DCA — FEE — UNICAMP

A Distribuição do Ambiente para Desenvolvimento de Software Tempo Real — STER

Eliane Gomes Guimarães

Novembro de 1990

Orientador : Prof. Dr. Maurício Ferreira Magalhães

Dissertação apresentada como requisito parcial para obtenção do Título de Mestre em Engenharia Elétrica pela Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas.

Este trabalho é dedicado a
Mariza e Geraldo (in memoriam),
meus pais.

Agradecimentos

Ao Prof. Dr. Maurício Ferreira Magalhães, pela possibilidade de participar do projeto que deu origem a este trabalho. Agradeço a sua orientação, incentivo, apoio, críticas construtivas e dedicação com que acompanhou o desenvolvimento deste trabalho.

A Mário Bento de Carvalho, Diretor do Instituto de Automação do Centro Tecnológico para Informática, Rubens Campos Machado, Chefe do Departamento de Controle de Processos do IA e Paulo César Berardi, Chefe da Divisão de Processos, pela possibilidade de concluir este trabalho.

Ao colega Juan Manuel Adán Coello, Chefe da Divisão de Programação de Tempo Real, pelas críticas construtivas, orientação, amizade, pela implementação do núcleo de tempo real da versão centralizada do STER e pelo auxílio no desenvolvimento da camada do Serviço de Comunicação de Rede Local.

Ao colega e grande amigo Adilson Barboza Lopes, pela constante colaboração, apoio, incentivo e suporte técnico, durante todo o desenvolvimento deste trabalho. Em particular, pela implementação das linguagens LPM e LCM das versões centralizada e distribuída.

Ao pessoal do Departamento de Controle de Processos do IA: Ailton, Ana, Aqueo, Beiral, Cidinha, Clarinda (UDD), Edeneziano, Gláucia, Hélio, Juan, Juziani, Koyama, Lázaro, Leonardo, Marcelo, Maurício, Olga, Roberto e Reinaldo.

Ao pessoal da Divisão de Robótica: Alfredo, Jorge, José Paulo, Josué, Othon e Ralph.

A Francisco Exner Neto, pela dedicação com que realizou os trabalhos de ilustração desta dissertação.

A Luiz Henrique Mansano, pelas valiosas contribuições na diagramação deste texto.

A Takao Suguiy, pelo auxílio na preparação do acabamento desta dissertação.

A Jodyr Wagner e a Maria Inês Lucena pelo apoio e incentivo.

Em especial, a Wilson Gomes de Moura, por incentivar e apoiar o meu reinício ao mestrado.

Resumo

Este trabalho apresenta a implementação de um ambiente para programação distribuída de tempo real baseado na troca de mensagens — STER. O ambiente destina-se, fundamentalmente, a aplicações na área de sistemas distribuídos em automação industrial. O STER proporciona uma abordagem baseada em linguagem para a programação de aplicações distribuídas, diferente de alguns sistemas existentes que proporcionam somente a conexão de sistemas autônomos. A abordagem utilizada reduz a complexidade na construção de aplicações distribuídas proporcionando modularidade, concorrência, comunicação e sincronização, integradas na estrutura de uma única linguagem. As suas principais características traduzem-se em: programação modular; independência dos módulos da aplicação relativamente ao fato da arquitetura alvo ser centralizada ou distribuída; possibilidades de reconfiguração dinâmica e de tolerância a falhas.

O STER proporciona um conjunto de ferramentas para a compilação, configuração, depuração e execução de programas em um ambiente distribuído. O desenvolvimento de um programa de aplicação divide-se em duas etapas: a programação dos módulos que implementam as funções do sistema e a configuração da aplicação a partir dos módulos disponíveis. O ambiente consiste basicamente de uma metodologia; de um Núcleo Operacional Distribuído Tempo Real (NOD); e de duas linguagens: a Linguagem de Programação de Módulos (LPM) e a Linguagem de Configuração de Módulos (LCM). A LCM é utilizada para especificar a configuração de componentes de software em estações lógicas, o que proporciona uma descrição de configuração concisa e facilita o reuso dos módulos componentes em configurações diferentes. As aplicações são construídas a partir de uma ou mais estações lógicas interconectadas. A versão implementada permite a execução de aplicações distribuídas em equipamentos compatíveis com o IBM-PC, interligados por rede local. Atualmente, o ambiente suporta apenas configuração estática das aplicações. Uma evolução do trabalho é a extensão do ambiente de forma a suportar a reconfiguração dinâmica e o transporte do STER para estações de trabalho executando sistemas UNIX.

Abstract

This work presents the implementation of the environment to distributed real time programming based on message passing — STER. The main application area of the environment is Distributed Systems in Industrial Automation. The STER provides a language-based approach to build distributed applications, unlike of existing systems which merely provide interconnection of autonomous systems. The approach used reduce the complexity of constructing distributed applications by providing modularity, concurrency, synchronisation and communication facilities integrated into a single language. The characteristics of this environment are: modular programming, applications modules independent of the target hardware architecture (centralized or distributed); possibilities of dynamic reconfiguration and fault-tolerance.

