol Testing", Anais do V Simpósio Brasileiro de Redes de Computadores, São Paulo (SP), abril 1987, pp. 372-392.

[ScRo 80] G.D. Schultz, D.B. Rose, C.H. West, J.P. Gray, "Executable description and validation of SNA", IEEE Transactions on Communications, vol. COM-28(4), abril 1980, pp. 661-667.

[SeMe 83] V.W. Setzer, I.S.H. de Melo, "A Construção de um Compilador", Editora Campus, Rio de Janeiro (RJ), 1983.

[StSh 86] L. Sterling, E. Shapiro, "The Art of Prolog", The MIT Press, Cambridge (USA), julho 1986.

[SuTh 81] C.A. Sunshine, D.H. Thompson, R.W. Erickson, S.L. Gerhart, D. Schwabe, "Specification and verification of communication protocols in AFFIRM using state

transition models", Information Sciences Institute, University of Southern California, Relatório técnico RR-81-88, 1981. (Também publicado em IEEE Transactions on Software Engineering, vol. SE-8(9), setembro 1982)

[Tane 81] A.S. Tanenbaum, "Computer Networks", Prentice-Hall Inc., New Jersey (EUA), 1981.

[TeLi 78] A.Y. Teng, M.T. Liu, "A Formal Model for Automatic Implemmentation and Logical Validation of Networks Communications Protocols", Anais do Computer Networking Symposium, National Bureau of Standard, dezembro 1978, pp. 114-123.

[VaBo 84] J. Vaucher, G.v. Bochmann, "A simulation tool for formal specifications", relatório técnico de Cerbo Informatique Inc., Montréal (Canadá), julho 1984.

[Warr 80] D.M.D. Warren, "Logic Programming and Compiler Writting", Software - Pratice and Experience, vol. 10, 1980, pp. 97-125.

[Wate ..] "Waterloo Prolog User's Manual", version 1.7, Intralogic, Inc., Waterloo (Canadá).

[Zimm 80] H. Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection", IEEE Transactions on Communications, vol. COM-28(4), abril 1980, pp.425-432.

- A N E X O A -

Sintaxe da TDF Estelle/83

```
⟨axiom⟩ ::= ⟨seqsect⟩.

⟨seqsect⟩ ::= ⟨section⟩ ";" ⟨seqsect⟩
            | empty.

⟨section⟩ ::= ⟨module⟩
            | ⟨process⟩
            | ⟨refinemt⟩
            | ⟨channel⟩.

⟨channel⟩ ::= ⟨constd⟩ ⟨typed⟩ "channel" ⟨ident⟩
              "(" ⟨rolelist⟩ ")" ";" ⟨byclause⟩ "end" ⟨ident⟩.

⟨rolelist⟩ ::= ⟨ident⟩ ⟨seqident⟩.

⟨seqident⟩ ::= "," ⟨rolelist⟩
             | empty.

⟨byclause⟩ ::= "by" ⟨rolelist⟩ ":" ⟨signal⟩ ⟨byclause⟩
             | empty.

⟨signal⟩ ::= ⟨ident⟩ ⟨signalpara⟩ ";" ⟨signal⟩
           | empty.

⟨signalpara⟩ ::= "(" ⟨paradef⟩ ")"
               | empty.

⟨seqparadef⟩ ::= ";" ⟨paradef⟩
               | empty.

⟨paradef⟩ ::= ⟨rolelist⟩ ":" ⟨ident⟩ ⟨seqparadef⟩.

⟨module⟩ ::= "module" ⟨ident⟩ ";" ⟨portlist⟩ "end" ⟨ident⟩.

⟨portlist⟩ ::= ⟨oprolelist⟩ ":" ⟨array⟩ ⟨ident⟩ "(" ⟨ident⟩ ")"
               ";" ⟨portlist⟩
             | empty.

⟨array⟩ ::= "array" "(." ⟨indextype⟩ ⟨seqindext⟩ ".)" "of"
          | empty.

⟨indextype⟩ ::= ⟨simpletype⟩.

⟨seqindext⟩ ::= "," ⟨indextype⟩ ⟨seqindext⟩
              | empty.

⟨refinemt⟩ ::= "refinement" ⟨ident⟩ ⟨signalpara⟩ "for" ⟨ident⟩
               ";" ⟨refbody⟩ "end" ⟨ident⟩.

⟨refbody⟩ ::= ⟨seqsect⟩ ⟨instance⟩ ⟨connect⟩ ⟨replace⟩
            | empty.
```

```
⟨instance⟩ ::= ⟨oprolelist⟩ ":" ⟨ident⟩ "with" ⟨ident⟩
               ⟨actualpar⟩ ";" ⟨seqinst⟩.

⟨seqinst⟩ ::= ⟨instance⟩
            | empty.

⟨connect⟩ ::= "connect" ⟨intconn⟩
            | empty.

⟨intconn⟩ ::= ⟨mport⟩ "to" ⟨mport⟩ ";" ⟨seqintconn⟩.

⟨seqintconn⟩ ::= ⟨intconn⟩
               | empty.

⟨replace⟩ ::= "replace" ⟨extconn⟩
            | empty.

⟨extconn⟩ ::= ⟨port⟩ "by" ⟨mport⟩ ";" ⟨seqintconn⟩.

⟨seqextconn⟩ ::= ⟨extconn⟩
               | empty.

⟨port⟩ ::= ⟨ident⟩ ⟨optindex⟩.

⟨optindex⟩ ::= "(." ⟨constant⟩ ⟨listconst⟩ ".)"
             | empty.

⟨mport⟩ ::= ⟨ident⟩ "." ⟨port⟩.

⟨process⟩ ::= "process" ⟨ident⟩ ⟨signalpara⟩ "for" ⟨ident⟩ ";"
              ⟨procbody⟩ "end" ⟨ident⟩.

⟨qchannel⟩ ::= "queued" ⟨rolelist⟩ ";"
             | empty.

⟨procbody⟩ ::= ⟨qchannel⟩ ⟨constd⟩ ⟨typed⟩ ⟨pvard⟩ ⟨procfuncd⟩
               ⟨init⟩ ⟨trans⟩
             | empty.

⟨pvard⟩ ::= "var" ⟨procvar⟩
          | empty.

⟨procvar⟩ ::= "state" ":" "(" ⟨rolelist⟩ ")" ";" ⟨seqvardecl⟩
            | ⟨vardecl⟩.

⟨stateset⟩ ::= ⟨ident⟩ "=" "(." ⟨seqsetint⟩ ".)" ";" ⟨stateset⟩
             | empty.

⟨init⟩ ::= "initialize" ⟨stateset⟩
           "begin" ⟨initstatmt⟩ ⟨seqstatmt⟩ "end" ";"
         | empty.

⟨initstatmt⟩ ::= "state" ":=" ⟨ident⟩
               | ⟨plainstatmt⟩.

⟨trans⟩ ::= "trans" ⟨seqclause⟩ ⟨opttrans⟩.

⟨opttrans⟩ ::= ⟨trans⟩
```

```
                | empty.

⟨seqclause⟩ ::= ⟨clause⟩ ⟨seqclause⟩
                | ⟨opttag⟩ ⟨blocktrans⟩ ";" ⟨seqtrans⟩.

⟨blocktrans⟩ ::= ⟨labeld⟩ ⟨constd⟩ ⟨vard⟩ ⟨procfuncd⟩
                 | "begin" ⟨statmt⟩ ⟨seqstatmt⟩ "end".

⟨clause⟩ ::= "any" ⟨paradef⟩ "do"
             | "with" ⟨variable⟩ "do"
             | "when" ⟨ident⟩ ⟨vparam⟩ "." ⟨ident⟩
             | "from" ⟨rolelist⟩
             | "to" ⟨nextmstate⟩
             | "save" ⟨ident⟩ ⟨vparam⟩ "." ⟨ident⟩ ⟨actualpar⟩
             | "provided" ⟨expression⟩
             | "priority" ⟨idorint⟩.

⟨seqtrans⟩ ::= ⟨seqclause⟩
               | empty.

⟨opttag⟩ ::= ⟨ident⟩ ":"
             | empty.

⟨vparam⟩ ::= "(." ⟨rolelist⟩ ".)"
             | empty.

⟨listvariable⟩ ::= "," ⟨variable⟩
                   | empty.

⟨nextmstate⟩ ::= ⟨rolelist⟩
                 | "same".

⟨idorint⟩ ::= ⟨ident⟩
              | ⟨integer⟩.

⟨block⟩ ::= ⟨labeld⟩ ⟨constd⟩ ⟨typed⟩ ⟨vard⟩ ⟨procfuncd⟩
            "begin" ⟨statmt⟩ ⟨seqstatmt⟩ "end".

⟨labeld⟩ ::= "label" ⟨integer⟩ ⟨seqinteger⟩ ";"
             | empty.

⟨seqinteger⟩ ::= "," ⟨integer⟩ ⟨seqinteger⟩
                 | empty.

⟨constd⟩ ::= "const" ⟨defconst⟩
             | empty.

⟨defconst⟩ ::= ⟨ident⟩ "=" ⟨constant⟩ ";" ⟨seqdefconst⟩.

⟨seqdefconst⟩ ::= ⟨defconst⟩
                  | empty.

⟨constant⟩ ::= ⟨optsign⟩ ⟨numconst⟩
               | ⟨string⟩
               | "...".

⟨sign⟩ ::= "+"
         | "-".
```

```
⟨optsign⟩ ::= ⟨sign⟩
            | empty.

⟨numconst⟩ ::= ⟨integer⟩
             | ⟨real⟩
             | ⟨ident⟩.

⟨typed⟩ ::= "type" ⟨deftype⟩
          | empty.

⟨deftype⟩ ::= ⟨ident⟩ "=" ⟨type⟩ ";" ⟨seqdeftype⟩.

⟨seqdeftype⟩ ::= ⟨deftype⟩
               | empty.

⟨type⟩ ::= ⟨optpack⟩ ⟨typstruct⟩
         | ⟨simpletype⟩
         | "->" ⟨ident⟩.

⟨simpletype⟩ ::= "(" ⟨rolelist⟩ ")"
               | ⟨sign⟩ ⟨numconst⟩ ".." ⟨constant⟩
               | ⟨string⟩ ".." ⟨constant⟩
               | ⟨integer⟩ ".." ⟨constant⟩
               | ⟨ident⟩ ⟨optconst⟩.

⟨optconst⟩ ::= ".." ⟨constant⟩
             | empty.

⟨optpack⟩ ::= "packed"
            | empty.

⟨typstruct⟩ ::= "array" "(." ⟨simpletype⟩ ⟨seqsimplet⟩ ".)" "of"
                ⟨type⟩
              | "record" ⟨field⟩ "end"
              | "set" "of" ⟨simpletype⟩
              | "file" "of" ⟨type⟩.

⟨seqsimplet⟩ ::= "," ⟨simpletype⟩ ⟨seqsimplet⟩
               | empty.

⟨field⟩ ::= "case" ⟨ident⟩ ⟨typselect⟩ "of" ⟨variant⟩
          | ⟨fixedpart⟩ ⟨seqfield⟩.

⟨fixedpart⟩ ::= ⟨oprolelist⟩ ":" ⟨type⟩
              | empty.

⟨seqfield⟩ ::= ";" ⟨field⟩
             | empty.

⟨typselect⟩ ::= ":" ⟨ident⟩
              | empty.

⟨variant⟩ ::= ⟨optconstant⟩ ⟨listconst⟩ ":" "(" ⟨field⟩ ")"
              ⟨seqvariant⟩
            | empty.

⟨seqvariant⟩ ::= ";" ⟨variant⟩
               | empty.
```

```
⟨listconst⟩ ::= "," ⟨constant⟩ ⟨listconst⟩
              | empty.

⟨vard⟩ ::= "var" ⟨vardecl⟩
         | empty.

⟨vardecl⟩ ::= ⟨oprolelist⟩ ":" ⟨type⟩ ";" ⟨seqvardecl⟩.

⟨seqvardecl⟩ ::= ⟨vardecl⟩
               | empty.

⟨procfuncd⟩ ::= ⟨pfheader⟩ ";" ⟨pfbody⟩ ";" ⟨procfund⟩
              | empty.

⟨pfheader⟩ ::= "procedure" ⟨ident⟩ ⟨lpara⟩
             | "predicate" ⟨ident⟩ ⟨lpara⟩
             | "function" ⟨ident⟩ ⟨lpara⟩ ":" ⟨ident⟩.

⟨pfbody⟩ ::= "extern"
           | "forward"
           | "primitive"
           | "..."
           | ⟨block⟩.

⟨lpara⟩ ::= "(" ⟨spara⟩ ⟨seqspara⟩ ")"
          | empty.

⟨seqspara⟩ ::= ":" ⟨spara⟩ ⟨seqspara⟩
             | empty.

⟨spara⟩ ::= ⟨oprolelist⟩ ":" ⟨ident⟩
          | "var" ⟨rolelist⟩ ":" ⟨ident⟩
          | "procedure" ⟨ident⟩ ⟨lpara⟩
          | "function" ⟨ident⟩ ⟨lpara⟩ ":" ⟨ident⟩.

⟨factor⟩ ::= ⟨integer⟩
           | ⟨real⟩
           | ⟨string⟩
           | "nil"
           | "..."
           | ⟨ident⟩ ⟨seqfactid⟩
           | "not" ⟨factor⟩
           | "(." ⟨seqsetint⟩ ".)"
           | "(" ⟨expression⟩ ")".

⟨seqfactid⟩ ::= "(" ⟨index⟩ ")"
              | ⟨lseqvaria⟩.

⟨index⟩ ::= ⟨expression⟩ ⟨seqindex⟩.

⟨seqindex⟩ ::= "," ⟨index⟩
             | empty.

⟨lseqvaria⟩ ::= "(." ⟨index⟩ ".)" ⟨lseqvaria⟩
              | "." ⟨ident⟩ ⟨lseqvaria⟩
              | "-)" ⟨lseqvaria⟩
              | empty.
```

```
⟨seqsetint⟩ ::= ⟨setint⟩ ⟨lseqset⟩
              | empty.

⟨lseqset⟩ ::= "," ⟨setint⟩ ⟨lseqset⟩
            | empty.

⟨setint⟩ ::= ⟨expression⟩ ⟨seqxpset⟩.

⟨seqxpset⟩ ::= ".." ⟨expression⟩
             | empty.

⟨term⟩ ::= ⟨factor⟩ ⟨seqfact⟩.

⟨seqfact⟩ ::= ⟨opermult⟩ ⟨term⟩
            | empty.

⟨opermult⟩ ::= "*"
             | "/"
             | "div"
             | "mod"
             | "and".

⟨simplexp⟩ ::= ⟨optsign⟩ ⟨term⟩ ⟨seqterm⟩.

⟨seqterm⟩ ::= ⟨operadd⟩ ⟨term⟩ ⟨seqterm⟩
            | empty.

⟨operadd⟩ ::= "+"
            | "-"
            | "or".

⟨expression⟩ ::= ⟨simplexp⟩ ⟨seqsimplexp⟩.

⟨seqsimplexp⟩ ::= ⟨operel⟩ ⟨simplexp⟩
                | empty.

⟨operel⟩ ::= "="
           | "⟨⟩"
           | "⟨"
           | "⟩"
           | "⟨="
           | "⟩="
           | "in".

⟨statmt⟩ ::= ⟨integer⟩ ":" ⟨plainstatmt⟩
           | ⟨plainstatmt⟩.

⟨plainstatmt⟩ ::= ⟨ident⟩ ⟨appendix⟩
                | "nextstate" ⟨newmstate⟩
                | "out" ⟨ident⟩ ⟨seqindice⟩ "." ⟨ident⟩
                  ⟨actualpar⟩
                | "goto" ⟨integer⟩
                | "begin" ⟨statmt⟩ ⟨seqstatmt⟩ "end"
                | "if" ⟨expression⟩ "then" ⟨statmt⟩ ⟨else⟩
                | "case" ⟨expression⟩ "of" ⟨case⟩ ⟨seqcase⟩
                  "end"
                | "repeat" ⟨statmt⟩ ⟨seqstatmt⟩ "until"
                  ⟨expression⟩
```

```
                  | "while" ⟨expression⟩ "do" ⟨statmt⟩
                  | "for" ⟨ident⟩ ":=" ⟨expression⟩ ⟨direction⟩
                    ⟨expression⟩ "do" ⟨statmt⟩
                  | "with" ⟨variable⟩ "do" ⟨statmt⟩
                  | empty.

⟨actualpar⟩ ::= "(" ⟨index⟩ ")"
                | empty.

⟨newmstate⟩ ::= ⟨ident⟩
                | "same".

⟨seqstatmt⟩ ::= ";" ⟨statmt⟩ ⟨seqstatmt⟩
                | empty.

⟨else⟩ ::= "else" ⟨statmt⟩
         | empty.

⟨seqcase⟩ ::= ":" ⟨case⟩ ⟨seqcase⟩
            | empty.

⟨case⟩ ::= ⟨optconstant⟩ ⟨listconst⟩ ":" ⟨statmt⟩
         | empty.

⟨direction⟩ ::= "to"
                | "downto".

⟨variable⟩ ::= ⟨ident⟩ ⟨lseqvaria⟩ ⟨listvariable⟩.

⟨appendix⟩ ::= ⟨lseqvaria⟩ ":=" ⟨expression⟩
             | ⟨actualpar⟩.

⟨seqindice⟩ ::= "(." ⟨index⟩ ".)" ⟨seqindice⟩
                | empty.

⟨oprolelist⟩ ::= ⟨ident⟩ ⟨seqident⟩.

⟨optconstant⟩ ::= ⟨optsign⟩ ⟨numconst⟩
                  | ⟨string⟩.
```