The STER provides a set of tools for program compilation, configuration, debugging and execution in a distributed environment. The development of an application program consists of two phases: the programming of modules that implement system functions and the configuration of the application using available modules. This environment integrates a design methodology; a Real Time Distributed Kernel (NOD) and the two languages: the Modules Programming Language (LPM) and the Modules Configuration Language (LCM). The LCM is employed to specify the configuration of software components into logical stations. This provides a concise configuration and facilitates the re-use of modules components in different configurations. Aplications are constructed as sets of one or more interconnected logical stations. The version implemented allows distributed applications running on IBM-PC compatible equipaments, interconnected via local networks. Currently, the environment supports only static configuration of the applications. Future steps related to the continuity of this work should include the support of dynamic reconfiguration and the transport of STER's tools to UNIX operating systems based workstations.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

As vantagens do uso de Sistemas Distribuídos (SDs) em termos de desempenho, confiabilidade, extensibilidade e disponibilidade são bem retratadas em diversos artigos [18,37,38]. Essas vantagens são conhecidas e aceitas, apesar de ainda não existir um consenso em como proporcionar o suporte necessário à modularidade, concorrência, comunicação, sincronização e configuração, inerentes a estes sistemas.

As ferramentas disponíveis, atualmente, para o projeto e implementação de software, são geralmente orientadas a sistemas com memória compartilhada. Essas ferramentas são inadequadas ao desenvolvimento de programas constituídos por tarefas remotas, que se comunicam por troca de mensagens, através de um meio de comunicação sujeito a erros e que impõe atrasos na troca destas informações. Dessa forma, as vantagens que podem ser obtidas com o uso de SDs não são gratuitas e demandam um ambiente adequado ao seu projeto e implementação.

Os SDs atuais variam entre aqueles que fazem adaptação e conexão dos diversos sistemas autônomos, através de uma abordagem ao nível do sistema operacional, e aqueles que proporcionam um ambiente baseado em uma linguagem completa para a programação das aplicações distribuídas.

Essa dissertação se insere neste último contexto, proporcionando uma abordagem orientada à linguagem na construção de aplicações distribuídas de controle de processos em tempo real. A abordagem utilizada reduz a complexidade, durante a construção de aplicações distribuídas, proporcionando facilidades de modularidade, concorrência, comunicação e sincronização, integradas na estrutura de uma única linguagem.

O objetivo dessa dissertação é a elaboração e implementação da distribuição de um ambiente para o desenvolvimento de software de tempo real - STER.

Devido às características dos SDs, a programação de uma aplicação envolve um alto grau de concorrência e paralelismo. Nesse sentido, foi implementado o ambiente STER, que atende aos requisitos essenciais para a programação de aplicações distribuídas em controle de processos e automação industrial, incluindo: multi-processamento; multi-programação; processamento concorrente com suporte em tempo real; capacidade de reconfiguração dinâmica e de tolerância a falhas.

O ambiente STER foi desenvolvido no Instituto de Automação do Centro Tecnológico para Informática - CTI em cooperação com a Universidade Estadual de Campinas - UNICAMP, e consiste de um modelo e de duas linguagens : a Linguagem de Programação de Módulos - LPM e a Linguagem de Configuração de Módulos - LCM. O modelo para programação distribuída de tempo real é baseado na troca de mensagens e foi elaborado levando em conta os principais aspectos dos sistemas distribuídos,

tais como: programação modular; eficiência do processo de comunicação e sincronização; forte tipificação dos dados; possibilidades de reconfiguração dinâmica e de tolerância a falhas.

Uma versão centralizada do ambiente STER tornou-se disponível em 1986 através de duas teses de Mestrado [1,43]. O STER tem sido utilizado no CTI e na UNICAMP em aplicações do seguinte tipo: simulador de células flexíveis, simulador de elevadores, simulador de metrô, desenvolvimento de um kit de laboratório de controle de processos, prototipagem de especificações em LOTOS e suporte para sistemas "Hard Real Time". O STER em suas duas versões, centralizada e distribuída, foi utilizado, também, como ferramenta de desenvolvimento durante o laboratório de Software Tempo Real realizado na IV EBAI (janeiro/89 - Argentina), onde pode ser constatada a facilidade de utilização do ambiente e assimilação do modelo de programação associado, pelos alunos participantes do laboratório.

Este trabalho está estruturado em sete capítulos e quatro apêndices.

O capítulo 2 apresenta uma visão geral dos SDs e do ambiente STER.

O capítulo 3 apresenta o modelo de arquitetura de software do SD implementado.

O capítulo 4 apresenta a sintaxe e semântica da linguagem de configuração de módulos.

O capítulo 5 apresenta a implementação do núcleo operacional distribuído de tempo real, discutindo-se em detalhes a infraestrutura necessária ao suporte das linguagens, dos serviços e da arquitetura distribuída, constituída de processadores autônomos homogêneos.

O capítulo 6 apresenta um exemplo de monitoramento de pacientes em um hospital , de modo a ilustrar o uso do ambiente para aplicações distribuídas. O pseudo código dos módulos componentes do exemplo constando das suas interfaces é apresentado. O corpo dos módulos, escrito em LPM, encontra-se disponível no DTIA STER (versão distribuída) — IA/CTI.

O capítulo 7 apresenta as conclusões e sugestões para futuras pesquisas.

O Apêndice A apresenta uma descrição BNF completa das linguagens LPM e LCM.

O Apêndice B apresenta os serviços oferecidos pelo STRE (Suporte de Tempo Real) da versão centralizada, com algumas extensões incluídas na versão distribuída.

O Apêndice C apresenta uma especificação formal dos módulos do SCR (Subsistema de Comunicação Remota entre Módulos) e dos módulos da CSCRL (Camada do Serviço de Comunicação de Rede Local).

O Apêndice D apresenta a CSCRL.

Capítulo 2

Sistemas Distribuídos

A evolução tecnológica dos microprocessadores e das comunicações tem contribuído para um crescente interesse na utilização de SDs. Atualmente, os fatores tecnológicos e econômicos tornam os SDs atrativos e eficientes para uma grande quantidade de aplicações. Os microprocessadores têm se tornado muito poderosos e baratos quando comparados com minicomputadores e computadores de maior porte, enquanto que os meios de comunicação permitem que tais processadores possam ser integrados em um ambiente de computação unificado. Desse modo, torna-se mais atrativo pensar-se em sistemas compostos de muitos computadores pequenos. Esses sistemas distribuídos têm vantagens no que diz respeito a preço/desempenho em relação aos sistemas tradicionais.

As vantagens potenciais da distribuição incluem:

- Melhora do desempenho através da exploração do paralelismo;
- Aumento da disponibilidade e confiabilidade através da exploração da redundância;
- Dispersão do poder de computação para localidades onde é necessário;
- Facilidade de crescimento através da adição de processadores e sistemas de comunicação.

Os sistemas distribuídos podem ser vistos em termos de três componentes funcionais principais:

- Sistemas Aplicativos;
- Sistemas Operacionais e Software de Suporte de Sistemas;
- Hardware.

Os sistemas aplicativos tornam-se mais complexos e exigem a integração de um conjunto de tarefas heterogêneas, muitas vezes com adição de requisitos e desempenho para aplicações onde a definição do fator-tempo para sistemas tempo real é necessária, em particular, no campo da informática industrial, tanto em controle de processos, como em automação da manufatura. A rápida evolução desse setor e a sua importância econômico-estratégica na indústria, contribuem para o surgimento dos sistemas distribuídos de tempo real, onde inúmeros trabalhos de pesquisa e desenvolvimento têm sido elaborados. As novas aplicações, por exemplo, visando as fábricas automatizadas do futuro, apresentam um grau elevado de complexidade e um sistema formado por um único processador não conseguiria realizar, adequadamente, todas as tarefas necessárias.

Atualmente, têm surgido novas arquiteturas de software e hardware para explorar todo o potencial dos SDs. Embora o custo do hardware esteja decrescendo rapidamente, o desenvolvimento do software

executando em um hardware distribuído ainda exige grandes investimentos. Esses investimentos devem-se ao esforço de programação requerido, pois o software precisa ser projetado cuidadosamente para explorar as vantagens potenciais do hardware distribuído, enquanto trata com os problemas causados por atrasos e falhas na comunicação entre estações, com o gerenciamento do paralelismo, com a comunicação entre processos, com o controle descentralizado, etc. Como essas questões não se apresentam em aplicações centralizadas, poucas experiências e metodologias existem acumuladas.

2.1 Taxonomias de Sistemas Distribuídos

O conceito de SD ainda não apresenta um consenso geral de definição sendo, muitas vezes, indevidamente considerado como sinônimo de redes de computadores.

Um SD, para os propósitos deste trabalho, é definido como um sistema de múltiplos elementos autônomos de processamento cooperando para alcançar um objetivo comum. Essa definição exclui máquinas "pipeline" e "array processors", cujos elementos não são autônomos; são também excluídas as redes de computadores, por exemplo a rede ARPANET, cujos nós não têm nenhum propósito comum. A maioria das aplicações que poderiam ser embutidas em arquiteturas de multiprocessadores, como controle de processos, aplicações comerciais orientadas a transações e aquisição de dados, adaptam-se à definição acima.

De uma maneira genérica, os SDs podem ser classificados como:

- Fortemente acoplados, onde os elementos de processamento, ou nós, têm acesso a uma memória comum. A comunicação e sincronização, é geralmente efetuada através de técnicas baseadas no uso de memória compartilhada;
- Fracamente acoplados, onde os elementos de processamento não têm acesso à memória comum. A comunicação e sincronização é realizada através de troca de mensagens.

Nesse capítulo, o termo SD refere-se a SDs fracamente acoplados. Maiores informações podem ser obtidas em [63].

Os SDs podem ser classificados segundo várias taxonomias diferentes. Nos próximos itens, examinaremos algumas taxonomias existentes.

2.1.1 Taxonomia quanto à variedade de processadores no sistema

Segundo esse critério, os SDs apresentam-se como sistemas homogêneos e sistemas heterogêneos.

Um sistema homogêneo é um sistema em que todos os processadores são do mesmo tipo.

Um sistema heterogêneo é um sistema que contém processadores de tipos diferentes. Uma discussão da transmissão de dados em sistemas heterogêneos é encontrada em [27].

2.1.2 Taxonomia quanto à abordagem de SDs baseados em SOs ou Linguagens

Muitas vezes, uma aplicação distribuída é implementada como uma coleção de programas seqüenciais que se comunicam através de chamadas ao sistema de comunicação da rede. Geralmente, essas interfaces são complexas e difíceis de usar. Pouca ou nenhuma checagem de erros, dessa interface, é proporcionada para assegurar a compatibilidade dos programas interconectados. As aplicações se tornam

mais difíceis para construir, depurar e manter. Os sistemas que implementam essa aproximação ao sistema operacional tendem a suportar flexibilidade e pronto acesso às facilidades do sistema, mas padecem por serem muito complexos para usar [57]. Exemplos dessa aproximação são a interface SNA LU 6.2 em SOs IBM [58], a interface DECNET NSP em SOs DEC [67] e a interface com protocolos TCP/IP, na maioria dos SOs UNIX [9].

Os SDs que fazem a sua implementação com uma aproximação baseada na linguagem de programação distribuída reduzem a complexidade durante a construção de aplicações distribuídas, proporcionando facilidades de modularidade, concorrência, comunicação e sincronização integradas na estrutura de uma única linguagem. Esses sistemas fornecem suporte para checagens de erro, em tempo de compilação, de ligação e execução da aplicação, de maneira a assegurar a compatibilidade de operação e troca de mensagem entre os componentes. Sistemas com essa abordagem também proporcionam comunicação e sincronização para as interações locais e remotas. Desse modo, os ambientes orientados a linguagens, geralmente são mais simples de usar e podem proporcionar ambientes mais seguros. Exemplos dessa aproximação são o ambiente CONIC [36], a linguagem de programação SR [4], a linguagem EMERALD [8] e a linguagem NIL [61].