- A N E X O B -

Subconjunto da sintaxe da linguagem Pascal/VS utilizado

```
⟨program⟩ ::= ⟨proghead⟩ ⟨segblock⟩.

⟨proghead⟩ ::= "segment" ⟨ident⟩ ";".

⟨segblock⟩ ::= ⟨labeld⟩ ⟨constd⟩ ⟨typed⟩ ⟨vard⟩ ⟨procfuncd⟩.

⟨labeld⟩ ::= "label" ⟨integer⟩ ⟨seqinteger⟩ ";"
           | empty.

⟨seqinteger⟩ ::= "," ⟨integer⟩ ⟨seqinteger⟩
               | empty.

⟨constd⟩ ::= "const" ⟨defconst⟩
           | empty.

⟨defconst⟩ ::= ⟨ident⟩ "=" ⟨constant⟩ ";" ⟨seqdefconst⟩.

⟨seqdefconst⟩ ::= ⟨defconst⟩
                | empty.

⟨constant⟩ ::= ⟨optsign⟩ ⟨numconst⟩
             | ⟨string⟩.

⟨sign⟩ ::= "+"
         | "-".

⟨optsign⟩ ::= ⟨sign⟩
            | empty.

⟨numconst⟩ ::= ⟨integer⟩
             | ⟨real⟩
             | ⟨ident⟩.

⟨typed⟩ ::= "type" ⟨deftype⟩
          | empty.

⟨deftype⟩ ::= ⟨ident⟩ "=" ⟨type⟩ ";" ⟨seqdeftype⟩.

⟨seqdeftype⟩ ::= ⟨deftype⟩
               | empty.

⟨type⟩ ::= ⟨optpack⟩ ⟨typstruct⟩
         | ⟨simpletype⟩
         | "->" ⟨ident⟩.

⟨simpletype⟩ ::= "(" ⟨rolelist⟩ ")"
               | ⟨sign⟩ ⟨numconst⟩ ".." ⟨constant⟩
               | ⟨string⟩ ".." ⟨constant⟩
               | ⟨integer⟩ ".." ⟨constant⟩
               | ⟨ident⟩ ⟨optconst⟩.

⟨optconst⟩ ::= ".." ⟨constant⟩
```

```
                | empty.

<optpack> ::= "packed"
            | empty.

<typstruct> ::= "array" "(." <simpletype> <seqsimplet> ".)" "of"
                <type>
              | "record" <field> "end"
              | "set" "of" <simpletype>
              | "file" "of" <type>.

<seqsimplet> ::= "," <simpletype> <seqsimplet>
               | empty.

<field> ::= "case" <ident> <typselect> "of" <variant>
          | <fixedpart> <seqfield>.

<fixedpart> ::= <rolelist> ":" <type>
              | empty.

<seqfield> ::= ";" <field>
             | empty.

<typselect> ::= ":" <ident>
              | empty.

<variant> ::= <constant> <listconst> ":" "(" <field> ")"
              <seqvariant>
            | empty.

<seqvariant> ::= ";" <variant>
               | empty.

<listconst> ::= "," <constant> <listconst>
              | empty.

<vard> ::= "var" <vardecl>
         | empty.

<vardecl> ::= <rolelist> ":" <type> ";" <seqvardecl>.

<seqvardecl> ::= <vardecl>
               | empty.

<procfuncd> ::= <pfheader> ";" <pfbody> ";" <procfund>
              | empty.

<pfheader> ::= "procedure" <ident> <lpara>
             | "function" <ident> <lpara> <seqfunc>.

<seqfunc> ::= ":" <ident>
            | empty.

<pfbody> ::= "external"
           | "forward"
           | <block>.

<block> ::= <labeld> <constd> <typed> <vard> <procfuncd>
```

```
            "begin" ⟨statmt⟩ ⟨seqstatmt⟩ "end".

⟨lpara⟩ ::= "(" ⟨spara⟩ ⟨seqspara⟩ ")"
          | empty.

⟨seqspara⟩ ::= ":" ⟨spara⟩ ⟨seqspara⟩
             | empty.

⟨spara⟩ ::= ⟨rolelist⟩ ":" ⟨ident⟩
          | "var" ⟨rolelist⟩ ":" ⟨ident⟩
          | "procedure" ⟨ident⟩ ⟨lpara⟩
          | "function" ⟨ident⟩ ⟨lpara⟩ ":" ⟨ident⟩.

⟨factor⟩ ::= ⟨integer⟩
           | ⟨real⟩
           | ⟨string⟩
           | "nil"
           | ⟨ident⟩ ⟨seqfactid⟩
           | "not" ⟨factor⟩
           | "(." ⟨seqsetint⟩ ".)"
           | "(" ⟨expression⟩ ")".

⟨seqfactid⟩ ::= "(" ⟨index⟩ ")"
              | ⟨lseqvaria⟩.

⟨index⟩ ::= ⟨expression⟩ ⟨seqindex⟩.

⟨seqindex⟩ ::= "," ⟨index⟩
             | empty.

⟨lseqvaria⟩ ::= "(." ⟨index⟩ ".)" ⟨lseqvaria⟩
              | "." ⟨ident⟩ ⟨lseqvaria⟩
              | "-)" ⟨lseqvaria⟩
              | empty.

⟨seqsetint⟩ ::= ⟨setint⟩ ⟨lseqset⟩
              | empty.

⟨lseqset⟩ ::= "," ⟨setint⟩ ⟨lseqset⟩
            | empty.

⟨setint⟩ ::= ⟨expression⟩ ⟨seqxpset⟩.

⟨seqxpset⟩ ::= ".." ⟨expression⟩
             | empty.

⟨term⟩ ::= ⟨factor⟩ ⟨seqfact⟩.

⟨seqfact⟩ ::= ⟨opermult⟩ ⟨term⟩
            | empty.

⟨opermult⟩ ::= "*"
             | "/"
             | "div"
             | "mod"
             | "and".

⟨simplexp⟩ ::= ⟨optsign⟩ ⟨term⟩ ⟨seqterm⟩.
```

```
⟨seqterm⟩ ::= ⟨operadd⟩ ⟨term⟩ ⟨seqterm⟩
            | empty.

⟨operadd⟩ ::= "+"
            | "-"
            | "or".

⟨expression⟩ ::= ⟨simplexp⟩ ⟨seqsimplexp⟩.

⟨seqsimplexp⟩ ::= ⟨operel⟩ ⟨simplexp⟩
                | empty.

⟨operel⟩ ::= "="
           | "⟨⟩"
           | "⟨"
           | "⟩"
           | "⟨="
           | "⟩="
           | "in".

⟨statmt⟩ ::= ⟨integer⟩ ":" ⟨plainstatmt⟩
           | ⟨plainstatmt⟩.

⟨plainstatmt⟩ ::= ⟨ident⟩ ⟨appendix⟩
                | "goto" ⟨integer⟩
                | "begin" ⟨statmt⟩ ⟨seqstatmt⟩ "end"
                | "if" ⟨expression⟩ "then" ⟨statmt⟩ ⟨else⟩
                | "case" ⟨expression⟩ "of" ⟨case⟩ ⟨seqcase⟩
                  "end"
                | "repeat" ⟨statmt⟩ ⟨seqstatmt⟩ "until"
                  ⟨expression⟩
                | "while" ⟨expression⟩ "do" ⟨statmt⟩
                | "for" ⟨ident⟩ ":=" ⟨expression⟩ ⟨direction⟩
                  ⟨expression⟩ "do" ⟨statmt⟩
                | "with" ⟨variable⟩ "do" ⟨statmt⟩
                | empty.

⟨actualpar⟩ ::= "(" ⟨index⟩ ")"
              | empty.

⟨seqstatmt⟩ ::= ";" ⟨statmt⟩ ⟨seqstatmt⟩
              | empty.

⟨else⟩ ::= "else" ⟨statmt⟩
         | empty.

⟨seqcase⟩ ::= ":" ⟨case⟩ ⟨seqcase⟩
            | empty.

⟨case⟩ ::= ⟨constant⟩ ⟨listconst⟩ ":" ⟨statmt⟩
         | empty.

⟨direction⟩ ::= "to"
              | "downto".

⟨variable⟩ ::= ⟨ident⟩ ⟨lseqvaria⟩ ⟨listvariable⟩.
⟨appendix⟩ ::= ⟨lseqvaria⟩ ":=" ⟨expression⟩
```

```
                | ⟨actualpar⟩.

⟨seqindice⟩ ::= "(." ⟨index⟩ ".)" ⟨seqindice⟩
                | empty.

⟨rolelist⟩ ::= ⟨ident⟩ ⟨seqident⟩.

⟨seqident⟩ ::= "," ⟨rolelist⟩
               | empty.

⟨listvariable⟩ ::= "," ⟨variable⟩
                   | empty.
```

- A N E X O C -

Especificação de um protocolo em Estelle/83

```
module SIDE;
end SIDE;

refinement SIDE_REFINEMENT for SIDE;

  const MAXLIN = 80;

  type DATA_TYPE = record
                     IND  : 0..MAXLIN;
                     LINE : array(.1..MAXLIN.) of char
                   end;

  channel TERMINAL_ACCESS_POINT(all);
    by all : DATA(D : DATA_TYPE);
  end TERMINAL_ACCESS_POINT;

  module TERMINAL;
    PORT : TERMINAL_ACCESS_POINT(all);
  end TERMINAL;

  process EX_TERMINAL for TERMINAL;

    var CH        : char;
        X         : DATA_TYPE;

    procedure GETCH(var CH : char);
       primitive;
    procedure WRITE_DATA(DATA : DATA_TYPE);
       primitive;

    initialize
      begin
        X.IND:=0
      end;

    trans when PORT.DATA
            begin
              WRITE_DATA(D)
            end;
    trans keyboard :
            begin
              GETCH(CH);
              if ord(CH) > 0 then
                if CH in (.'A'..'Z','a'..'z','0'..'9',' '.) then
                  begin
                    X.IND:=X.IND+1;
                    X.LINE(.X.IND.):=CH;
                    if X.IND >= MAXLIN then
                      begin
                        out PORT.DATA(X);
                        X.IND:=0
                      end
```

```
                end
            else
              if X.IND > 0 then
                begin
                  out PORT.DATA(X);
                  X.IND:=0
                end
          end;

end EX_TERMINAL;

channel CLOCK_ACCESS_POINT(user,provider);
  by user     : SET_CLOCK(DELAY : integer);
                DISABLE_CLOCK;
  by provider : TIME_OUT;
end CLOCK_ACCESS_POINT;

module CLOCK;
  PORT : CLOCK_ACCESS_POINT(provider);
end CLOCK;

process EX_CLOCK for CLOCK;

  var state : (RUNNING,IDLE);

  procedure SETTIMER(DELAY : integer);
    primitive;
  procedure RESETTIMER;
    primitive;
  function TIMEOUT : boolean;
    primitive;

  initialize
    begin
      state:=IDLE
    end;

  trans when PORT.SET_CLOCK
        to RUNNING
          begin
            SETTIMER(DELAY)
          end;
        when PORT.DISABLE_CLOCK
        from RUNNING
        to IDLE
          begin
            RESETTIMER
          end;
  trans provided TIMEOUT
        from RUNNING
        to IDLE
          begin
            out PORT.TIME_OUT
          end;

end EX_CLOCK;

module PROTOCOL;
```

```
  T_PORT : TERMINAL_ACCESS_POINT(all);
  C_PORT : CLOCK_ACCESS_POINT(user);
end PROTOCOL;

process EX_PROTOCOL for PROTOCOL;

  queued T_PORT;

  const MAXSEQ = 1;

  type SEQ_TYPE   = 0..MAXSEQ;
       MSG_TYPE   = record
                      NR,NS : SEQ_TYPE;
                      D     : DATA_TYPE
                    end;
       BUFELPTR = ->BUFEL;
       BUFEL    = record
                    NEXT : BUFELPTR;
                    INFO : DATA_TYPE
                  end;

  var P : BUFELPTR;
      MENS                  : MSG_TYPE;
      N,
      NEXT_FRAME_TO_SEND,
      FRAME_EXPECTED        : SEQ_TYPE;

  procedure ENVIA_MSG(X : MSG_TYPE);
    primitive;
  procedure ENVIA_ACK(NR : SEQ_TYPE);
    primitive;
  function RECEBE_MSG(var M : MSG_TYPE) : boolean;
    primitive;
  function RECEBE_ACK(var NR : SEQ_TYPE) : boolean;
    primitive;
  procedure SENDMSG;
    var X : MSG_TYPE;
    begin
      X.NR:=1-FRAME_EXPECTED;
      X.NS:=NEXT_FRAME_TO_SEND;
      X.D:=P->.INFO;
      ENVIA_MSG(X);
      out C_PORT.SET_CLOCK(10)
    end;
  procedure PUTBUF(D : DATA_TYPE);
    var Q,R : BUFELPTR;
    begin
      new(R);
      R->.INFO:=D;
      R->.NEXT:=nil;
      if P = nil then
        begin
          P:=R;
          SENDMSG
        end
      else
        begin
          Q:=P;
```

```
            while Q->.NEXT <> nil do
              Q:=Q->.NEXT;
            Q->.NEXT:=R
          end
      end;
    procedure GETBUF;
      var Q : BUFELPTR;
      begin
        Q:=P;
        P:=P->.NEXT;
        dispose(Q);
        out C_PORT.DISABLE_CLOCK;
        NEXT_FRAME_TO_SEND:=1-NEXT_FRAME_TO_SEND
      end;

    initialize
      begin
        P:=nil;
        NEXT_FRAME_TO_SEND:=0;
        FRAME_EXPECTED:=0
      end;

    trans when T_PORT.DATA
            begin
              PUTBUF(D)
            end;
          provided RECEBE_MSG(MENS)
            begin
              if MENS.NS = FRAME_EXPECTED then
                begin
                  out T_PORT.DATA(MENS.D);
                  FRAME_EXPECTED:=1-FRAME_EXPECTED
                end;
              if MENS.NR = NEXT_FRAME_TO_SEND then
                GETBUF;
              if P = nil then
                ENVIA_ACK(1-FRAME_EXPECTED)
              else
                SENDMSG
            end;
          provided RECEBE_ACK(N)
            begin
              if N = NEXT_FRAME_TO_SEND then
                GETBUF;
              if P <> nil then
                SENDMSG
            end;
          when C_PORT.TIME_OUT
            begin
              SENDMSG
            end;

  end EX_PROTOCOL;

 /*** instanciacao dos modulos em SIDE_REFINIMENT ***/

  C : CLOCK    with EX_CLOCK;
  T : TERMINAL with EX_TERMINAL;
```

```
  P : PROTOCOL with EX_PROTOCOL;

  connect T.PORT to P.T_PORT;
          C.PORT to P.C_PORT;

end SIDE_REFINEMENT;
```

- A N E X O D -

Implementação gerada a partir da especificação do Anexo C

```
segment simulap;

const
  MAXLIN = 80;
  MAXSEQ = 1;

type
  DATA_TYPE =
    record
      IND : 0..MAXLIN;
      LINE : array (.1..MAXLIN.) of char
    end;
  SEQ_TYPE =
    0..MAXSEQ;
  MSG_TYPE =
    record
      NR,NS : SEQ_TYPE;
      D : DATA_TYPE
    end;
  BUFELPTR =
    ->BUFEL;
  BUFEL =
    record
      NEXT : BUFELPTR;
      INFO : DATA_TYPE
    end;

/*---{ declaracao dos canais especificados (com um prefixo
       Cn, n > 0) e canais auxiliares para a implementacao
       das transicoes expontaneas (com prefixo C0)          }---*/

  channel =
    (C1TERMINAL_ACCESS_POINT,C2CLOCK_ACCESS_POINT,C0EX_CLOCK,
C0EX_PROTOCOL,C0EX_TERMINAL);

/*---{ mensagens associadas a cada canal                    }---*/

  S1TERMINAL_ACCESS_POINT =
    (S1DATA);
  S2CLOCK_ACCESS_POINT =
    (S2SET_CLOCK,S2DISABLE_CLOCK,S2TIME_OUT);
  S0EX_TERMINAL =
    (R1keyboard);
  S0EX_CLOCK =
    (R2ANY);
  S0EX_PROTOCOL =
    (R3ANY);

  S1TYPE =
    ->S0TYPE;
```

```
/*---{ estrutura de dados que representa uma mensagem
       enviada de um modulo emissor para um modulo
       receptor                                          }---*/

  S0TYPE =
    record
      NEXT : S1TYPE;
      case CHANNEL of
        C1TERMINAL_ACCESS_POINT :
         (case T1TERMINAL_ACCESS_POINT:S1TERMINAL_ACCESS_POINT of
            S1DATA :
              (D1DATA : record
                D : DATA_TYPE
              end;
              );
          );
        C2CLOCK_ACCESS_POINT :
          (case T2CLOCK_ACCESS_POINT : S2CLOCK_ACCESS_POINT of
            S2SET_CLOCK :
              (D2SET_CLOCK : record
                DELAY : integer
              end;
              );
            S2DISABLE_CLOCK :
              ();
            S2TIME_OUT :
              ();
          );
        C0EX_CLOCK :
          (case T0EX_CLOCK : S0EX_CLOCK of
            R2ANY :
              ();
          );
        C0EX_PROTOCOL :
          (case T0EX_PROTOCOL : S0EX_PROTOCOL of
            R3ANY :
              ();
          );
        C0EX_TERMINAL :
          (case T0EX_TERMINAL : S0EX_TERMINAL of
            R1keyboard :
              ();
          );
    end;

  I1TYPE =
    ->I0TYPE;
  C1TYPE =
    ->C0TYPE;
  P1TYPE =
    ->P0TYPE;

/*---{ estrutura de dados que representa um indice de uma
       porta estruturada                                 }---*/

  I0TYPE =
    record
      NEXT : I1TYPE;
```

```
      IDENT : integer
    end;

/*---{ estrutura de dados que aponta para uma estrutura
       CnTYPE, representando um canal                          }---*/

  A0TYPE =
    record
      PROC : P1TYPE;
      CHAN : C1TYPE
    end;

/*---{ estrutura de dados que representa uma porta            }---*/

  C0TYPE =
    record
      IDENT : integer;
      INDLIST : I1TYPE;
      TARGET : A0TYPE
    end;

  P2TYPE =
    packed array (.1..10.) of char;
  PROCESS =
    (P0EX_TERMINAL,P0EX_CLOCK,P0EX_PROTOCOL);

/*---{ estrutura de dados que representa um processo, com
       suas respectivas variaveis de estado                    }---*/

  P0TYPE =
    record
      IDENT : P2TYPE;
      CHANLIST : C1TYPE;
      NEXT : P1TYPE;
      case PROCESS of
        P0EX_TERMINAL :
          (D0EX_TERMINAL : record
            CH : char;
            X : DATA_TYPE;
          end);
        P0EX_CLOCK :
          (D0EX_CLOCK : record
            state : (RUNNING,IDLE);
          end);
        P0EX_PROTOCOL :
          (D0EX_PROTOCOL : record
            P : BUFELPTR;
            MENS : MSG_TYPE;
            N,NEXT_FRAME_TO_SEND,FRAME_EXPECTED : SEQ_TYPE;
          end);
    end;

var
  P0VAR : P1TYPE; /*---{ processo corrente    }---*/
  C0VAR : C1TYPE; /*---{ canal corrente       }---*/
  S0VAR : S1TYPE; /*---{ sinal sendo tratado }---*/
```

```
/*---{ procedimentos auxiliares definidos externamente      }---*/

function FOFUNCTION(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 : A0TYPE);
external;
procedure P7PROCEDURE(P : P1TYPE; var Q : P1TYPE);
external;
procedure P8PROCEDURE(P : P1TYPE);
external;
procedure GETCH(var CH : char);
external;
procedure WRITE_DATA(DATA : DATA_TYPE);
external;
procedure SETTIMER(DELAY : integer);
external;
procedure RESETTIMER;
external;
function TIMEOUT : boolean;
external;
procedure ENVIA_MSG(X : MSG_TYPE);
external;
procedure ENVIA_ACK(NR : SEQ_TYPE);
external;
function RECEBE_MSG(var M : MSG_TYPE) : boolean;
external;
function RECEBE_ACK(var NR : SEQ_TYPE) : boolean;
external;

/*---{ procedimentos que representam o comportamento dos
       processos                                          }---*/

procedure EX_TERMINAL;
external;
procedure EX_TERMINAL;
label
  1;
var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
begin
  C1VAR:=C0VAR ;
  S1VAR:=S0VAR ;
  with P0VAR->.D0EX_TERMINAL do
```

```
begin

  /*---{ "when PORT.DATA"
         como PORT eh a primeira porta especificada,
         no modulo TERMINAL, essa eh identificada pelo
         numero 1                                          }---*/

  if COVAR->.IDENT = 1 then
    if SOVAR->.T1TERMINAL_ACCESS_POINT = S1DATA then
      with SOVAR->.D1DATA do
        begin
          WRITE_DATA(D );
          goto 1
        end
  ;

  /*---{ como ha uma transicao expontanea, nomeada
         por "keyboard", essa eh identificada pelo
         numero 0                                          }---*/

  if COVAR->.IDENT = 0 then
    if SOVAR->.TOEX_TERMINAL = R1keyboard then
      begin
        GETCH(CH );
        if ord(CH ) > 0 then
          if CH in (.'A'..'Z','a'..'z','0'..'9',' '.) then
            begin
              X.IND:=X.IND + 1 ;
              X.LINE(.X.IND .):=CH ;
              if X.IND >= MAXLIN then
                begin
                  begin
                    POPROCEDURE(1 );
                   new(SOVAR,C1TERMINAL_ACCESS_POINT,S1DATA);
                    SOVAR->.T1TERMINAL_ACCESS_POINT:=S1DATA ;
                    SOVAR->.D1DATA.D:=X ;
                    OUT
                  end;
                  X.IND:=0
                end
            end
          else
            if X.IND > 0 then
              begin
                begin
                  POPROCEDURE(1 );
                  new(SOVAR,C1TERMINAL_ACCESS_POINT,S1DATA);
                  SOVAR->.T1TERMINAL_ACCESS_POINT:=S1DATA ;
                  SOVAR->.D1DATA.D:=X ;
                  OUT
                end;
                X.IND:=0
              end
        ;
        goto 1
      end
  ;
end;
```

```
  /*---{ liberacao da estrutura utilizada na representacao
         do sinal tratado                                    }---*/

  1 : case C1VAR->.IDENT of
    1 :
      case S1VAR->.T1TERMINAL_ACCESS_POINT of
        S1DATA :
          dispose(S1VAR ,C1TERMINAL_ACCESS_POINT ,S1DATA );
      end;
    0 :
      case S1VAR->.T0EX_TERMINAL of
        R1keyboard :
          dispose(S1VAR ,C0EX_TERMINAL ,R1keyboard );
      end;
  end;
end;

procedure EX_CLOCK;
external;
procedure EX_CLOCK;
label
  1;
var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
begin
  C1VAR:=C0VAR ;
  S1VAR:=S0VAR ;
  with P0VAR->.D0EX_CLOCK do
    begin
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T2CLOCK_ACCESS_POINT = S2SET_CLOCK then
          with S0VAR->.D2SET_CLOCK do
            begin
              state:=RUNNING ;
              SETTIMER(DELAY );
              goto 1
            end
      ;
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T2CLOCK_ACCESS_POINT = S2DISABLE_CLOCK then
          if state in (.RUNNING .) then
            begin
              state:=IDLE ;
              RESETTIMER;
              goto 1
            end
      ;
      if C0VAR->.IDENT = 0 then
        if S0VAR->.T0EX_CLOCK = R2ANY then
          if TIMEOUT then
            if state in (.RUNNING .) then
              begin
                state:=IDLE ;
                begin
                  P0PROCEDURE(1 );
                  new(S0VAR ,C2CLOCK_ACCESS_POINT ,S2TIME_OUT );
                  S0VAR->.T2CLOCK_ACCESS_POINT:=S2TIME_OUT ;
```

```
                    OUT
                  end;
                  goto 1
                end
        ;
      end;
    1 : case C1VAR->.IDENT of
      1 :
        case S1VAR->.T2CLOCK_ACCESS_POINT of
          S2SET_CLOCK :
            dispose(S1VAR ,C2CLOCK_ACCESS_POINT ,S2SET_CLOCK );
          S2DISABLE_CLOCK :
            dispose(S1VAR ,C2CLOCK_ACCESS_POINT ,S2DISABLE_CLOCK );
        end;
      0 :
        case S1VAR->.T0EX_CLOCK of
          R2ANY :
            dispose(S1VAR ,C0EX_CLOCK ,R2ANY );
        end;
    end;
end;

procedure EX_PROTOCOL;
external;
procedure EX_PROTOCOL;
label
  1;
var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
procedure SENDMSG;
var
  X : MSG_TYPE;
begin
  with P0VAR->.D0EX_PROTOCOL do
    begin
      X.NR:=1 - FRAME_EXPECTED ;
      X.NS:=NEXT_FRAME_TO_SEND ;
      X.D:=P->.INFO ;
      ENVIA_MSG(X );
      begin
        P0PROCEDURE(2 );
        new(S0VAR ,C2CLOCK_ACCESS_POINT ,S2SET_CLOCK );
        S0VAR->.T2CLOCK_ACCESS_POINT:=S2SET_CLOCK ;
        S0VAR->.D2SET_CLOCK.DELAY:=10 ;
        OUT
      end
    end
end;
procedure PUTBUF(D : DATA_TYPE);
var
  Q,R : BUFELPTR;
begin
  with P0VAR->.D0EX_PROTOCOL do
    begin
      new(R );
      R->.INFO:=D ;
      R->.NEXT:=nil ;
```

```
      if P = nil then
        begin
          P:=R ;
          SENDMSG
        end
      else
        begin
          Q:=P ;
          while Q->.NEXT <> nil do
            Q:=Q->.NEXT ;
          Q->.NEXT:=R
        end
    end
end;
procedure GETBUF;
var
  Q : BUFELPTR;
begin
  with POVAR->.DOEX_PROTOCOL do
    begin
      Q:=P ;
      P:=P->.NEXT ;
      dispose(Q );
      begin
        POPROCEDURE(2 );
        new(SOVAR ,C2CLOCK_ACCESS_POINT ,S2DISABLE_CLOCK );
        SOVAR->.T2CLOCK_ACCESS_POINT:=S2DISABLE_CLOCK ;
        OUT
      end;
      NEXT_FRAME_TO_SEND:=1 - NEXT_FRAME_TO_SEND
    end
end;
begin
  C1VAR:=COVAR ;
  S1VAR:=SOVAR ;
  with POVAR->.DOEX_PROTOCOL do
    begin
      if COVAR->.IDENT = 1 then
        if SOVAR->.T1TERMINAL_ACCESS_POINT = S1DATA then
          with SOVAR->.D1DATA do
            begin
              PUTBUF(D );
              goto 1
            end
      ;
      if COVAR->.IDENT = 0 then
        if SOVAR->.TOEX_PROTOCOL = R3ANY then
          if RECEBE_MSG(MENS ) then
            begin
              if MENS.NS = FRAME_EXPECTED then
                begin
                  begin
                    POPROCEDURE(1 );
                    new(SOVAR ,C1TERMINAL_ACCESS_POINT ,S1DATA );
                    SOVAR->.T1TERMINAL_ACCESS_POINT:=S1DATA ;
                    SOVAR->.D1DATA.D:=MENS.D ;
                    OUT
                  end;
```

```
                    FRAME_EXPECTED:=1 - FRAME_EXPECTED
                  end
                ;
                if MENS.NR = NEXT_FRAME_TO_SEND then
                  GETBUF
                ;
                if P = nil then
                  ENVIA_ACK(1 - FRAME_EXPECTED )
                else
                  SENDMSG;
                goto 1
              end
        ;
        if C0VAR->.IDENT = 0 then
          if S0VAR->.T0EX_PROTOCOL = R3ANY then
            if RECEBE_ACK(N ) then
              begin
                if N = NEXT_FRAME_TO_SEND then
                  GETBUF
                ;
                if P <> nil then
                  SENDMSG
                ;
                goto 1
              end
        ;
        if C0VAR->.IDENT = 2 then
          if S0VAR->.T2CLOCK_ACCESS_POINT = S2TIME_OUT then
            begin
              SENDMSG;
              goto 1
            end
        ;
      end;
    1 : case C1VAR->.IDENT of
      1 :
        case S1VAR->.T1TERMINAL_ACCESS_POINT of
          S1DATA :
            dispose(S1VAR ,C1TERMINAL_ACCESS_POINT ,S1DATA );
        end;
      0 :
        case S1VAR->.T0EX_PROTOCOL of
          R3ANY :
            dispose(S1VAR ,C0EX_PROTOCOL ,R3ANY );
        end;
      2 :
        case S1VAR->.T2CLOCK_ACCESS_POINT of
          S2TIME_OUT :
            dispose(S1VAR ,C2CLOCK_ACCESS_POINT ,S2TIME_OUT );
        end;
    end;
end;

/*---{ procedimento de geracao e inicializacao das
       estruturas de dados que representam a arquitetura
       do sistema especificado                                    }---*/

procedure IOSIDE_REFINEMENT(var P1VAR : P1TYPE);
```

```
external;
procedure IOSIDE_REFINEMENT;
var
  P2VAR : P1TYPE;
  A0VAR,A1VAR : A0TYPE;
procedure IOEX_TERMINAL(var P1VAR : P1TYPE);
begin

  /*---{ geracao das estruturas de dados que representam
         um modulo, com suas respectivas portas              }---*/

  new(P0VAR ,P0EX_TERMINAL );
  P1PROCEDURE('EX_TERMINA' ,P1VAR );
  P3PROCEDURE(false ,0 );
  P3PROCEDURE(false ,1 );

  /*---{inicializacao das variaveis de estado               }---*/

  with P0VAR->.D0EX_TERMINAL do
    begin
      X.IND:=0
    end
end;
procedure IOEX_CLOCK(var P1VAR : P1TYPE);
begin
  new(P0VAR ,P0EX_CLOCK );
  P1PROCEDURE('EX_CLOCK  ' ,P1VAR );
  P3PROCEDURE(false ,0 );
  P3PROCEDURE(false ,1 );
  with P0VAR->.D0EX_CLOCK do
    begin
      state:=IDLE
    end
end;
procedure IOEX_PROTOCOL(var P1VAR : P1TYPE);
begin
  new(P0VAR ,P0EX_PROTOCOL );
  P1PROCEDURE('EX_PROTOCO' ,P1VAR );
  P3PROCEDURE(false ,0 );
  P3PROCEDURE(true ,1 );
  P3PROCEDURE(false ,2 );
  with P0VAR->.D0EX_PROTOCOL do
    begin
      P:=nil ;
      NEXT_FRAME_TO_SEND:=0 ;
      FRAME_EXPECTED:=0
    end
end;
begin
  P2VAR:=nil ;
  IOEX_CLOCK(P2VAR );
  IOEX_TERMINAL(P2VAR );
  IOEX_PROTOCOL(P2VAR );

  /*---{ conexao entre as portas do sistema                 }---*/

  P7PROCEDURE(P2VAR ,P1VAR );
  P5PROCEDURE(2 ,1 ,P1VAR );
```

```
    P6PROCEDURE(A0VAR );
    P5PROCEDURE(1 ,1 ,P1VAR );
    P6PROCEDURE(A1VAR );
    A0VAR.CHAN->.TARGET:=A1VAR ;
    A1VAR.CHAN->.TARGET:=A0VAR ;
    P5PROCEDURE(3 ,1 ,P1VAR );
    P6PROCEDURE(A0VAR );
    P5PROCEDURE(1 ,2 ,P1VAR );
    P6PROCEDURE(A1VAR );
    A0VAR.CHAN->.TARGET:=A1VAR ;
    A1VAR.CHAN->.TARGET:=A0VAR ;
    P8PROCEDURE(P1VAR );
end;
```