2.1.3 Taxonomia quanto às bases de tempo dos processadores

Segundo esse critério os SDs apresentam-se como síncronos ou assíncronos [2].

Em um SD síncrono, todos os processadores do sistema funcionam segundo um relógio global. Esses sistemas constituem apenas uma abstração teórica. Não existem, na prática, mas são de grande importância no entendimento na área de processamento distribuído em geral.

Um SD assíncrono é um sistema real. Esses sistemas têm, como característica principal, uma ausência completa de uma base de tempo comum a todos os processadores que o compõem. Cada processador pertencente ao sistema possui um relógio local e independente dos demais. Além desse fato, geralmente, os sistemas assíncronos são modelados de tal forma que mensagens sofram atrasos finitos, porém indeterminados durante seu trânsito entre dois vizinhos no sistema.

2.1.4 Outras Taxonomias

Uma classificação de sistemas distribuídos que se enquadra no contexto do ambiente distribuído desenvolvido neste trabalho, é aquela definida por Cardozo [15]. Essa taxonomia compreende quatro dimensões:

- Complexidade de Módulo
  - — A complexidade de um módulo pode ser simplificada ou não no processo de decomposição de um sistema em módulos. A principal consideração na decomposição é quanto à natureza assíncrona das funções dentro de um módulo. Os critérios consistem na dependência de E/S, funções com tempo de resposta crítico, coesão, execução periódica.
- Granularidade de Módulo
  - — A independência de módulos pode ser qualitativamente avaliada segundo critérios de coesão e acoplamento. Coesão é a medida de unidade funcional de um módulo, isto é, um módulo altamente coeso deve desempenhar apenas (idealmente) uma função. Acoplamento é a medida da interdependência relativa entre módulos. É desejável decompor o sistema em módulos com alta coesão e baixo acoplamento. Um SD que tem alta

coesão e baixo acoplamento tem alta granularidade, o que implica em um número pequeno de comunicação.

- Heterogeneidade de Módulo

  — Reflete a uniformidade do SD em termos das tarefas que os módulos executam. Um extremo de uniformidade é um SD onde todos os módulos executam exatamente a mesma tarefa (correspondente a uma baixa heterogeneidade).

- Generalidade do HARDWARE

  — Denota o hardware exigido para as aplicações de controle de processos em tempo real, onde o desenvolvimento da aplicação é independente da estrutura do hardware em que será implementado.

Uma classificação onde apresenta-se um consenso geral dos SDs, segundo [47], consta da associação dos seguintes atributos:

- A arquitetua física é formada por uma coleção de estações autônomas (genericamente pares CPU + memória + periferia local);
- Os serviços que o SD coloca à disposição dos usuários externos são executados através de cooperação mútua de diferentes programas fracamente acoplados, em geral residentes em estações distintas;
- A comunicação entre processos remotos cooperantes é feita através de mensagens;
- Os detalhes da distribuição física não são relevantes para os usuários do SD ("location transparency"). O ambiente será transparente ao usuário, que deverá visualizar o sistema como se fosse uma única máquina.

A grande maioria das experiências com o desenvolvimento de SDs utiliza redes locais de computadores, que oferecem características bastante adequadas para a obtenção de um bom desempenho, compartilhamento e tolerância a falhas.

A tolerância a falhas no hardware e no software é uma característica dos SDs de forma a garantir que , sob a ação de defeitos, o sistema continue funcionando, embora com alguma degradação no seu desempenho.

2.2 Sistemas Operacionais Distribuídos

Um Sistema Operacional Distribuído (SOD) compreende um único sistema operacional homogêneo implementado para o ambiente como um todo [34]. O que distingue o SOD de outros sistemas operacionais centralizados é o fato de que os recursos a serem gerenciados e os serviços oferecidos serão distribuídos ao longo do sistema.

A estruturação lógica de SDs, principalmente de SODs, utiliza basicamente o modelo de processos ou o modelo de objetos [63].

O modelo de processos baseia-se na utilização de elementos ativos (processos), para a estruturação de todo o sistema. Esses processos podem encapsular tanto elementos ativos, consistindo dos programas referentes às atividades do sistema e do usuário, quanto elementos passivos, correspondendo aos recursos e suas operações, alocados em gerenciadores de recursos. A comunicação entre os processos, implementada pelo SOD, pode ocorrer de duas formas: através de troca explícita de mensagem e através de chamada remota de procedimento.

O modelo de objetos baseia-se no encapsulamento das várias partes de um sistema em elementos denominados objetos. Os objetos são estruturados de forma a apresentarem um conjunto de operações responsáveis pelo seu comportamento. Para acessar um objeto, um processo deve ter capacidade para fazê-lo, usando o conhecimento do nome do objeto e a posse da autorização para acessar algumas ou todas as suas operações. A permissão para executar uma função específica sobre um objeto é definida como um direito.

No campo de sistemas operacionais para ambientes distribuídos têm sido utilizadas duas abordagens para o desenvolvimento desses sistemas: a primeira refere-se a sistemas servidores, e a segunda, a sistemas simétricos ou integrados.

Na abordagem do servidor, algumas máquinas específicas são designadas para atender alguns tipos de serviços do sistema (servidores), enquanto outras são utilizadas para a execução de processos (clientes).