- A N E X O E -

Núcleo de exploração para a implementação apresentada no Anexo D

```
program NUCLEOP(input,output);

/*---{ Obs.: todas as linhas de codigo, com comentarios
            'a direita, dependem da implementacao gerada
            pelo pre'-processador Estelle/83                    }---*/

const
  IDENTLEN = 10;

type

  /*---{ canais e mensagens que representam as transicoes
         expontaneas, definidas na especificacao                }---*/

  CHANNEL        = (C0EX_TERMINAL,C0EX_CLOCK);                    /**/
  S0EX_TERMINAL  = (R1KEYBOARD);                                  /**/
  S0EX_CLOCK     = (R2ANY);                                       /**/
  S1TYPE         = ->S0TYPE;                                      /**/
  S0TYPE         = record
                     NEXT : S1TYPE;
                     case CHANNEL of
                       C0EX_TERMINAL :                            /**/
                         (case T0EX_TERMINAL:S0EX_TERMINAL of     /**/
                            R1KEYBOARD :                          /**/
                              ()                                  /**/
                         );                                       /**/
                       C0EX_CLOCK :                               /**/
                         (case T0EX_CLOCK : S0EX_CLOCK of         /**/
                            R2ANY :                               /**/
                              ()                                  /**/
                         );
                   end;
  I1TYPE         = ->I0TYPE;
  C1TYPE         = ->C0TYPE;
  P1TYPE         = ->P0TYPE;
  I0TYPE         = record
                     NEXT  : I1TYPE;
                     IDENT : integer;
                   end;
  A0TYPE         = record
                     PROC : P1TYPE;
                     CHAN : C1TYPE;
                   end;
  C0TYPE         = record
                     IDENT   : integer;
                     INDLIST : I1TYPE;
                     TARGET  : A0TYPE;
                     NEXT    : C1TYPE;
                     case QUEUED : boolean of
                       false :
                         ();
                       true :
```

```
                        (HEAD : S1TYPE);
                    end;
  P2TYPE          = packed array (.1..10.) of char;
  P0TYPE          = record
                      IDENT       : P2TYPE;
                      CHANLIST    : C1TYPE;
                      NEXT,
                      REFINEMENT  : P1TYPE
                    end;


var
  P0VAR           : P1TYPE;
  C0VAR           : C1TYPE;
  S0VAR           : S1TYPE;
  SIGNAL_PENDING  : integer;

procedure IOSIDE_REFINEMENT(var P1VAR : P1TYPE);                    /**/
  external;                                                         /**/
                                                                    /**/
procedure EX_MEDIUM;                                                /**/
  external;                                                         /**/
                                                                    /**/
procedure EX_TERMINAL;                                              /**/
  external;                                                         /**/
                                                                    /**/
procedure EX_CLOCK;                                                 /**/
  external;                                                         /**/
                                                                    /**/
procedure EX_PROTOCOL;                                              /**/
  external;                                                         /**/
                                                                    /**/
procedure OUT; external;
procedure OUT;

procedure RENDEZVOUS;

var
  SAVEPROCESS : P1TYPE;
  SAVECHANNEL : C1TYPE;

begin
  SAVEPROCESS:=P0VAR;
  SAVECHANNEL:=C0VAR;
  P0VAR:=C0VAR->.TARGET.PROC;
  C0VAR:=C0VAR->.TARGET.CHAN;

  /*---{ chamada aos procedimentos com comunicacao
         do tipo rendezvous                                      }---*/

  if P0VAR->.IDENT = 'EX_TERMINA' then                              /**/
    EX_TERMINAL                                                     /**/
  else                                                              /**/
    if P0VAR->.IDENT = 'EX_CLOCK  ' then                            /**/
      EX_CLOCK                                                      /**/
    else                                                            /**/
      if P0VAR->.IDENT = 'EX_PROTOCO' then                          /**/
        EX_PROTOCOL;                                                /**/
```

```
  COVAR:=SAVECHANNEL;
  POVAR:=SAVEPROCESS
end;

begin
  with COVAR->.TARGET.CHAN-> do
    if QUEUED then

      /*---{ comunicacao assincrona (com filas)                    }---*/

      begin
        SIGNAL_PENDING:=SIGNAL_PENDING+1;
        SOVAR->.NEXT:=HEAD;
        HEAD:=SOVAR
      end
    else

      /*---{ comunicacao sincrona                                  }---*/

      RENDEZVOUS
end;

procedure SYSTEM_INIT;

/*---{ chamada `a rotina que gera e inicializa as
       estruturas de dados que representam a arquitetura
       do sistema especificado                                    }---*/

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

begin
  P:=nil;
  IOSIDE_REFINEMENT(P);                                           /**/
  POVAR:=P->.REFINEMENT;
  dispose(P);
  P:=POVAR;
  SIGNAL_PENDING:=0;
  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
end;

/*---{ procedimento que define a polictica de
       escalonamento, responsavel pela execucao
       pseudo-paralela dos procedimentos que
       implementam o comportamento dos modulos              }---*/

procedure SCHEDULER;

/*---{ localizacao da mensagem a ser tratada               }---*/

procedure GETSIGNAL;

var
  FOUND : boolean;
  S     : S1TYPE;

begin
  FOUND:=false;
  SIGNAL_PENDING:=SIGNAL_PENDING-1;
```

```
  repeat
    while COVAR = nil do
      begin
        POVAR:=POVAR->.NEXT;
        COVAR:=POVAR->.CHANLIST
      end;
    if COVAR->.QUEUED then
      begin
        SOVAR:=COVAR->.HEAD;
        if SOVAR <> nil then
          begin
            FOUND:=true;
            S:=nil;
            while SOVAR->.NEXT <> nil do
              begin
                S:=SOVAR;
                SOVAR:=SOVAR->.NEXT
              end
          end
      end;
    if not FOUND then
      COVAR:=COVAR->.NEXT
  until FOUND;
  if S = nil then
    COVAR->.HEAD:=nil
  else
    S->.NEXT:=nil
end;

begin
  repeat
    if SIGNAL_PENDING > 0 then
      begin
        GETSIGNAL;

        /*---{ acionamento das rotinas que definem o
               comportamento dos modulos frente a uma
               mensagem a ser tratada                    }---*/

        if POVAR->.IDENT = 'EX_TERMINA' then              /**/
          EX_TERMINAL                                     /**/
        else                                              /**/
          if POVAR->.IDENT = 'EX_CLOCK  ' then            /**/
            EX_CLOCK                                      /**/
          else                                            /**/
            if POVAR->.IDENT = 'EX_PROTOCO' then          /**/
              EX_PROTOCOL                                 /**/
      end
    else

      /*---{ geracao das mensagens que acionam as
             transicoes expontaneas e camada das
             rotinas que tratam essas mensagens          }---*/

      begin
        POVAR:=POVAR->.NEXT;
        COVAR:=POVAR->.CHANLIST;
        if POVAR->.IDENT = 'EX_TERMINA' then              /**/
```

```
          begin                                                    /**/
            new(SOVAR,COEX_TERMINAL,R1KEYBOARD);                   /**/
            SOVAR->.TOEX_TERMINAL:=R1KEYBOARD;                     /**/
            EX_TERMINAL                                            /**/
          end                                                      /**/
        else                                                       /**/
          if POVAR->.IDENT = 'EX_CLOCK   ' then                    /**/
            begin                                                  /**/
              new(SOVAR,COEX_CLOCK,R2ANY);                         /**/
              SOVAR->.TOEX_CLOCK:=R2ANY;                           /**/
              EX_CLOCK                                             /**/
            end                                                    /**/
      end
  until false
end;

begin
  SYSTEM_INIT;
  SCHEDULER
end.
```

- A N E X O F -

Especificação de um sistema para a análise do protocolo apresentado no Anexo C (meio ideal)

```
module SYSTEM;
end SYSTEM;

refinement SYSTEM_REFINEMENT for SYSTEM;

  const MAXSEQ = 1;
        MAXLIN = 80;

  type boolean    = (false,true);
       DATA_TYPE = record
                     IND  : 0..MAXLIN;
                     LINE : array(.1..MAXLIN.) of char
                   end;
       SEQ_TYPE  = 0..MAXSEQ;
       MSG_TYPE  = record
                     NR,NS : SEQ_TYPE;
                     D     : DATA_TYPE
                   end;

  channel MEDIUM_ACCESS_POINT(all);
    by all : MSG(M : MSG_TYPE);
             ACK(NR : SEQ_TYPE);
  end MEDIUM_ACCESS_POINT;

  module MEDIUM;
    PORT : array(.boolean.) of MEDIUM_ACCESS_POINT(all);
  end MEDIUM;

  process EX_MEDIUM for MEDIUM;

    trans when PORT(.B.).MSG
            begin
              out PORT(.not B.).MSG(M)
            end;
          when PORT(.B.).ACK
            begin
              out PORT(.not B.).ACK(NR)
            end;

  end EX_MEDIUM;

  module SIDE;
    PORT : MEDIUM_ACCESS_POINT(all);
  end SIDE;

  refinement SIDE_REFINEMENT(CHOICE : boolean) for SIDE;

    channel TERMINAL_ACCESS_POINT(all);
      by all : DATA(D : DATA_TYPE);
    end TERMINAL_ACCESS_POINT;
```

```
module TERMINAL;
  PORT : TERMINAL_ACCESS_POINT(all);
end TERMINAL;

process EX_TERMINAL for TERMINAL;

  var THIS_SIDE : boolean;
      CH        : char;
      X         : DATA_TYPE;
  procedure GETCH(CHOICE : boolean; var CH : char);
    primitive;
  procedure WRITE_DATA(CHOICE : boolean; DATA : DATA_TYPE);
    primitive;

  initialize
    begin
      THIS_SIDE:=CHOICE;
      X.IND:=0
    end;

  trans when PORT.DATA
          begin
            WRITE_DATA(THIS_SIDE,D)
          end;
  trans keyboard :
          begin
            GETCH(THIS_SIDE,CH);
            if ord(CH) > 0 then
              if CH in (.'A'..'Z','a'..'z','0'..'9',' '.) then
                begin
                  X.IND:=X.IND+1;
                  X.LINE(.X.IND.):=CH;
                  if X.IND >= MAXLIN then
                    begin
                      out PORT.DATA(X);
                      X.IND:=0
                    end
                end
            else
              if X.IND > 0 then
                begin
                  out PORT.DATA(X);
                  X.IND:=0
                end
          end;

end EX_TERMINAL;

channel CLOCK_ACCESS_POINT(user,provider);
  by user     : SET_CLOCK(DELAY : integer);
                DISABLE_CLOCK;
  by provider : TIME_OUT;
end CLOCK_ACCESS_POINT;

module CLOCK;
  PORT : CLOCK_ACCESS_POINT(provider);
end CLOCK;
```

```
process EX_CLOCK for CLOCK;

  var state     : (RUNNING,IDLE);
      THIS_SIDE : boolean;

  procedure SETTIMER(CHOICE : boolean; DELAY : integer);
    primitive;
  procedure RESETTIMER(CHOICE : boolean);
    primitive;
  function TIMEOUT(CHOICE : boolean) : boolean;
    primitive;

  initialize
    begin
      state:=IDLE;
      THIS_SIDE:=CHOICE
    end;

  trans when PORT.SET_CLOCK
        to RUNNING
          begin
            SETTIMER(THIS_SIDE,DELAY)
          end;
        when PORT.DISABLE_CLOCK
        from RUNNING to IDLE
          begin
            RESETTIMER(THIS_SIDE)
          end;
  trans provided TIMEOUT(THIS_SIDE)
        from RUNNING to IDLE
          begin
            out PORT.TIME_OUT
          end;

end EX_CLOCK;

module PROTOCOL;
  T_PORT : TERMINAL_ACCESS_POINT(all);
  C_PORT : CLOCK_ACCESS_POINT(user);
  M_PORT : MEDIUM_ACCESS_POINT(all);
end PROTOCOL;

process EX_PROTOCOL for PROTOCOL;

  queued T_PORT,M_PORT;

  type BUFELPTR = ->BUFEL;
       BUFEL    = record
                    NEXT : BUFELPTR;
                    INFO : DATA_TYPE
                  end;

  var P : BUFELPTR;
      NEXT_FRAME_TO_SEND,FRAME_EXPECTED : SEQ_TYPE;

  procedure SENDMSG;
    var X : MSG_TYPE;
    begin
```

```
      X.NR:=1-FRAME_EXPECTED;
      X.NS:=NEXT_FRAME_TO_SEND;
      X.D:=P->.INFO;
      out M_PORT.MSG(X);
      out C_PORT.SET_CLOCK(10)
    end;
  procedure PUTBUF(D : DATA_TYPE);
    var Q,R : BUFELPTR;
    begin
      new(R);
      R->.INFO:=D;
      R->.NEXT:=nil;
      if P = nil then
        begin
          P:=R;
          SENDMSG
        end
      else
        begin
          Q:=P;
          while Q->.NEXT <> nil do
            Q:=Q->.NEXT;
          Q->.NEXT:=R
        end
    end;
  procedure GETBUF;
    var Q : BUFELPTR;
    begin
      Q:=P;
      P:=P->.NEXT;
      dispose(Q);
      out C_PORT.DISABLE_CLOCK;
      NEXT_FRAME_TO_SEND:=1-NEXT_FRAME_TO_SEND
    end;

  initialize
    begin
      P:=nil;
      NEXT_FRAME_TO_SEND:=0;
      FRAME_EXPECTED:=0
    end;

  trans when T_PORT.DATA
          begin
            PUTBUF(D)
          end;
        when M_PORT.MSG
          begin
            if M.NS = FRAME_EXPECTED then
              begin
                out T_PORT.DATA(M.D);
                FRAME_EXPECTED:=1-FRAME_EXPECTED
              end;
            if M.NR = NEXT_FRAME_TO_SEND then
              GETBUF;
            if P = nil then
              out M_PORT.ACK(1-FRAME_EXPECTED)
            else
```

```
                SENDMSG
            end;
          when M_PORT.ACK
            begin
              if NR = NEXT_FRAME_TO_SEND then
                GETBUF;
              if P <> nil then
                SENDMSG
            end;
          when C_PORT.TIME_OUT
            begin
              SENDMSG
            end;

    end EX_PROTOCOL;

   /*** instanciacao dos modulos em SIDE_REFINIMENT ***/

    C : CLOCK    with EX_CLOCK;
    T : TERMINAL with EX_TERMINAL;
    P : PROTOCOL with EX_PROTOCOL;

    connect T.PORT to P.T_PORT;
            C.PORT to P.C_PORT;

    replace PORT by P.M_PORT;

  end SIDE_REFINEMENT;

 /*** instanciacao dos modulos para SYSTEM_REFINEMENT ***/

  S0 : SIDE   with SIDE_REFINEMENT(false);
  S1 : SIDE   with SIDE_REFINEMENT(true);
  M  : MEDIUM with EX_MEDIUM;

  connect S0.PORT to M.PORT(.false.);
          S1.PORT to M.PORT(.true.);

end SYSTEM_REFINEMENT;
```

- A N E X O G -

Implementação do sistema apresentado no Anexo F

```
segment simula1;

const
  MAXSEQ = 1;
  MAXLIN = 80;

type
  DATA_TYPE =
    record
      IND : 0..MAXLIN;
      LINE : array (.1..MAXLIN.) of char
    end;
  SEQ_TYPE =
    0..MAXSEQ;
  MSG_TYPE =
    record
      NR,NS : SEQ_TYPE;
      D : DATA_TYPE
    end;
  BUFELPTR =
    ->BUFEL;
  BUFEL =
    record
      NEXT : BUFELPTR;
      INFO : DATA_TYPE
    end;
  channel =
    (C1MEDIUM_ACCESS_POINT,C2TERMINAL_ACCESS_POINT,
C3CLOCK_ACCESS_POINT,C0EX_CLOCK,C0EX_TERMINAL);
  S1MEDIUM_ACCESS_POINT =
    (S1MSG,S1ACK);
  S2TERMINAL_ACCESS_POINT =
    (S2DATA);
  S3CLOCK_ACCESS_POINT =
    (S3SET_CLOCK,S3DISABLE_CLOCK,S3TIME_OUT);
  S0EX_TERMINAL =
    (R1keyboard);
  S0EX_CLOCK =
    (R2ANY);
  S1TYPE =
    ->S0TYPE;
  S0TYPE =
    record
      NEXT : S1TYPE;
      case CHANNEL of
        C1MEDIUM_ACCESS_POINT :
          (case T1MEDIUM_ACCESS_POINT : S1MEDIUM_ACCESS_POINT of
            S1MSG :
              (D1MSG : record
                M : MSG_TYPE
              end;
              );
```

```
            S1ACK :
              (D1ACK : record
                NR : SEQ_TYPE
              end;
              );
          );
        C2TERMINAL_ACCESS_POINT :
         (case T2TERMINAL_ACCESS_POINT:S2TERMINAL_ACCESS_POINT of
            S2DATA :
              (D2DATA : record
                D : DATA_TYPE
              end;
              );
          );
        C3CLOCK_ACCESS_POINT :
          (case T3CLOCK_ACCESS_POINT : S3CLOCK_ACCESS_POINT of
            S3SET_CLOCK :
              (D3SET_CLOCK : record
                DELAY : integer
              end;
              );
            S3DISABLE_CLOCK :
              ();
            S3TIME_OUT :
              ();
          );
        C0EX_CLOCK :
          (case T0EX_CLOCK : S0EX_CLOCK of
            R2ANY :
              ();
          );
        C0EX_TERMINAL :
          (case T0EX_TERMINAL : S0EX_TERMINAL of
            R1keyboard :
              ();
          );
  end;
I1TYPE =
  ->I0TYPE;
C1TYPE =
  ->C0TYPE;
P1TYPE =
  ->P0TYPE;
I0TYPE =
  record
    NEXT : I1TYPE;
    IDENT : integer
  end;
A0TYPE =
  record
    PROC : P1TYPE;
    CHAN : C1TYPE
  end;
C0TYPE =
  record
    IDENT : integer;
    INDLIST : I1TYPE;
    TARGET : A0TYPE
```

```
    end;
  P2TYPE =
    packed array (.1..10.) of char;
  PROCESS =
    (P0EX_MEDIUM,P0EX_TERMINAL,P0EX_CLOCK,P0EX_PROTOCOL);
  P0TYPE =
    record
      IDENT : P2TYPE;
      CHANLIST : C1TYPE;
      NEXT : P1TYPE;
      case PROCESS of
        P0EX_MEDIUM :
          ();
        P0EX_TERMINAL :
          (D0EX_TERMINAL : record
            THIS_SIDE : boolean;
            CH : char;
            X : DATA_TYPE;
          end);
        P0EX_CLOCK :
          (D0EX_CLOCK : record
            state : (RUNNING,IDLE);
            THIS_SIDE : boolean;
          end);
        P0EX_PROTOCOL :
          (D0EX_PROTOCOL : record
            P : BUFELPTR;
            NEXT_FRAME_TO_SEND,FRAME_EXPECTED : SEQ_TYPE;
          end);
    end;

var
  P0VAR : P1TYPE;
  C0VAR : C1TYPE;
  S0VAR : S1TYPE;

function F0FUNCTION(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 : A0TYPE);
external;
procedure P7PROCEDURE(P : P1TYPE; var Q : P1TYPE);
external;
procedure P8PROCEDURE(P : P1TYPE);
external;
```

```
procedure GETCH(CHOICE : boolean; var CH : char);
external;
procedure WRITE_DATA(CHOICE : boolean; DATA : DATA_TYPE);
external;
procedure SETTIMER(CHOICE : boolean; DELAY : integer);
external;
procedure RESETTIMER(CHOICE : boolean);
external;
function TIMEOUT(CHOICE : boolean) : boolean;
external;
function FOBOOLEAN(I : integer) : boolean;
external;
function FOBOOLEAN ;
begin
  if I = 0 then
    FOBOOLEAN:=false
  else
    FOBOOLEAN:=true
end;

procedure EX_MEDIUM;
external;
procedure EX_MEDIUM;
label
  1;
var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
procedure P10PROCEDURE(B : boolean);
begin
  with S0VAR->.D1MSG do
    begin
      begin
        P0PROCEDURE(1 );
        P4PROCEDURE(ord(not B ) ,0 );
        new(S0VAR ,C1MEDIUM_ACCESS_POINT ,S1MSG );
        S0VAR->.T1MEDIUM_ACCESS_POINT:=S1MSG ;
        S0VAR->.D1MSG.M:=M ;
        OUT
      end;
      goto 1
    end;
end;
procedure P11PROCEDURE(B : boolean);
begin
  with S0VAR->.D1ACK do
    begin
      begin
        P0PROCEDURE(1 );
        P4PROCEDURE(ord(not B ) ,0 );
        new(S0VAR ,C1MEDIUM_ACCESS_POINT ,S1ACK );
        S0VAR->.T1MEDIUM_ACCESS_POINT:=S1ACK ;
        S0VAR->.D1ACK.NR:=NR ;
        OUT
      end;
      goto 1
    end;
end;
```

```
begin
  C1VAR:=C0VAR ;
  S1VAR:=S0VAR ;
  if C0VAR->.IDENT = 1 then
    if S0VAR->.T1MEDIUM_ACCESS_POINT = S1MSG then
      P10PROCEDURE(F0BOOLEAN(F0FUNCTION(0 ) ) )
  ;
  if C0VAR->.IDENT = 1 then
    if S0VAR->.T1MEDIUM_ACCESS_POINT = S1ACK then
      P11PROCEDURE(F0BOOLEAN(F0FUNCTION(0 ) ) )
  ;
  1 : case C1VAR->.IDENT of
    1 :
      case S1VAR->.T1MEDIUM_ACCESS_POINT of
        S1MSG :
          dispose(S1VAR ,C1MEDIUM_ACCESS_POINT ,S1MSG );
        S1ACK :
          dispose(S1VAR ,C1MEDIUM_ACCESS_POINT ,S1ACK );
      end;
  end;
end;

procedure EX_TERMINAL;
external;
procedure EX_TERMINAL;
label
  1;
var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
begin
  C1VAR:=C0VAR ;
  S1VAR:=S0VAR ;
  with P0VAR->.D0EX_TERMINAL do
    begin
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T2TERMINAL_ACCESS_POINT = S2DATA then
          with S0VAR->.D2DATA do
            begin
              WRITE_DATA(THIS_SIDE ,D );
              goto 1
            end
      ;
      if C0VAR->.IDENT = 0 then
        if S0VAR->.T0EX_TERMINAL = R1keyboard then
          begin
            GETCH(THIS_SIDE ,CH );
            if ord(CH ) > 0 then
              if CH in (.'A'..'Z','a'..'z','0'..'9',' '.) then
                begin
                  X.IND:=X.IND + 1 ;
                  X.LINE(.X.IND .):=CH ;
                  if X.IND >= MAXLIN then
                    begin
                      begin
                        P0PROCEDURE(1 );
                       new(S0VAR,C2TERMINAL_ACCESS_POINT,S2DATA);
                        S0VAR->.T2TERMINAL_ACCESS_POINT:=S2DATA ;
```

```
                        S0VAR->.D2DATA.D:=X ;
                        OUT
                      end;
                      X.IND:=0
                    end
                end
              else
                if X.IND > 0 then
                  begin
                    begin
                      POPROCEDURE(1 );
                      new(S0VAR,C2TERMINAL_ACCESS_POINT,S2DATA);
                      S0VAR->.T2TERMINAL_ACCESS_POINT:=S2DATA ;
                      S0VAR->.D2DATA.D:=X ;
                      OUT
                    end;
                    X.IND:=0
                  end
            ;
            goto 1
          end
      ;
    end;
  1 : case C1VAR->.IDENT of
    1 :
      case S1VAR->.T2TERMINAL_ACCESS_POINT of
        S2DATA :
          dispose(S1VAR ,C2TERMINAL_ACCESS_POINT ,S2DATA );
      end;
    0 :
      case S1VAR->.T0EX_TERMINAL of
        R1keyboard :
          dispose(S1VAR ,C0EX_TERMINAL ,R1keyboard );
      end;
  end;
end;

procedure EX_CLOCK;
external;
procedure EX_CLOCK;
label
  1;
var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
begin
  C1VAR:=C0VAR ;
  S1VAR:=S0VAR ;
  with P0VAR->.D0EX_CLOCK do
    begin
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T3CLOCK_ACCESS_POINT = S3SET_CLOCK then
          with S0VAR->.D3SET_CLOCK do
            begin
              state:=RUNNING ;
              SETTIMER(THIS_SIDE ,DELAY );
              goto 1
            end
```

```
      ;
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T3CLOCK_ACCESS_POINT = S3DISABLE_CLOCK then
          if state in (.RUNNING .) then
            begin
              state:=IDLE ;
              RESETTIMER(THIS_SIDE );
              goto 1
            end
      ;
      if C0VAR->.IDENT = 0 then
        if S0VAR->.T0EX_CLOCK = R2ANY then
          if TIMEOUT(THIS_SIDE ) then
            if state in (.RUNNING .) then
              begin
                state:=IDLE ;
                begin
                  P0PROCEDURE(1 );
                  new(S0VAR ,C3CLOCK_ACCESS_POINT ,S3TIME_OUT );
                  S0VAR->.T3CLOCK_ACCESS_POINT:=S3TIME_OUT ;
                  OUT
                end;
                goto 1
              end
      ;
    end;
  1 : case C1VAR->.IDENT of
    1 :
      case S1VAR->.T3CLOCK_ACCESS_POINT of
        S3SET_CLOCK :
          dispose(S1VAR ,C3CLOCK_ACCESS_POINT ,S3SET_CLOCK );
        S3DISABLE_CLOCK :
          dispose(S1VAR ,C3CLOCK_ACCESS_POINT ,S3DISABLE_CLOCK );
      end;
    0 :
      case S1VAR->.T0EX_CLOCK of
        R2ANY :
          dispose(S1VAR ,C0EX_CLOCK ,R2ANY );
      end;
  end;
end;

procedure EX_PROTOCOL;
external;
procedure EX_PROTOCOL;
label
  1;
var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
procedure SENDMSG;
var
  X : MSG_TYPE;
begin
  with P0VAR->.D0EX_PROTOCOL do
    begin
      X.NR:=1 - FRAME_EXPECTED ;
      X.NS:=NEXT_FRAME_TO_SEND ;
```

```
      X.D:=P->.INFO ;
      begin
        POPROCEDURE(3 );
        new(SOVAR ,C1MEDIUM_ACCESS_POINT ,S1MSG );
        SOVAR->.T1MEDIUM_ACCESS_POINT:=S1MSG ;
        SOVAR->.D1MSG.M:=X ;
        OUT
      end;
      begin
        POPROCEDURE(2 );
        new(SOVAR ,C3CLOCK_ACCESS_POINT ,S3SET_CLOCK );
        SOVAR->.T3CLOCK_ACCESS_POINT:=S3SET_CLOCK ;
        SOVAR->.D3SET_CLOCK.DELAY:=10 ;
        OUT
      end
    end
end;
procedure PUTBUF(D : DATA_TYPE);
var
  Q,R : BUFELPTR;
begin
  with POVAR->.DOEX_PROTOCOL do
    begin
      new(R );
      R->.INFO:=D ;
      R->.NEXT:=nil ;
      if P = nil then
        begin
          P:=R ;
          SENDMSG
        end
      else
        begin
          Q:=P ;
          while Q->.NEXT <> nil do
            Q:=Q->.NEXT ;
          Q->.NEXT:=R
        end
    end
end;
procedure GETBUF;
var
  Q : BUFELPTR;
begin
  with POVAR->.DOEX_PROTOCOL do
    begin
      Q:=P ;
      P:=P->.NEXT ;
      dispose(Q );
      begin
        POPROCEDURE(2 );
        new(SOVAR ,C3CLOCK_ACCESS_POINT ,S3DISABLE_CLOCK );
        SOVAR->.T3CLOCK_ACCESS_POINT:=S3DISABLE_CLOCK ;
        OUT
      end;
      NEXT_FRAME_TO_SEND:=1 - NEXT_FRAME_TO_SEND
    end
end;
```

```
begin
  C1VAR:=C0VAR ;
  S1VAR:=S0VAR ;
  with P0VAR->.D0EX_PROTOCOL do
    begin
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T2TERMINAL_ACCESS_POINT = S2DATA then
          with S0VAR->.D2DATA do
            begin
              PUTBUF(D );
              goto 1
            end
      ;
      if C0VAR->.IDENT = 3 then
        if S0VAR->.T1MEDIUM_ACCESS_POINT = S1MSG then
          with S0VAR->.D1MSG do
            begin
              if M.NS = FRAME_EXPECTED then
                begin
                  begin
                    P0PROCEDURE(1 );
                    new(S0VAR ,C2TERMINAL_ACCESS_POINT ,S2DATA );
                    S0VAR->.T2TERMINAL_ACCESS_POINT:=S2DATA ;
                    S0VAR->.D2DATA.D:=M.D ;
                    OUT
                  end;
                  FRAME_EXPECTED:=1 - FRAME_EXPECTED
                end
              ;
              if M.NR = NEXT_FRAME_TO_SEND then
                GETBUF
              ;
              if P = nil then
                begin
                  P0PROCEDURE(3 );
                  new(S0VAR ,C1MEDIUM_ACCESS_POINT ,S1ACK );
                  S0VAR->.T1MEDIUM_ACCESS_POINT:=S1ACK ;
                  S0VAR->.D1ACK.NR:=1 - FRAME_EXPECTED ;
                  OUT
                end
              else
                SENDMSG;
              goto 1
            end
      ;
      if C0VAR->.IDENT = 3 then
        if S0VAR->.T1MEDIUM_ACCESS_POINT = S1ACK then
          with S0VAR->.D1ACK do
            begin
              if NR = NEXT_FRAME_TO_SEND then
                GETBUF
              ;
              if P <> nil then
                SENDMSG
              ;
              goto 1
            end
      ;
```

```
      if C0VAR->.IDENT = 2 then
        if S0VAR->.T3CLOCK_ACCESS_POINT = S3TIME_OUT then
          begin
            SENDMSG;
            goto 1
          end
      ;
    end;
  1 : case C1VAR->.IDENT of
    1 :
      case S1VAR->.T2TERMINAL_ACCESS_POINT of
        S2DATA :
          dispose(S1VAR ,C2TERMINAL_ACCESS_POINT ,S2DATA );
      end;
    3 :
      case S1VAR->.T1MEDIUM_ACCESS_POINT of
        S1MSG :
          dispose(S1VAR ,C1MEDIUM_ACCESS_POINT ,S1MSG );
        S1ACK :
          dispose(S1VAR ,C1MEDIUM_ACCESS_POINT ,S1ACK );
      end;
    2 :
      case S1VAR->.T3CLOCK_ACCESS_POINT of
        S3TIME_OUT :
          dispose(S1VAR ,C3CLOCK_ACCESS_POINT ,S3TIME_OUT );
      end;
  end;
end;

procedure IOSYSTEM_REFINEMENT(var P1VAR : P1TYPE);
external;
procedure IOSYSTEM_REFINEMENT;
var
  P2VAR : P1TYPE;
  A0VAR,A1VAR : A0TYPE;
procedure IOEX_MEDIUM(var P1VAR : P1TYPE);
var
  IOVAR : integer;
begin
  new(P0VAR ,P0EX_MEDIUM );
  P1PROCEDURE('EX_MEDIUM ' ,P1VAR );
  for IOVAR:=0 to 1 do
    begin
      P3PROCEDURE(false ,1 );
      P2PROCEDURE(IOVAR );
    end;
end;
procedure IOSIDE_REFINEMENT(var P1VAR : P1TYPE;CHOICE : boolean);
var
  P2VAR : P1TYPE;
  A0VAR,A1VAR : A0TYPE;
procedure IOEX_TERMINAL(var P1VAR : P1TYPE);
begin
  new(P0VAR ,P0EX_TERMINAL );
  P1PROCEDURE('EX_TERMINA' ,P1VAR );
  P3PROCEDURE(false ,0 );
  P3PROCEDURE(false ,1 );
  with P0VAR->.D0EX_TERMINAL do
```

```
    begin
      THIS_SIDE:=CHOICE ;
      X.IND:=0
    end
end;
procedure IOEX_CLOCK(var P1VAR : P1TYPE);
begin
  new(P0VAR ,P0EX_CLOCK );
  P1PROCEDURE('EX_CLOCK  ' ,P1VAR );
  P3PROCEDURE(false ,0 );
  P3PROCEDURE(false ,1 );
  with P0VAR->.D0EX_CLOCK do
    begin
      state:=IDLE ;
      THIS_SIDE:=CHOICE
    end
end;
procedure IOEX_PROTOCOL(var P1VAR : P1TYPE);
begin
  new(P0VAR ,P0EX_PROTOCOL );
  P1PROCEDURE('EX_PROTOCO' ,P1VAR );
  P3PROCEDURE(true ,1 );
  P3PROCEDURE(false ,2 );
  P3PROCEDURE(true ,3 );
  with P0VAR->.D0EX_PROTOCOL do
    begin
      P:=nil ;
      NEXT_FRAME_TO_SEND:=0 ;
      FRAME_EXPECTED:=0
    end
end;
begin
  P2VAR:=nil ;
  IOEX_CLOCK(P2VAR );
  IOEX_TERMINAL(P2VAR );
  IOEX_PROTOCOL(P2VAR );
  P7PROCEDURE(P2VAR ,P1VAR );
  P3PROCEDURE(false ,1 );
  P5PROCEDURE(2 ,1 ,P1VAR );
  P6PROCEDURE(A0VAR );
  P5PROCEDURE(1 ,1 ,P1VAR );
  P6PROCEDURE(A1VAR );
  A0VAR.CHAN->.TARGET:=A1VAR ;
  A1VAR.CHAN->.TARGET:=A0VAR ;
  P5PROCEDURE(3 ,1 ,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(1 ,3 ,P1VAR );
  P6PROCEDURE(A0VAR.CHAN->.TARGET );
  P8PROCEDURE(P1VAR );
end;
begin
  P2VAR:=nil ;
```

```
  IOSIDE_REFINEMENT(P2VAR ,false );
  IOSIDE_REFINEMENT(P2VAR ,true );
  IOEX_MEDIUM(P2VAR );
  P7PROCEDURE(P2VAR ,P1VAR );
  P5PROCEDURE(3 ,1 ,P1VAR );
  P6PROCEDURE(A0VAR );
  P5PROCEDURE(1 ,1 ,P1VAR );
  P4PROCEDURE(ord(false ) ,0 );
  P6PROCEDURE(A1VAR );
  A0VAR.CHAN->.TARGET:=A1VAR ;
  A1VAR.CHAN->.TARGET:=A0VAR ;
  P5PROCEDURE(2 ,1 ,P1VAR );
  P6PROCEDURE(A0VAR );
  P5PROCEDURE(1 ,1 ,P1VAR );
  P4PROCEDURE(ord(true ) ,0 );
  P6PROCEDURE(A1VAR );
  A0VAR.CHAN->.TARGET:=A1VAR ;
  A1VAR.CHAN->.TARGET:=A0VAR ;
  P8PROCEDURE(P1VAR );
end;
```