Na abordagem simétrica, cada processador pertencente a rede apresenta-se como uma estação independente. Nesse caso, o SOD tem a função de integrar as máquinas da rede em uma única máquina virtual, que forneça o acesso transparente e o compartilhamento implícito dos recursos disponíveis.

Os SODs, cujas características serão descritas a seguir, foram retirados da literatura por apresentarem alguns aspectos relevantes aos ambientes distribuídos no contexto deste trabalho.

2.2.1 CONIC

O projeto CONIC [35] é um sistema operacional distribuído "Soft Real Time", responsável por uma abordagem unificada e integrada de desenvolvimento, implementação e gerenciamento de grandes SODs de controle de processos de tempo real. O desenvolvimento do projeto abrange os itens:

- Uma rede de computadores;
- Uma metodologia de desenvolvimento de software de aplicação;
- Um SOD e um sistema de comunicação;
- Um sistema de gerenciamento de alterações.

O desenvolvimento do sistema foi realizado utilizando-se a linguagem PASCAL, estendida com o conceito de módulo e com primitivas de troca de mensagem. A separação da linguagem de programação de módulos individuais da linguagem de configuração do sistema facilita a obtenção de configuração flexível, modularidade e componentes de software reusáveis. O ambiente de execução em que o sistema CONIC é implementado, no Imperial College, consiste de computadores heterogêneos tais como: computadores VAXs, estações de trabalho SUNs e alguns PDP11s, interligados via rede ETHERNET. Cada estação pertencente a rede apresenta um cópia do núcleo que fornece o suporte a módulos concorrentes. A comunicação é realizada através de troca de mensagens. O sistema operacional de cada estação mantém um conjunto de serviços relativos ao tratamento de módulos e coopera com as outras estações formando o SOD. A sua estrutura lógica corresponde a uma abordagem simétrica orientada a processos. O núcleo foi implementado, na sua maior parte, com o uso da linguagem PASCAL, e de uma pequena parte em linguagem "Assembly". O sistema operacional foi desenvolvido com o uso da linguagem CONIC.

2.2.2 MICROS

O sitema operacional distribuído MICROS [74] é um sistema multi-usuário, executando programação concorrente. O sistema contém dezesseis processadores, fracamente interligados através de uma rede extensível e reconfigurável. Cada nó da rede contém uma cópia local do núcleo do sistema, que consiste de um conjunto de processos concorrentes programados em Pascal Concorrente. O sistema utiliza gerentes redundantes no topo da hierarquia e ligações latentes entre gerentes do mesmo nível, com o objetivo de recuperação de falhas. Cada nó da estrutura hierárquica troca informações de controle e de estado somente com os níveis vizinhos da hierarquia. Os processos responsáveis pelo gerenciamento e os programas de usuários são escritos em Pascal Seqüencial e são colocados dinamicamente nos nós. A comunicação entre processos é feita através de troca de mensagens. A evolução do projeto desse sistema consistiu de sua reprogramação com o uso da linguagem MODULA-2.

2.2.3 AMOEBA

O sistema operacional AMOEBA [62] foi desenvolvido para controlar um conjunto de processadores interligados por uma rede. O sistema baseia-se no conceito de serviço implementado em processos servidores, cuja requisição de solicitação de serviços deve ocorrer por meio de troca de mensagens. A estrutura utilizada considera seis níveis de protocolos. Os dois primeiros correspondem aos níveis físico e de enlace de dados. O terceiro nível, corresponde ao monitor, que juntamente aos dois anteriores formam o núcleo do SOD. Os outros dois níveis constituem o sistema operacional. O último nível é por conta do usuário.

2.2.4 LOCUS

O sistema operacional distribuído LOCUS [66] é um sistema operacional simétrico, compatível com o UNIX, e orientado para aplicações de uso geral. O sistema foi implementado em um protótipo formado por um conjunto de dezessete computadores VAX/750, interligados através de um barramento tipo ETHERNET. O sistema oferece alto grau de transparência da rede, ao mesmo tempo em que apresenta bom desempenho e facilidades tais como: repetição automática e flexível de armazenamento a nível de arquivos. Cada computador possui sua cópia do núcleo e do sistema operacional. A programação do sistema foi realizada com a linguagem C.

2.2.5 MOS

O sistema operacional distribuído MOS ("Multicomputer Distributed Operating System) é um sistema operacional simétrico voltado para aplicações de uso geral [7]. O sistema é constituído de um conjunto de multiprocessadores homogêneos em máquinas autônomas interligados por uma rede local. O sistema tem as seguintes características: multiplicidade de recursos, transparência da rede, controle descentralizado, autonomia, disponibilidade e cooperação. A elaboração do sistema operacional foi feita de acordo com o modelo de objetos, configurando-se como um gerente de objetos abstratos. Os únicos objetos ativos do sistema são os processos que podem alterar o estado dos outros objetos. Cada objeto reside em uma máquina, podendo ser manipulado diretamente somente pelo núcleo. Cada máquina possui uma cópia idêntica do núcleo.

2.3 Sistemas Distribuídos Tempo Real

O conceito de Sistemas Distribuídos Tempo Real (SDTR) tem muitas interpretações, porém todas têm como característica comum o tempo de resposta - o tempo em que o sistema precisa para produzir uma saída a partir de uma determinada entrada associada.

A correção de Sistemas de Tempo Real (STR), ao contrário dos sistemas processados em lote ou tempo compartilhado, não depende somente do resultado lógico do processamento, mas também dos instantes de tempo em que esses resultados são produzidos [14]. Esse requisito básico tem influenciado decisivamente as arquiteturas de hardware e software de sistemas dedicados ao suporte às aplicações em tempo real. Tais sistemas tendem a ser muito complexos, particularmente quando são distribuídos, onde os atrasos e erros de comunicação devem ser considerados.