- A N E X O H -

Especificação de um sistema para a análise do protocolo apresentado no Anexo D (meio com perdas)

```
module SYSTEM;
end SYSTEM;

refinement SYSTEM_REFINEMENT for SYSTEM;

  const MAXSEQ = 1;
        MAXLIN = 80;

  type boolean   = (false,true);
       DATA_TYPE = record
                     IND  : 0..MAXLIN;
                     LINE : array(.1..MAXLIN.) of char
                   end;
       SEQ_TYPE  = 0..MAXSEQ;
       MSG_TYPE  = record
                     NR,NS : SEQ_TYPE;
                     D     : DATA_TYPE
                   end;

  channel MEDIUM_ACCESS_POINT(all);
    by all : MSG(M : MSG_TYPE);
             ACK(NR : SEQ_TYPE);
  end MEDIUM_ACCESS_POINT;

  module MEDIUM;
    PORT : array(.boolean.) of MEDIUM_ACCESS_POINT(all);
  end MEDIUM;

  process EX_MEDIUM for MEDIUM;

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

    trans when PORT(.B.).MSG
            begin
              if not PERDA_NO_MEIO then
                out PORT(.not B.).MSG(M)
            end;
          when PORT(.B.).ACK
            begin
              if not PERDA_NO_MEIO then
                out PORT(.not B.).ACK(NR)
            end;

  end EX_MEDIUM;
```

```
module SIDE;
  PORT : MEDIUM_ACCESS_POINT(all);
end SIDE;

refinement SIDE_REFINEMENT(CHOICE : boolean) for SIDE;

  channel TERMINAL_ACCESS_POINT(all);
    by all : DATA(D : DATA_TYPE);
  end TERMINAL_ACCESS_POINT;

  module TERMINAL;
    PORT : TERMINAL_ACCESS_POINT(all);
  end TERMINAL;

  process EX_TERMINAL for TERMINAL;

    var THIS_SIDE : boolean;
        CH        : char;
        X         : DATA_TYPE;

    procedure GETCH(CHOICE : boolean; var CH : char);
      primitive;
    procedure WRITE_DATA(CHOICE : boolean; DATA : DATA_TYPE);
      primitive;

    initialize
      begin
        THIS_SIDE:=CHOICE;
        X.IND:=0
      end;

    trans when PORT.DATA
            begin
              WRITE_DATA(THIS_SIDE,D)
            end;
    trans keyboard :
            begin
              GETCH(THIS_SIDE,CH);
              if ord(CH) > 0 then
                if CH in (.'A'..'Z','a'..'z','0'..'9',' '.) then
                  begin
                    X.IND:=X.IND+1;
                    X.LINE(.X.IND.):=CH;
                    if X.IND >= MAXLIN then
                      begin
                        out PORT.DATA(X);
                        X.IND:=0
                      end
                  end
                else
                  if X.IND > 0 then
                    begin
                      out PORT.DATA(X);
                      X.IND:=0
                    end
            end;

  end EX_TERMINAL;
```

```
channel CLOCK_ACCESS_POINT(user,provider);
  by user     : SET_CLOCK(DELAY : integer);
                DISABLE_CLOCK;
  by provider : TIME_OUT;
end CLOCK_ACCESS_POINT;

module CLOCK;
  PORT : CLOCK_ACCESS_POINT(provider);
end CLOCK;

process EX_CLOCK for CLOCK;

  var state     : (RUNNING,IDLE);
      THIS_SIDE : boolean;

  procedure SETTIMER(CHOICE : boolean; DELAY : integer);
    primitive;
  procedure RESETTIMER(CHOICE : boolean);
    primitive;
  function TIMEOUT(CHOICE : boolean) : boolean;
    primitive;

  initialize
    begin
      state:=IDLE;
      THIS_SIDE:=CHOICE
    end;

  trans when PORT.SET_CLOCK
        to RUNNING
          begin
            SETTIMER(THIS_SIDE,DELAY)
          end;
        when PORT.DISABLE_CLOCK
        from RUNNING to IDLE
          begin
            RESETTIMER(THIS_SIDE)
          end;
  trans provided TIMEOUT(THIS_SIDE)
        from RUNNING to IDLE
          begin
            out PORT.TIME_OUT
          end;

end EX_CLOCK;

module PROTOCOL;
  T_PORT : TERMINAL_ACCESS_POINT(all);
  C_PORT : CLOCK_ACCESS_POINT(user);
  M_PORT : MEDIUM_ACCESS_POINT(all);
end PROTOCOL;

process EX_PROTOCOL for PROTOCOL;

  queued T_PORT,M_PORT;

  type BUFELPTR = ->BUFEL;
```

```
      BUFEL     = record
                    NEXT : BUFELPTR;
                    INFO : DATA_TYPE
                  end;

var P : BUFELPTR;
    NEXT_FRAME_TO_SEND,FRAME_EXPECTED : SEQ_TYPE;

procedure SENDMSG;
  var X : MSG_TYPE;
  begin
    X.NR:=1-FRAME_EXPECTED;
    X.NS:=NEXT_FRAME_TO_SEND;
    X.D:=P->.INFO;
    out M_PORT.MSG(X);
    out C_PORT.SET_CLOCK(10)
  end;
procedure PUTBUF(D : DATA_TYPE);
  var Q,R : BUFELPTR;
  begin
    new(R);
    R->.INFO:=D;
    R->.NEXT:=nil;
    if P = nil then
      begin
        P:=R;
        SENDMSG
      end
    else
      begin
        Q:=P;
        while Q->.NEXT <> nil do
          Q:=Q->.NEXT;
        Q->.NEXT:=R
      end
  end;
procedure GETBUF;
  var Q : BUFELPTR;
  begin
    Q:=P;
    P:=P->.NEXT;
    dispose(Q);
    out C_PORT.DISABLE_CLOCK;
    NEXT_FRAME_TO_SEND:=1-NEXT_FRAME_TO_SEND
  end;

initialize
  begin
    P:=nil;
    NEXT_FRAME_TO_SEND:=0;
    FRAME_EXPECTED:=0
  end;

trans when T_PORT.DATA
        begin
          PUTBUF(D)
        end;
      when M_PORT.MSG
```

```
                begin
                  if M.NS = FRAME_EXPECTED then
                    begin
                      out T_PORT.DATA(M.D);
                      FRAME_EXPECTED:=1-FRAME_EXPECTED
                    end;
                  if M.NR = NEXT_FRAME_TO_SEND then
                    GETBUF;
                  if P = nil then
                    out M_PORT.ACK(1-FRAME_EXPECTED)
                  else
                    SENDMSG
                end;
              when M_PORT.ACK
                begin
                  if NR = NEXT_FRAME_TO_SEND then
                    GETBUF;
                  if P <> nil then
                    SENDMSG
                end;
              when C_PORT.TIME_OUT
                begin
                  SENDMSG
                end;

    end EX_PROTOCOL;

   /*** instanciacao dos modulos em SIDE_REFINIMENT ***/

    C : CLOCK    with EX_CLOCK;
    T : TERMINAL with EX_TERMINAL;
    P : PROTOCOL with EX_PROTOCOL;

    connect T.PORT to P.T_PORT;
            C.PORT to P.C_PORT;

    replace PORT by P.M_PORT;

  end SIDE_REFINEMENT;

 /*** instanciacao dos modulos para SYSTEM_REFINEMENT ***/

  S0 : SIDE   with SIDE_REFINEMENT(false);
  S1 : SIDE   with SIDE_REFINEMENT(true);
  M  : MEDIUM with EX_MEDIUM;

  connect S0.PORT to M.PORT(.false.);
          S1.PORT to M.PORT(.true.);

end SYSTEM_REFINEMENT;
```

- A N E X O I -

Implementação do sistema apresentado no Anexo H

```
segment simula2;

const
  MAXSEQ = 1;
  MAXLIN = 80;

type
  DATA_TYPE =
    record
      IND : 0..MAXLIN;
      LINE : array (.1..MAXLIN.) of char
    end;
  SEQ_TYPE =
    0..MAXSEQ;
  MSG_TYPE =
    record
      NR,NS : SEQ_TYPE;
      D : DATA_TYPE
    end;
  BUFELPTR =
    ->BUFEL;
  BUFEL =
    record
      NEXT : BUFELPTR;
      INFO : DATA_TYPE
    end;
  channel =
    (C1MEDIUM_ACCESS_POINT,C2TERMINAL_ACCESS_POINT,
C3CLOCK_ACCESS_POINT,C0EX_CLOCK,C0EX_TERMINAL);
  S1MEDIUM_ACCESS_POINT =
    (S1MSG,S1ACK);
  S2TERMINAL_ACCESS_POINT =
    (S2DATA);
  S3CLOCK_ACCESS_POINT =
    (S3SET_CLOCK,S3DISABLE_CLOCK,S3TIME_OUT);
  S0EX_TERMINAL =
    (R1keyboard);
  S0EX_CLOCK =
    (R2ANY);
  S1TYPE =
    ->S0TYPE;
  S0TYPE =
    record
      NEXT : S1TYPE;
      case CHANNEL of
        C1MEDIUM_ACCESS_POINT :
          (case T1MEDIUM_ACCESS_POINT : S1MEDIUM_ACCESS_POINT of
            S1MSG :
              (D1MSG : record
                M : MSG_TYPE
              end;
              );
```

```
          S1ACK :
            (D1ACK : record
              NR : SEQ_TYPE
            end;
            );
        );
      C2TERMINAL_ACCESS_POINT :
       (case T2TERMINAL_ACCESS_POINT:S2TERMINAL_ACCESS_POINT of
          S2DATA :
            (D2DATA : record
              D : DATA_TYPE
            end;
            );
        );
      C3CLOCK_ACCESS_POINT :
        (case T3CLOCK_ACCESS_POINT : S3CLOCK_ACCESS_POINT of
          S3SET_CLOCK :
            (D3SET_CLOCK : record
              DELAY : integer
            end;
            );
          S3DISABLE_CLOCK :
            ();
          S3TIME_OUT :
            ();
        );
      COEX_CLOCK :
        (case TOEX_CLOCK : SOEX_CLOCK of
          R2ANY :
            ();
        );
      COEX_TERMINAL :
        (case TOEX_TERMINAL : SOEX_TERMINAL of
          R1keyboard :
            ();
        );
  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 =
    (P0EX_MEDIUM,P0EX_TERMINAL,P0EX_CLOCK,P0EX_PROTOCOL);
  P0TYPE =
    record
      IDENT : P2TYPE;
      CHANLIST : C1TYPE;
      NEXT : P1TYPE;
      case PROCESS of
        P0EX_MEDIUM :
          ();
        P0EX_TERMINAL :
          (D0EX_TERMINAL : record
            THIS_SIDE : boolean;
            CH : char;
            X : DATA_TYPE;
          end);
        P0EX_CLOCK :
          (D0EX_CLOCK : record
            state : (RUNNING,IDLE);
            THIS_SIDE : boolean;
          end);
        P0EX_PROTOCOL :
          (D0EX_PROTOCOL : record
            P : BUFELPTR;
            NEXT_FRAME_TO_SEND,FRAME_EXPECTED : SEQ_TYPE;
          end);
    end;

var
  P0VAR : P1TYPE;
  C0VAR : C1TYPE;
  S0VAR : S1TYPE;

function F0FUNCTION(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 : A0TYPE);
external;
procedure P7PROCEDURE(P : P1TYPE; var Q : P1TYPE);
external;
procedure P8PROCEDURE(P : P1TYPE);
external;
```

```
procedure GETCH(CHOICE : boolean; var CH : char);
external;
procedure WRITE_DATA(CHOICE : boolean; DATA : DATA_TYPE);
external;
procedure SETTIMER(CHOICE : boolean; DELAY : integer);
external;
procedure RESETTIMER(CHOICE : boolean);
external;
function TIMEOUT(CHOICE : boolean) : boolean;
external;
function FOBOOLEAN(I : integer) : boolean;
external;
function FOBOOLEAN ;
begin
  if I = 0 then
    FOBOOLEAN:=false
  else
    FOBOOLEAN:=true
end;

procedure EX_MEDIUM;
external;
procedure EX_MEDIUM;
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;
procedure P10PROCEDURE(B : boolean);
begin
  with S0VAR->.D1MSG do
    begin
      if not PERDA_NO_MEIO then
        begin
          P0PROCEDURE(1 );
          P4PROCEDURE(ord(not B ) ,0 );
          new(S0VAR ,C1MEDIUM_ACCESS_POINT ,S1MSG );
          S0VAR->.T1MEDIUM_ACCESS_POINT:=S1MSG ;
          S0VAR->.D1MSG.M:=M ;
          OUT
        end
      ;
      goto 1
    end;
end;
procedure P11PROCEDURE(B : boolean);
begin
  with S0VAR->.D1ACK do
    begin
      if not PERDA_NO_MEIO then
        begin
```

```
            P0PROCEDURE(1 );
            P4PROCEDURE(ord(not B ) ,0 );
            new(S0VAR ,C1MEDIUM_ACCESS_POINT ,S1ACK );
            S0VAR->.T1MEDIUM_ACCESS_POINT:=S1ACK ;
            S0VAR->.D1ACK.NR:=NR ;
            OUT
          end
        ;
        goto 1
      end;
end;
begin
  C1VAR:=C0VAR ;
  S1VAR:=S0VAR ;
  if C0VAR->.IDENT = 1 then
    if S0VAR->.T1MEDIUM_ACCESS_POINT = S1MSG then
      P10PROCEDURE(F0BOOLEAN(F0FUNCTION(0 ) ) )
  ;
  if C0VAR->.IDENT = 1 then
    if S0VAR->.T1MEDIUM_ACCESS_POINT = S1ACK then
      P11PROCEDURE(F0BOOLEAN(F0FUNCTION(0 ) ) )
  ;
  1 : case C1VAR->.IDENT of
    1 :
      case S1VAR->.T1MEDIUM_ACCESS_POINT of
        S1MSG :
          dispose(S1VAR ,C1MEDIUM_ACCESS_POINT ,S1MSG );
        S1ACK :
          dispose(S1VAR ,C1MEDIUM_ACCESS_POINT ,S1ACK );
      end;
  end;
end;

procedure EX_TERMINAL;
external;
procedure EX_TERMINAL;
label
  1;
var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
begin
  C1VAR:=C0VAR ;
  S1VAR:=S0VAR ;
  with P0VAR->.D0EX_TERMINAL do
    begin
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T2TERMINAL_ACCESS_POINT = S2DATA then
          with S0VAR->.D2DATA do
            begin
              WRITE_DATA(THIS_SIDE ,D );
              goto 1
            end
      ;
      if C0VAR->.IDENT = 0 then
        if S0VAR->.T0EX_TERMINAL = R1keyboard then
          begin
            GETCH(THIS_SIDE ,CH );
```

```
            if ord(CH ) > 0 then
              if CH in (.'A'..'Z','a'..'z','0'..'9',' '.) then
                begin
                  X.IND:=X.IND + 1 ;
                  X.LINE(.X.IND .):=CH ;
                  if X.IND >= MAXLIN then
                    begin
                      begin
                        POPROCEDURE(1 );
                       new(SOVAR,C2TERMINAL_ACCESS_POINT,S2DATA);
                        SOVAR->.T2TERMINAL_ACCESS_POINT:=S2DATA ;
                        SOVAR->.D2DATA.D:=X ;
                        OUT
                      end;
                      X.IND:=0
                    end
                end
              else
                if X.IND > 0 then
                  begin
                    begin
                      POPROCEDURE(1 );
                      new(SOVAR,C2TERMINAL_ACCESS_POINT,S2DATA);
                      SOVAR->.T2TERMINAL_ACCESS_POINT:=S2DATA ;
                      SOVAR->.D2DATA.D:=X ;
                      OUT
                    end;
                    X.IND:=0
                  end
            ;
            goto 1
          end
      ;
    end;
  1 : case C1VAR->.IDENT of
    1 :
      case S1VAR->.T2TERMINAL_ACCESS_POINT of
        S2DATA :
          dispose(S1VAR ,C2TERMINAL_ACCESS_POINT ,S2DATA );
      end;
    0 :
      case S1VAR->.T0EX_TERMINAL of
        R1keyboard :
          dispose(S1VAR ,C0EX_TERMINAL ,R1keyboard );
      end;
  end;
end;

procedure EX_CLOCK;
external;
procedure EX_CLOCK;
label
  1;
var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
begin
  C1VAR:=C0VAR ;
```

```
  S1VAR:=S0VAR ;
  with P0VAR->.D0EX_CLOCK do
    begin
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T3CLOCK_ACCESS_POINT = S3SET_CLOCK then
          with S0VAR->.D3SET_CLOCK do
            begin
              state:=RUNNING ;
              SETTIMER(THIS_SIDE ,DELAY );
              goto 1
            end
      ;
      if C0VAR->.IDENT = 1 then
        if S0VAR->.T3CLOCK_ACCESS_POINT = S3DISABLE_CLOCK then
          if state in (.RUNNING .) then
            begin
              state:=IDLE ;
              RESETTIMER(THIS_SIDE );
              goto 1
            end
      ;
      if C0VAR->.IDENT = 0 then
        if S0VAR->.T0EX_CLOCK = R2ANY then
          if TIMEOUT(THIS_SIDE ) then
            if state in (.RUNNING .) then
              begin
                state:=IDLE ;
                begin
                  P0PROCEDURE(1 );
                  new(S0VAR ,C3CLOCK_ACCESS_POINT ,S3TIME_OUT );
                  S0VAR->.T3CLOCK_ACCESS_POINT:=S3TIME_OUT ;
                  OUT
                end;
                goto 1
              end
      ;
    end;
  1 : case C1VAR->.IDENT of
    1 :
      case S1VAR->.T3CLOCK_ACCESS_POINT of
        S3SET_CLOCK :
          dispose(S1VAR ,C3CLOCK_ACCESS_POINT ,S3SET_CLOCK );
        S3DISABLE_CLOCK :
          dispose(S1VAR ,C3CLOCK_ACCESS_POINT ,S3DISABLE_CLOCK );
      end;
    0 :
      case S1VAR->.T0EX_CLOCK of
        R2ANY :
          dispose(S1VAR ,C0EX_CLOCK ,R2ANY );
      end;
  end;
end;

procedure EX_PROTOCOL;
external;
procedure EX_PROTOCOL;
label
  1;
```

```
var
  C1VAR : C1TYPE;
  S1VAR : S1TYPE;
procedure SENDMSG;
var
  X : MSG_TYPE;
begin
  with POVAR->.DOEX_PROTOCOL do
    begin
      X.NR:=1 - FRAME_EXPECTED ;
      X.NS:=NEXT_FRAME_TO_SEND ;
      X.D:=P->.INFO ;
      begin
        POPROCEDURE(3 );
        new(SOVAR ,C1MEDIUM_ACCESS_POINT ,S1MSG );
        SOVAR->.T1MEDIUM_ACCESS_POINT:=S1MSG ;
        SOVAR->.D1MSG.M:=X ;
        OUT
      end;
      begin
        POPROCEDURE(2 );
        new(SOVAR ,C3CLOCK_ACCESS_POINT ,S3SET_CLOCK );
        SOVAR->.T3CLOCK_ACCESS_POINT:=S3SET_CLOCK ;
        SOVAR->.D3SET_CLOCK.DELAY:=10 ;
        OUT
      end
    end
end;
procedure PUTBUF(D : DATA_TYPE);
var
  Q,R : BUFELPTR;
begin
  with POVAR->.DOEX_PROTOCOL do
    begin
      new(R );
      R->.INFO:=D ;
      R->.NEXT:=nil ;
      if P = nil then
        begin
          P:=R ;
          SENDMSG
        end
      else
        begin
          Q:=P ;
          while Q->.NEXT <> nil do
            Q:=Q->.NEXT ;
          Q->.NEXT:=R
        end
    end
end;
procedure GETBUF;
var
  Q : BUFELPTR;
begin
  with POVAR->.DOEX_PROTOCOL do
    begin
      Q:=P ;
```

```
      P:=P->.NEXT ;
      dispose(Q );
      begin
        POPROCEDURE(2 );
        new(SOVAR ,C3CLOCK_ACCESS_POINT ,S3DISABLE_CLOCK );
        SOVAR->.T3CLOCK_ACCESS_POINT:=S3DISABLE_CLOCK ;
        OUT
      end;
      NEXT_FRAME_TO_SEND:=1 - NEXT_FRAME_TO_SEND
    end
end;
begin
  C1VAR:=COVAR ;
  S1VAR:=SOVAR ;
  with POVAR->.DOEX_PROTOCOL do
    begin
      if COVAR->.IDENT = 1 then
        if SOVAR->.T2TERMINAL_ACCESS_POINT = S2DATA then
          with SOVAR->.D2DATA do
            begin
              PUTBUF(D );
              goto 1
            end
      ;
      if COVAR->.IDENT = 3 then
        if SOVAR->.T1MEDIUM_ACCESS_POINT = S1MSG then
          with SOVAR->.D1MSG do
            begin
              if M.NS = FRAME_EXPECTED then
                begin
                  begin
                    POPROCEDURE(1 );
                    new(SOVAR ,C2TERMINAL_ACCESS_POINT ,S2DATA );
                    SOVAR->.T2TERMINAL_ACCESS_POINT:=S2DATA ;
                    SOVAR->.D2DATA.D:=M.D ;
                    OUT
                  end;
                  FRAME_EXPECTED:=1 - FRAME_EXPECTED
                end
              ;
              if M.NR = NEXT_FRAME_TO_SEND then
                GETBUF
              ;
              if P = nil then
                begin
                  POPROCEDURE(3 );
                  new(SOVAR ,C1MEDIUM_ACCESS_POINT ,S1ACK );
                  SOVAR->.T1MEDIUM_ACCESS_POINT:=S1ACK ;
                  SOVAR->.D1ACK.NR:=1 - FRAME_EXPECTED ;
                  OUT
                end
              else
                SENDMSG;
              goto 1
            end
      ;
      if COVAR->.IDENT = 3 then
        if SOVAR->.T1MEDIUM_ACCESS_POINT = S1ACK then
```

```
          with S0VAR->.D1ACK do
            begin
              if NR = NEXT_FRAME_TO_SEND then
                GETBUF
              ;
              if P <> nil then
                SENDMSG
              ;
              goto 1
            end
        ;
        if C0VAR->.IDENT = 2 then
          if S0VAR->.T3CLOCK_ACCESS_POINT = S3TIME_OUT then
            begin
              SENDMSG;
              goto 1
            end
        ;
      end;
    1 : case C1VAR->.IDENT of
      1 :
        case S1VAR->.T2TERMINAL_ACCESS_POINT of
          S2DATA :
            dispose(S1VAR ,C2TERMINAL_ACCESS_POINT ,S2DATA );
        end;
      3 :
        case S1VAR->.T1MEDIUM_ACCESS_POINT of
          S1MSG :
            dispose(S1VAR ,C1MEDIUM_ACCESS_POINT ,S1MSG );
          S1ACK :
            dispose(S1VAR ,C1MEDIUM_ACCESS_POINT ,S1ACK );
        end;
      2 :
        case S1VAR->.T3CLOCK_ACCESS_POINT of
          S3TIME_OUT :
            dispose(S1VAR ,C3CLOCK_ACCESS_POINT ,S3TIME_OUT );
        end;
    end;
end;

procedure IOSYSTEM_REFINEMENT(var P1VAR : P1TYPE);
external;
procedure IOSYSTEM_REFINEMENT;
var
  P2VAR : P1TYPE;
  A0VAR,A1VAR : A0TYPE;
procedure IOEX_MEDIUM(var P1VAR : P1TYPE);
var
  I0VAR : integer;
begin
  new(P0VAR ,P0EX_MEDIUM );
  P1PROCEDURE('EX_MEDIUM ' ,P1VAR );
  for I0VAR:=0 to 1 do
    begin
      P3PROCEDURE(false ,1 );
      P2PROCEDURE(I0VAR );
    end;
end;
```

```
procedure IOSIDE_REFINEMENT(var P1VAR : P1TYPE;CHOICE : boolean);
var
  P2VAR : P1TYPE;
  A0VAR,A1VAR : A0TYPE;
procedure IOEX_TERMINAL(var P1VAR : P1TYPE);
begin
  new(P0VAR ,P0EX_TERMINAL );
  P1PROCEDURE('EX_TERMINA' ,P1VAR );
  P3PROCEDURE(false ,0 );
  P3PROCEDURE(false ,1 );
  with P0VAR->.D0EX_TERMINAL do
    begin
      THIS_SIDE:=CHOICE ;
      X.IND:=0
    end
end;
procedure IOEX_CLOCK(var P1VAR : P1TYPE);
begin
  new(P0VAR ,P0EX_CLOCK );
  P1PROCEDURE('EX_CLOCK  ' ,P1VAR );
  P3PROCEDURE(false ,0 );
  P3PROCEDURE(false ,1 );
  with P0VAR->.D0EX_CLOCK do
    begin
      state:=IDLE ;
      THIS_SIDE:=CHOICE
    end
end;
procedure IOEX_PROTOCOL(var P1VAR : P1TYPE);
begin
  new(P0VAR ,P0EX_PROTOCOL );
  P1PROCEDURE('EX_PROTOCO' ,P1VAR );
  P3PROCEDURE(true ,1 );
  P3PROCEDURE(false ,2 );
  P3PROCEDURE(true ,3 );
  with P0VAR->.D0EX_PROTOCOL do
    begin
      P:=nil ;
      NEXT_FRAME_TO_SEND:=0 ;
      FRAME_EXPECTED:=0
    end
end;
begin
  P2VAR:=nil ;
  IOEX_CLOCK(P2VAR );
  IOEX_TERMINAL(P2VAR );
  IOEX_PROTOCOL(P2VAR );
  P7PROCEDURE(P2VAR ,P1VAR );
  P3PROCEDURE(false ,1 );
  P5PROCEDURE(2 ,1 ,P1VAR );
  P6PROCEDURE(A0VAR );
  P5PROCEDURE(1 ,1 ,P1VAR );
  P6PROCEDURE(A1VAR );
  A0VAR.CHAN->.TARGET:=A1VAR ;
  A1VAR.CHAN->.TARGET:=A0VAR ;
  P5PROCEDURE(3 ,1 ,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(1 ,3 ,P1VAR );
    P6PROCEDURE(A0VAR.CHAN->.TARGET );
    P8PROCEDURE(P1VAR );
end;
begin
    P2VAR:=nil ;
    IOSIDE_REFINEMENT(P2VAR ,false );
    IOSIDE_REFINEMENT(P2VAR ,true );
    IOEX_MEDIUM(P2VAR );
    P7PROCEDURE(P2VAR ,P1VAR );
    P5PROCEDURE(3 ,1 ,P1VAR );
    P6PROCEDURE(A0VAR );
    P5PROCEDURE(1 ,1 ,P1VAR );
    P4PROCEDURE(ord(false ) ,0 );
    P6PROCEDURE(A1VAR );
    A0VAR.CHAN->.TARGET:=A1VAR ;
    A1VAR.CHAN->.TARGET:=A0VAR ;
    P5PROCEDURE(2 ,1 ,P1VAR );
    P6PROCEDURE(A0VAR );
    P5PROCEDURE(1 ,1 ,P1VAR );
    P4PROCEDURE(ord(true ) ,0 );
    P6PROCEDURE(A1VAR );
    A0VAR.CHAN->.TARGET:=A1VAR ;
    A1VAR.CHAN->.TARGET:=A0VAR ;
    P8PROCEDURE(P1VAR );
end;
```