Os profissionais do campo do projeto de STRs distinguem esses sistemas de dois modos:

- Sistemas "Hard Real Time"
  - — Esses são sistemas onde é extremamente imperativo que as respostas ocorram dentro de prazos especificados. Exemplo: sistemas de controle de tráfego aéreo, sistemas para aplicações militares, satélites, etc.
- Sistemas "Soft Real Time"
  - — Esses são sistemas onde os tempos de resposta são importantes, mas o sistema funcionará corretamente no caso de ocasionalmente ocorrerem alguns atrasos. Exemplo: sistema de aquisição de dados, controle de processos lentos, sistemas de manufatura, reservas de passagens., etc.

Neste capítulo, os termos STR e SDTR são usados tanto para sistemas "Soft Real Time " quanto para sistemas " Hard Real Time". Nesses sistemas, o computador é ligado diretamente a algum equipamento físico e tem como objetivo monitorar ou controlar a operação desse equipamento.

Os SDTRs são, em geral, complexos. A próxima geração desses sistemas será mais complexa do que a atual devido a varios fatores [45] : conterá um comportamento altamente dinâmico e adaptativo; deverá exibir um comportamento inteligente; deverá possuir um longo período de vida útil e será caracterizada por possuir consequências catastróficas se as restrições lógicas e temporais não forem obedecidas [59]. O desenvolvimento desses sistemas pode ser muito simplificado se dispusermos de ambientes e linguagens de tempo real que particionem esses sistemas complexos em componentes menores, modulares, que possam ser gerenciados de uma maneira mais fácil.

Um dos problemas, também associado com a produção de software para sistemas que exibem concorrência, é o de como expressar essa concorrência. As linguagens modernas, tais como, ADA, CONIC, MESA e OCCAM, têm suporte à programação concorrente, o que simplifica a programação e o desenvolvimento de aplicativos tempo real.

A fim de reduzir a complexidade desses sistemas, estão sendo realizadas pesquisas em ambientes que possibilitem considerar abstratamente a correção lógica e temporal e de garantir, em tempo de execução, o cumprimento dos requisitos de tempo previamente especificados. As pesquisas são recentes e buscam a integração de metodologias de desenvolvimento [56], formalismos, ferramentas de especificação e validação [20,54] e estratégias de escalonamento [79].

Algumas considerações que se originam quando aplicações distribuídas são utilizadas e suas implicações em aplicações de SDTRs [14] são apresentadas nos próximos itens.

Alguns dos tópicos brevemente discutidos são:

- Partição e Configuração;
- Confiabilidade;
- Escalonamento por Prazo.

2.3.1 Partição e Configuração

Uma aplicação distribuída precisa ser particionada e configurada em algum ponto durante o seu projeto e implementação. O ponto em que essa partição ocorre pode ter reflexos em termos de portabilidade e confiabilidade da aplicação, bem como da funcionalidade da linguagem de programação de tempo real.

Existem muitas maneiras de particionar uma aplicação para executar em um SD. Essas maneiras podem ser, genericamente, divididas em dois grupos: um grupo que faz uma abordagem considerando a aplicação como um único programa e outro grupo que faz uma abordagem da aplicação considerando-a como um multi-programa [12].

Dentro do primeiro grupo, a aplicação é escrita como um único programa e depois particionada em módulos que se comunicam utilizando mecanismos normais de comunicação. No segundo grupo, a aplicação é escrita como uma coleção de programas separados e alocados em diversas máquinas do sistema, comunicando-se e sincronizando-se através de um sistema operacional básico.

Esse item refere-se a aplicações escritas como um único programa. A vantagem dessa abordagem é que ela permite que as interfaces sejam checadas em tempo de compilação e não restringe a comunicação entre processos com tipos pré-definidos. Existem dois tipos de estratégias, para essa aproximação: pré-partição e a pós-partição [12].

2.3.1.1 Pós-Partição

Essa estratégia particiona o programa depois que ele foi escrito. O programa é escrito sem preocupações quanto à arquitetura a ser utilizada. As ferramentas de software, proporcionadas pelo ambiente, devem ser responsáveis em:

- Descrever a configuração (que pode ser escolhida pelos projetistas);
- Particionar o programa em componentes para a distribuição;
- Configurar os componentes em nós individuais.

O "Honeywell Distributed ADA Project" [17,32] adota essa estratégia, usando ADA em um ambiente distribuído.

Alguns pontos podem ser observados nessa estratégia:

- A linguagem necessita de facilidades de gerenciamento para partição e configuração, requerendo uma linguagem não trivial;
- O software é portável: o mesmo programa pode ser mapeado em diferentes configurações;
- O custo em tempo de execução da replicação e partição é escondido do programador da aplicação;
- É difícil para o programador especificar que procedimentos de recuperação ou serviços de degradação devem ser usados em caso de ocorrer falha de um processador.

A estratégia de pós-partição requer o suporte de ferramentas de software apropriadas e facilidades em tempo de execução.

2.3.1.2 Pré-Partição

Essa estratégia seleciona uma construção de uma linguagem (construção em módulos), como sendo a unidade de partição que deve ser usada através do projeto e programação da aplicação. A idéia básica, nessa estratégia, é a da existência de um nó virtual, que é uma abstração de um nó físico em um sistema distribuído.

As características dos nós virtuais são:

- Unidades de modularidade em um SD;
- Unidades de reuso;
- Proporcionam interfaces bem definidas para outros nós virtuais no sistema;
- Encapsulam recursos locais. Todos os acessos para esses recursos, feitos por nós virtuais remotos, são realizados via a interface do nó virtual;
- Podem consistir de um ou mais processos, que podem se comunicar com processos em outros nós virtuais via as interfaces proporcionadas; normalmente essa comunicação é feita através de alguma forma de mecanismo de troca de mensagem;
- Mais de um nó virtual pode ser mapeado em um único nó físico; contudo, um nó virtual não pode ser distribuído entre máquinas;
- Unidades usadas para configuração e reconfiguração.