- A N E X O J -

Núcleo de exploração para as implementações apresentadas no Anexos G e I

```
program NUCLEO12(input,output);

const
  IDENTLEN = 10;
  MAXLIN = 80;

type
  DATA_TYPE      = record
                     IND  : 0..MAXLIN;
                     LINE : packed array (.1..MAXLIN.) of char
                   end;
  CHANNEL        = (COEX_TERMINAL,COEX_CLOCK);
  SOEX_TERMINAL  = (R1KEYBOARD);                                  /**/
  SOEX_CLOCK     = (R2ANY);                                       /**/
  S1TYPE         = ->SOTYPE;                                      /**/
  SOTYPE         = record
                     NEXT : S1TYPE;
                     case CHANNEL of
                       COEX_TERMINAL :                           /**/
                         (case TOEX_TERMINAL:SOEX_TERMINAL of    /**/
                            R1KEYBOARD :                         /**/
                              ()                                 /**/
                         );                                      /**/
                       COEX_CLOCK :                              /**/
                         (case TOEX_CLOCK : SOEX_CLOCK 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;
  P0TYPE          = record
                      IDENT       : P2TYPE;
                      CHANLIST    : C1TYPE;
                      NEXT,
                      REFINEMENT  : P1TYPE
                    end;

var
  P0VAR           : P1TYPE;
  C0VAR           : C1TYPE;
  S0VAR           : S1TYPE;
  SIGNAL_PENDING  : integer;

  /*---{ variaveis auxiliares para a implementacao das
         primitivas de baixo nivel, inseridas aqui para
         facilitar suas implementacoes                          }---*/

  LINHA           : array (.boolean.) of                          /**/
                    packed array(.1..MAXLIN.) of char;            /**/
  COL,                                                            /**/
  CONT_TEMPO      : array (.boolean.) of integer;                 /**/
  ACABOU,                                                         /**/
  FIM             : array (.boolean.) of boolean;                 /**/
  ENTRADA1,                                                       /**/
  ENTRADA2,                                                       /**/
  SAIDA1,                                                         /**/
  SAIDA2          : text;                                         /**/


/*---{ implementacao das primitivas de baixo nivel              }---*/

procedure WRITE_DATA(THIS_SIDE : boolean; D : DATA_TYPE);         /**/
  external;                                                       /**/
                                                                  /**/
procedure WRITE_DATA;                                             /**/
                                                                  /**/
begin                                                             /**/
  if THIS_SIDE then                                               /**/
    writeln(SAIDA1,D.LINE:80)                                     /**/
  else                                                            /**/
    writeln(SAIDA2,D.LINE:80);                                    /**/
  write(clock:9,' usuario ');                                     /**/
  if THIS_SIDE then                                               /**/
    write('1 recebeu de 2')                                       /**/
  else                                                            /**/
    write('2 recebeu de 1');                                      /**/
  writeln(' a mensagem ',D.LINE:80)                               /**/
end;                                                              /**/
                                                                  /**/
procedure GETCH(THIS_SIDE : boolean; var CH : char);external;     /**/
                                                                  /**/
procedure GETCH;                                                  /**/
                                                                  /**/
var AUX : packed array(.1..MAXLIN.) of char;                      /**/
                                                                  /**/
begin                                                             /**/
  if FIM(.THIS_SIDE.) then                                        /**/
```

```
    begin                                                              /**/
      ACABOU(.THIS_SIDE.):=true;                                       /**/
      CH:=chr(0)                                                       /**/
    end                                                                /**/
  else                                                                 /**/
    begin                                                              /**/
      if COL(.THIS_SIDE.) = 80 then                                    /**/
        begin                                                          /**/
          if THIS_SIDE then                                            /**/
            begin                                                      /**/
              readln(ENTRADA1,LINHA(.true.));                          /**/
              if eof(ENTRADA1) then                                    /**/
                FIM(.true.):=true                                      /**/
            end                                                        /**/
          else                                                         /**/
            begin                                                      /**/
              readln(ENTRADA2,LINHA(.false.));                         /**/
              if eof(ENTRADA2) then                                    /**/
                FIM(.false.):=true                                     /**/
            end;                                                       /**/
          COL(.THIS_SIDE.):=0;                                         /**/
          write(clock:9,' usuario ');                                  /**/
          if THIS_SIDE then                                            /**/
            write('1 enviou para 2')                                   /**/
          else                                                         /**/
            write('2 enviou para 1');                                  /**/
          writeln(' a mensagem ',LINHA(.THIS_SIDE.))                   /**/
        end;                                                           /**/
      COL(.THIS_SIDE.):=COL(.THIS_SIDE.)+1;                            /**/
      AUX:=LINHA(.THIS_SIDE.);                                         /**/
      CH:=AUX(.COL(.THIS_SIDE.).)                                      /**/
    end                                                                /**/
end;                                                                   /**/
                                                                       /**/
procedure SETTIMER(THIS_SIDE : boolean; DELAY : integer);              /**/
  external;                                                            /**/
                                                                       /**/
procedure SETTIMER;                                                    /**/
                                                                       /**/
begin                                                                  /**/
  CONT_TEMPO(.THIS_SIDE.):=DELAY*10000+clock;                          /**/
  write(clock:9,' acionado temporizador de ');                         /**/
  if THIS_SIDE then                                                    /**/
    writeln('1 para ',CONT_TEMPO(.THIS_SIDE.):9)                       /**/
  else                                                                 /**/
    writeln('2 para ',CONT_TEMPO(.THIS_SIDE.):9)                       /**/
end;                                                                   /**/
                                                                       /**/
procedure RESETTIMER(THIS_SIDE : boolean); external;                   /**/
                                                                       /**/
procedure RESETTIMER;                                                  /**/
                                                                       /**/
begin                                                                  /**/
  CONT_TEMPO(.THIS_SIDE.):=0;                                          /**/
  write(clock:9,'desligado temporiz. para mensagem enviada');          /**/
  if THIS_SIDE then                                                    /**/
    writeln('por 1')                                                   /**/
  else                                                                 /**/
```

```
    writeln('por 2')                                                    /**/
end;                                                                    /**/
                                                                        /**/
function TIMEOUT(THIS_SIDE : boolean) : boolean; external;              /**/
                                                                        /**/
function TIMEOUT;                                                       /**/
                                                                        /**/
begin                                                                   /**/
  TIMEOUT:=true;                                                        /**/
  if (CONT_TEMPO(.THIS_SIDE.) = 0) or                                   /**/
     (CONT_TEMPO(.THIS_SIDE.) < clock) then                             /**/
    TIMEOUT:=false                                                      /**/
  else                                                                  /**/
    begin                                                               /**/
      write(clock:9,' TIMEOUT para mensagem enviada por ');             /**/
      if THIS_SIDE then                                                 /**/
        writeln('1')                                                    /**/
      else                                                              /**/
        writeln('2')                                                    /**/
    end                                                                 /**/
end;                                                                    /**/
                                                                        /**/
procedure IOSYSTEM_REFINEMENT(var P1VAR : P1TYPE);                      /**/
  external;                                                             /**/
                                                                        /**/
procedure EX_MEDIUM;                                                    /**/
  external;                                                             /**/
                                                                        /**/
procedure EX_TERMINAL;                                                  /**/
  external;                                                             /**/
                                                                        /**/
procedure EX_CLOCK;                                                     /**/
  external;                                                             /**/
                                                                        /**/
procedure EX_PROTOCOL;                                                  /**/
  external;                                                             /**/

/*---{ inicializacao das variaveis auxiliares                         }---*/

procedure INICIALIZACAO;                                                /**/
                                                                        /**/
begin                                                                   /**/
  FIM(.false.):=false;                                                  /**/
  FIM(.true.):=false;                                                   /**/
  ACABOU(.false.):=false;                                               /**/
  ACABOU(.true.):=false;                                                /**/
  COL(.false.):=80;                                                     /**/
  COL(.true.):=80;                                                      /**/
  reset(ENTRADA1);                                                      /**/
  reset(ENTRADA2);                                                      /**/
  rewrite(SAIDA1);                                                      /**/
  rewrite(SAIDA2);                                                      /**/
end;                                                                    /**/
                                                                        /**/
procedure OUT; external;
procedure OUT;

procedure RENDEZVOUS;
```

```
var
  SAVEPROCESS : P1TYPE;
  SAVECHANNEL : C1TYPE;

begin
  SAVEPROCESS:=POVAR;
  SAVECHANNEL:=COVAR;
  POVAR:=COVAR->.TARGET.PROC;
  COVAR:=COVAR->.TARGET.CHAN;
  if POVAR->.IDENT = 'EX_MEDIUM ' then                          /**/
    EX_MEDIUM                                                   /**/
  else                                                          /**/
    if POVAR->.IDENT = 'EX_TERMINA' then                        /**/
      EX_TERMINAL                                               /**/
    else                                                        /**/
      if POVAR->.IDENT = 'EX_CLOCK  ' then                      /**/
        EX_CLOCK                                                /**/
      else                                                      /**/
        if POVAR->.IDENT = 'EX_PROTOCO' then                    /**/
          EX_PROTOCOL;                                          /**/
  COVAR:=SAVECHANNEL;
  POVAR:=SAVEPROCESS
end;

begin
  with COVAR->.TARGET.CHAN-> do
    if QUEUED then
      begin
        SIGNAL_PENDING:=SIGNAL_PENDING+1;
        SOVAR->.NEXT:=HEAD;
        HEAD:=SOVAR
      end
    else
      RENDEZVOUS
end;

procedure SYSTEM_INIT;

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

begin
  P:=nil;
  IOSYSTEM_REFINEMENT(P);
  POVAR:=P->.REFINEMENT;
  dispose(P);
  P:=POVAR;
  SIGNAL_PENDING:=0;
  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
end;

procedure SCHEDULER;

procedure GETSIGNAL;

var
  FOUND : boolean;
  S     : S1TYPE;
```

```
begin
  FOUND:=false;
  SIGNAL_PENDING:=SIGNAL_PENDING-1;
  repeat
    while COVAR = nil do
      begin
        POVAR:=POVAR->.NEXT;
        COVAR:=POVAR->.CHANLIST
      end;
    if COVAR->.QUEUED then
      begin
        SOVAR:=COVAR->.HEAD;
        if SOVAR <> nil then
          begin
            FOUND:=true;
            S:=nil;
            while SOVAR->.NEXT <> nil do
              begin
                S:=SOVAR;
                SOVAR:=SOVAR->.NEXT
              end
          end
      end;
    if not FOUND then
      COVAR:=COVAR->.NEXT
  until FOUND;
  if S = nil then
    COVAR->.HEAD:=nil
  else
    S->.NEXT:=nil
end;

begin
  repeat
    if SIGNAL_PENDING > 0 then
      begin
        GETSIGNAL;
        if POVAR->.IDENT = 'EX_MEDIUM ' then                    /**/
          EX_MEDIUM                                             /**/
        else                                                    /**/
          if POVAR->.IDENT = 'EX_TERMINA' then                  /**/
            EX_TERMINAL                                         /**/
          else                                                  /**/
            if POVAR->.IDENT = 'EX_CLOCK  ' then                /**/
              EX_CLOCK                                          /**/
            else                                                /**/
              if POVAR->.IDENT = 'EX_PROTOCO' then              /**/
                EX_PROTOCOL                                     /**/
      end
    else
      begin
        POVAR:=POVAR->.NEXT;
        COVAR:=POVAR->.CHANLIST;
        if POVAR->.IDENT = 'EX_TERMINA' then                    /**/
          begin                                                 /**/
            new(SOVAR,COEX_TERMINAL,R1KEYBOARD);                /**/
            SOVAR->.TOEX_TERMINAL:=R1KEYBOARD;                  /**/
```

```
            EX_TERMINAL                                              /**/
          end                                                        /**/
        else                                                         /**/
          if POVAR->.IDENT = 'EX_CLOCK   ' then                      /**/
            begin                                                    /**/
              new(SOVAR,COEX_CLOCK,R2ANY);                           /**/
              SOVAR->.TOEX_CLOCK:=R2ANY;                             /**/
              EX_CLOCK                                               /**/
            end                                                      /**/
      end
  until ACABOU(.true.) and ACABOU(.false.)                           /**/
end;

begin
  INICIALIZACAO;                                                     /**/
  SYSTEM_INIT;
  SCHEDULER
end.
```