De muitas maneiras os nós virtuais são análogos a objetos em uma abordagem de projeto direcionada a programação orientada a objetos [6].

A característica de nó virtual é encontrada em muitas linguagens projetadas para suporte à programação distribuída. Por exemplo: o "group module" em CONIC [35], o "resource" em SR [4], o "guardian" em ARGUS [40] e o "processor module" em StarMod [16]. Em algumas linguagens, como ARGUS, os nós virtuais são usados como uma base para a distribuição e para contribuir na tolerância a falhas.

OCCAM-2 [13], embora projetado para SDs é uma construção de baixo nível, sendo difícil identificar um nó virtual precisamente. MODULA-2 foi projetada para um sistema de um único processador; ela pode ser usada para SDs modificando alguns conceitos da linguagem. No caso de ADA, há muita controvérsia em como a linguagem seria usada em um ambiente distribuído [6,12].

O termo nó virtual tem sido muito utilizado em construções de linguagens de programação distribuídas. As experiências com linguagens como CONIC, StarMod, SR e ARGUS indicam que nós virtuais constituem uma abstração de distribuição muito útil. Embora essas linguagens tenham muitas diferenças, elas são baseadas em processos ou módulos contendo processos e utilizam primitivas de passagem de mensagem.

Três linguagens de programação (CONIC, SR e ARGUS) são apresentadas nos próximos itens, para ilustrar os tipos de facilidades disponíveis. Elas estão relacionadas, de uma maneira ou de outra, com o ambiente distribuído, apresentado a partir do capítulo 3.

a) CONIC

O CONIC [35] proporciona um ambiente distribuído com uma abordagem de SD mais próximo à linguagem para o projeto e implementação de "soft" SDTR dedicados. A característica básica desse ambiente é a separação da linguagem de programação dos requisitos funcionais do sistema ("CONIC Programming Language") [36], da linguagem para configuração de programas a partir de módulos pré-definidos ("CONIC Configuration Language") [22].

A linguagem de programação é baseada no PASCAL, com extensões para modularidade e troca de mensagens. Consta de um "Task Module" que substitui a unidade de programa PASCAL. Uma definição de um "Task Module" especifica um tipo (possivelmente com parâmetros) a partir do qual instâncias de módulos podem ser criadas usando a linguagem de configuração. Um programa, em CONIC, consiste de uma ou mais instâncias de um ou mais "Task Modules", que se comunicam através de portas, usando troca de mensagens fortemente tipificadas. Cada "Task Module" declara portas de entrada e saída. As mensagens são recebidas de portas de entrada e são enviadas para portas de saída. CONIC suporta troca de mensagens síncronas e assíncronas. A conexão entre as portas de entrada de um módulo e as portas de saída de outros módulos é realizada em tempo de configuração.

O nó virtual, no ambiente CONIC, é chamado de um nó lógico e é definido como um "Group Module", usando a linguagem de configuração. Este módulo consiste de um conjunto de tarefas que executam concorrentemente dentro de um espaço de endereçamento compartilhado. A comunicação entre tarefas em um mesmo "Group Module" é a mesma que a comunicação entre tarefas em "Group Modules" separados, exceto que apontadores podem ser passados no primeiro caso, o que não ocorre no último caso. A interface para um "Group Module" é especificada da mesma maneira que a interface de um "Task Module", isto é, como porta de entrada e saída. A configuração de um programa complexo pode ser feita em camadas.

b) "SYNCHRONIZING RESOURCES"

Synchronizing Resources SR [3,4] é uma linguagem de programação distribuída projetada para a implementação de sistemas de propósito geral. SR tem características muito relevantes e interessantes, embora faltem algumas facilidades referentes ao tempo, o que é fundamental em uma linguagem para tempo real.

Um nó virtual em SR é chamado "Resource" sendo semelhante, em termos de funcionalidade, ao conceito de módulo. "Resources" podem ser criados dinamicamente, mas não podem ser aninhados. Um "Resource" tem uma parte de especificação, que define as operações proporcionadas por ele e uma parte que implementa a funcionalidade do "Resource". A parte de implementação de um "Resource" é conhecida como seu corpo e, como módulos e "Packages", ele tem uma seção de iniciação para declarações, processos e procedimentos. Diferentes de módulos "Packages", um "Resource" tem também uma seção de finalização, que é executada antes do "Resource" sair do contexto. Isso permite a liberação de outros "Resources", a liberação de espaços de "Heap" e outros. SR contém comandos para a criação e destruição de instâncias de um "Resouce" em máquinas particulares.

Operações entre "Resources" são canais de comunicação parametrizados, embora o acesso à variáveis compartilhadas seja permitido entre processos no mesmo "Resource". SR suporta uma variedade de modelos de sincronização. O cliente de um "Resource" pode requisitar uma operação de duas maneiras: via um comando de requisição "Call" ou via um comando de requisição "Send". Na requisição "Call", um processo emissor não é reiniciável até que o "Resource" tenha recebido a requisição, processado-a e retornado uma resposta. A requisição "Send" é uma mistura entre os envios assíncronos e síncronos, em que processo cliente é bloqueado até que a mensagem seja recebida pela máquina em que o "Resource" reside. Todas as combinações possíveis de interações cliente/servidor são possíveis, embora o servidor possa especificar algumas restrições.