- A N E X O K -

Rotinas de suporte para a criação e manipulação das estruturas de dados

```
segment PFAUX;

const
  IDENTLEN = 10;

type
  S1TYPE          = ->S0TYPE;
  S0TYPE          = record
                    end;
  I1TYPE          = ->I0TYPE;
  C1TYPE          = ->C0TYPE;
  P1TYPE          = ->P0TYPE;
  I0TYPE          = record
                      NEXT  : I1TYPE;
                      IDENT : integer;
                    end;
  A0TYPE          = record
                      PROC : P1TYPE;
                      CHAN : C1TYPE;
                    end;
  C0TYPE          = record
                      IDENT   : integer;
                      INDLIST : I1TYPE;
                      TARGET  : A0TYPE;
                      NEXT    : C1TYPE;
                      case QUEUED : boolean of
                        false :
                          ();
                        true :
                          (HEAD : S1TYPE);
                    end;
  P2TYPE          = packed array (.1..10.) of char;
  P0TYPE          = record
                      IDENT      : P2TYPE;
                      CHANLIST   : C1TYPE;
                      NEXT,
                      REFINEMENT : P1TYPE
                    end;


var
  P0VAR           : P1TYPE;
  C0VAR           : C1TYPE;

function F0FUNCTION(INDPOS : integer) : integer; external;

/*---{ retorna o valor do indice inteiro para um dado
       posicao de indice                                    }---*/

function F0FUNCTION;
```

```
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;

/*---{ localiza o primeiro canal identificado pelo IDENT,
       na lista de canais                                    }---*/

procedure P0PROCEDURE;

begin
  COVAR:=P0VAR->.CHANLIST;
  while COVAR->.IDENT < IDENT do
    COVAR:=COVAR->.NEXT
end;

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

/*---{ inicializa o elemento de dados do processo com
       identificacao IDENT e insere-o na cabeca P da
       lista de processos                                    }---*/

procedure P1PROCEDURE;

begin
  P0VAR->.IDENT:=IDENT;
  P0VAR->.CHANLIST:=nil;
  P0VAR->.NEXT:=P;
  P:=P0VAR;
  COVAR:=nil
end;

procedure P2PROCEDURE(IDENT : integer); external;

/*---{ gera elemento com valor de indice IDENT na cabeca
       da lista de indices                                   }---*/

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;

/*---{ gera elemento com identificacao IDENT e opcao de
       fila Q na lista de canais                          }---*/

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;

/*---{ localiza a porta com valor de indice INDVAL na
       posicao de indice INDPOS (Obs.: INDVAL fora do
       limite ira' causar um erro em tempo de execucao)   }---*/

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;

/*---{ localiza o J-esimo canal da I-esima ocorrencia do
       modulo (na ordem inversa) especificado no corpo do
       refinamento cuja cabeca temporaria e' apontada por
       P (Obs.: Se I <= 0, entao o J-esimo canal do modulo
       de refinamento e' acessado)                          }---*/

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;

/*---{ faz com que X acesse uma porta para que ele possa
       ser conectado a outra porta para formar um canal     }---*/

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;

/*---{ gera e inicializa um cabecalho temporario de
       refinamento, com cabeca Q, da lista de processos;
       P aponta para o primeiro elemento de dados do
       processo local ao refinamento dado                    }---*/

procedure P7PROCEDURE;

begin
  new(POVAR);
  P1PROCEDURE('REFINEMENT',Q);
  POVAR->.REFINEMENT:=P
end;

procedure P8PROCEDURE(P : P1TYPE); external;

/*---{ elimina as interfaces temporarias de refinamento
       (isto e', lista de cabecas e lista de canais          }---*/

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;
```

- A N E X O L -

Resultado da análise da simulação com meio ideal

```
*** Dangling connections ***

process          channel  index
seqnr/id          seqnr   intval
 1/EX_MEDIUM        1      (0)
 1/EX_MEDIUM        1      (1)
 2/EX_PROTOCO       1
 2/EX_PROTOCO       2
 2/EX_PROTOCO       3
 3/EX_TERMINA       1
 4/EX_CLOCK         1
 5/EX_PROTOCO       1
 5/EX_PROTOCO       2
 5/EX_PROTOCO       3
 6/EX_TERMINA       1
 7/EX_CLOCK         1

-------------------------------------------

427486472 usuario 1 enviou para 2 a mensagem registro 1a
427491328 usuario 2 enviou para 1 a mensagem registro 2a
427659340 acionado temporizador de 1 para 427759238
427665691 usuario 2 recebeu de 1 a mensagem registro 1a
427670645 desligado temporizador para mensagem enviada por 1
427674959 usuario 1 enviou para 2 a mensagem registro 1b
427682772 acionado temporizador de 2 para 427782672
427689146 usuario 1 recebeu de 2 a mensagem registro 2a
427694006 desligado temporizador para mensagem enviada por 2
427698426 usuario 2 enviou para 1 a mensagem registro 2b
427866434 acionado temporizador de 1 para 427966328
427871421 usuario 2 recebeu de 1 a mensagem registro 1b
427876381 desligado temporizador para mensagem enviada por 1
427880728 usuario 1 enviou para 2 a mensagem registro 1c
427886868 acionado temporizador de 2 para 427986766
427892078 usuario 1 recebeu de 2 a mensagem registro 2b
427896920 desligado temporizador para mensagem enviada por 2
427901339 usuario 2 enviou para 1 a mensagem registro 2c
428070663 acionado temporizador de 1 para 428170561
428075683 usuario 2 recebeu de 1 a mensagem registro 1c
428080635 desligado temporizador para mensagem enviada por 1
428085076 usuario 1 enviou para 2 a mensagem registro 1d
428091250 acionado temporizador de 2 para 428191147
428096276 usuario 1 recebeu de 2 a mensagem registro 2c
428101152 desligado temporizador para mensagem enviada por 2
428105538 usuario 2 enviou para 1 a mensagem registro 2d
428298072 acionado temporizador de 1 para 428397968
428303183 usuario 2 recebeu de 1 a mensagem registro 1d
428308136 desligado temporizador para mensagem enviada por 1
428313895 usuario 1 enviou para 2 a mensagem registro 1e
428320107 acionado temporizador de 2 para 428420001
428325184 usuario 1 recebeu de 2 a mensagem registro 2d
```

```
428330057 desligado temporizador para mensagem enviada por 2
428334631 usuario 2 enviou para 1 a mensagem registro 2e
428502850 acionado temporizador de 1 para 428602752
428508016 usuario 2 recebeu de 1 a mensagem registro 1e
428512964 desligado temporizador para mensagem enviada por 1
428517371 usuario 1 enviou para 2 a mensagem
428523584 acionado temporizador de 2 para 428623476
428528622 usuario 1 recebeu de 2 a mensagem registro 2e
428534639 desligado temporizador para mensagem enviada por 2
428708960 usuario 2 enviou para 2 a mensagem
```

- A N E X O M -

Resultado da análise da simulação com meio com perdas

```
*** Dangling connections ***

process        channel index
seqnr/id        seqnr  intval
 1/EX_MEDIUM       1    (0)
 1/EX_MEDIUM       1    (1)
 2/EX_PROTOCO      1
 2/EX_PROTOCO      2
 2/EX_PROTOCO      3
 3/EX_TERMINA      1
 4/EX_CLOCK        1
 5/EX_PROTOCO      1
 5/EX_PROTOCO      2
 5/EX_PROTOCO      3
 6/EX_TERMINA      1
 7/EX_CLOCK        1

-------------------------------------------

629556418 usuario 1 enviou para 2 a mensagem registro 1a
629561206 usuario 2 enviou para 1 a mensagem registro 2a
629728632 acionado temporizador de 1 para 629828532
629735311 usuario 2 recebeu de 1 a mensagem registro 1a
629740735 desligado temporizador para mensagem enviada por 1
629745033 usuario 1 enviou para 2 a mensagem registro 1b
629753309 acionado temporizador de 2 para 629853189
629759657 usuario 1 recebeu de 2 a mensagem registro 2a
629766598 usuario 2 enviou para 1 a mensagem registro 2b
629770347 TIMEOUT para mensagem enviada por 2
629774958 acionado temporizador de 2 para 629874854
629780714 desligado temporizador para mensagem enviada por 2
629946198 acionado temporizador de 1 para 630046104
629951149 usuario 2 recebeu de 1 a mensagem registro 1b
629956627 desligado temporizador para mensagem enviada por 1
629960966 usuario 1 enviou para 2 a mensagem registro 1c
629967593 acionado temporizador de 2 para 630067487
629972560 usuario 1 recebeu de 2 a mensagem registro 2b
629978946 desligado temporizador para mensagem enviada por 2
629983303 usuario 2 enviou para 1 a mensagem registro 2c
630150620 acionado temporizador de 1 para 630250518
630155551 usuario 2 recebeu de 1 a mensagem registro 1c
630163029 acionado temporizador de 2 para 630262930
630168024 usuario 1 recebeu de 2 a mensagem registro 2c
630171240 desligado temporizador para mensagem enviada por 1
630176220 desligado temporizador para mensagem enviada por 2
630180492 usuario 2 enviou para 1 a mensagem registro 2d
630185307 usuario 1 enviou para 2 a mensagem registro 1d
630352619 acionado temporizador de 2 para 630452522
630357596 usuario 1 recebeu de 2 a mensagem registro 2d
630363999 desligado temporizador para mensagem enviada por 2
630368343 usuario 2 enviou para 1 a mensagem registro 2e
```

```
630379106 usuario 1 enviou para 2 a mensagem registro 1e
630382242 TIMEOUT para mensagem enviada por 1
630386795 acionado temporizador de 1 para 630486673
630391802 usuario 2 recebeu de 1 a mensagem registro 1d
630397084 desligado temporizador para mensagem enviada por 1
630563541 acionado temporizador de 1 para 630663436
630568466 usuario 2 recebeu de 1 a mensagem registro 1e
630573893 desligado temporizador para mensagem enviada por 1
630578198 usuario 1 enviou para 2 a mensagem
630585988 acionado temporizador de 2 para 630685890
630591028 usuario 1 recebeu de 2 a mensagem registro 2e
630596300 desligado temporizador para mensagem enviada por 2
630600601 usuario 2 enviou para 1 a mensagem
```

- A N E X O N -

Manual para a utilização do Compilador Estelle

PASSO 01 : editar um arquivo com a especificação:

```
xedit <nome> estelle a
```

Obs.: Nesse arquivo, as palavras reservadas devem ser escritas necessariamente em letras minúsculas (**set case mixed**).

PASSO 02 : aumentar a área de trabalho para a configuração máxima permitida ao usuário (por exemplo, 3 Mb):

```
def stor 3m
i cms
```

PASSO 03 : entrar no ambiente Waterloo-Prolog:

```
prolog 2600
```

(2600 é, aproximadamente, o tamanho, em Kb, da área de trabalho acessada, pelo Waterloo-Prolog, para uma máquina dom 3 Mb de memória disponivel)

PASSO 04 : carregar o compilador Estelle:

```
load(estelle).
```

Aparecerá, então, uma mensagem

```
<<<<< PROGRAMA SENDO CARREGADO >>>>>
```

e o número de módulos do programa que ainda faltam serem carregados.

PASSO 05 : Um pedido na forma

```
entre com o nome do arquivo fonte na forma <nome>"."
```

surgirá para que o nome do arquivo com a especificação a ser considerada seja definido. Caso o arquivo indicado não conste do diretorio, uma mensagem

```
ARQUIVO NAO EXISTE
```

será impressa e o processo terminará.

PASSO 06 : A cada linha da especificação já processada, esta será mostrada na tela. Caso haja algum erro léxico, sintático ou semântico, o processo para após a impressão da mensagem de erro. Um arquivo **trace** estará disponivel com a listagem da compilação.

PASSO 07 : Uma vez que a especificação esteja livre de erros, um pedido na forma

entre com o nome do arquivo de saida na forma ‹nome›"."

informará ao sistema o nome do programa Pascal correspondente à especificação considerada a ser gerado.

PASSO 08 : Aparecerá então a mensagem

‹‹‹‹‹ CODIGO SENDO GERADO ›››››

junto com os passos que ainda faltam para que o código esteja pronto.

PASSO 09 : A mensagem

‹‹‹‹‹ CODIGO SENDO GRAVADO ›››

surgirá seguido do número de passos restantes para que o programa Pascal esteja completamente criado. A mensagem

‹‹‹‹‹ CODIGO SENDO COMPILADO ›››››

Nesta fase, se houver erros léxico ou de contexto, nas partes da especificação com construções próprias do Pascal, eles serão acusados. Ao fim dessa etapa, o processo de compilação termina.

PASSO 10 : Acrescentar, no programa **NUCLEOUS**, as estruturas de dados, as primitivas e os trechos de rotinas necessários para a execução da simulação do protocolo sendo considerado.

PASSO 11 : Executar a sequência de instruções

pascmod ‹nome› **nucleous pfaux**
start

para se executar a simulação do protocolo. O arquivo **OUTPUT** conterá o resultado da inicialização das estruturas de dados do protocolo.