C) ARGUS

ARGUS [41] é uma linguagem de programação projetada para suportar aplicações envolvidas com a manipulação de dados "on-line" de longa vida em um ambiente distribuído. Por exemplo, sistemas de reservas de passagens aéreas e bancos. A linguagem suporta a abstração de nós virtuais (na forma de "guardians" e a noção de transações atômicas (chamadas "atomic actions"). Um programa em ARGUS consiste de uma coleção de "guardians". A interface para um "guardian" é realizada via chamada remota de procedimentos (chamados "handlers"). Outros processos podem ser criados para executar em "background". Um estado de um "guardian" consiste de objetos estáveis e voláteis, onde os objetos estáveis são escritos periodicamente para a memória estável. Quando ocorre a falha de um "guardian", os objetos voláteis e os processos ativos são destruídos, mas os objetos estáveis permanecem. Durante a recuperação do processador do "guardian", a memória estável de cada "guardian" é examinada e a memória volátil é recriada para um estado consistente. O "guardian" pode, então, continuar sua operação.

ARGUS suporta a característica de um tipo de dado abstrato atômico. As transações atômicas podem ser escritas pelo programador, elas podem iniciar em um "guardian" e fazer chamadas para "handlers" em outros "guardians", que podem acessar "guardians" remotos. As transações atômicas podem se submeter ou abortar; se elas abortam, então todos os objetos atômicos atualizados, durante a transação, são restaurados para seu estado anterior. As transações interrompidas por falhas do hardware são abortadas.

2.3.2 Confiabilidade

A disponibilidade de SDs permite a uma aplicação tornar-se independente da ocorrência de falhas de algum dos processadores pertencentes ao sistema, mas introduz a possibilidade do aparecimento de novos problemas não encontrados em sistemas centralizados com um único processador.

Sistemas com multi-processadores introduzem o conceito de falha parcial do sistema. Em algum momento, durante a execução do sistema, o meio de comunicação pode perder,corromper ou mudar a ordem das mensagens.

A comunicação entre processos em máquinas remotas requer camadas de protocolos de maneira que as condições de erro transitórias possam ser toleradas pelo sistema. A ISO ("International Standards Organization") definiu padrões envolvendo o conceito do OSI ("Open Systems Interconnections"). O Modelo de Referência OSI é um modelo em camadas consistindo das camadas: aplicação; apresentação; sessão; transporte; rede; enlace de dados e física. Esse modelo foi desenvolvido, primariamente, para redes de longa distância, caracterizadas por comunicação "low-bandwith" com taxas altas de erros. A maioria dos SDs dedicados utilizam redes locais e estão fechados ao mundo exterior. Redes locais são caracterizadas por comunicação "high-bandwith" com taxas baixas de erros.

As falhas de processadores podem ser toleradas através da redundância estática ou dinâmica. Em ambientes com ferramentas que utilizam o conceito de nó virtual é possível realizar-se, estaticamente, a replicação de nós virtuais em processadores diferentes. Um dos problemas de proporcionar tolerância a falhas de forma transparente para a aplicação, é que se torna impossível para o programador especificar se a execução realizou-se de forma segura ou degradada. Uma alternativa para a redundância estática e replicação virtual dos nós é permitir que a falha do processador seja manipulada dinamicamente pelo programador da aplicação. Isso requer:

- A falha do processador seja detectada e comunicada para os outros processadores do sistema;
- A degradação do sistema seja avaliada;
- O software concorde com o envio de respostas para a falha e tome as ações necessárias para

efetuar essa resposta;

- Tão logo quanto possível, o processador que falhou e seu software, precisam ser reparados de modo a permitir que o sistema volte ao seu estado normal, sem erros.

A maioria das linguagens de programação não proporcionam facilidades adequadas para tratar a reconfiguração dinâmica, após a falha de um processador. O ambiente CONIC proporciona um sistema operacional que permite, ao operador do sistema , reconfigurar os "task modules"; contudo, o sistema tem o propósito de atualizar o sistema no caso de "bugs" fixos ou de introduzir uma nova funcionalidade devido a mudanças de requisitos [57].

2.3.3 Escalonamento por Prazo ("Deadline Scheduling")

Nos STRs com restrições severas de tempo ("Hard Real Time") as tarefas têm de ser executadas corretamente do ponto de vista lógico e do instante em que esses resultados são fornecidos [45]. Caso contrário, o sistema pode sofrer conseqüências severas. Tipicamente, uma tarefa de tempo real é caracterizada pelas suas restrições temporais, restrições de precedência e recursos necessários à execução. Nos STRs, o escalonamneto das tarefas é o problema mais importante já que é o algoritmo de escalonamento que permite que as tarefas cumpram os seus "deadlines".

O desenvolvimento de políticas de escalonamento apropriadas aos sistemas com mais de um processador (multiprocessamento ou SDs) é um problema muito complexo. Os algoritmos ótimos para os sistemas monoprocessadores ("Earliest Deadline", "Least Laxity Time", Taxa Monotômica, e outros) não são ótimos para um número maior de processadores [14]. Essa dificuldade não é surpresa, pois é conhecido que o escalonamento ótimo para sistemas multiprocessadores é NP-completo, ou seja, o tempo de processamento cresce exponencialmente com o número de tarefas [14]. É necessário simplificar o problema e obter-se algoritmos que forneçam resultados aceitáveis apesar de sub-ótimos. Supondo-se uma alocação estática das tarefas nesses sistemas, cada processador, localmente, poderia utilizar os algoritmos discutidos no contexto dos sistemas monoprocessadores. Essa situação, entretanto, não utiliza toda a flexibilidade oferecida pela estrutura dos sistemas com mais de um processador, especialmente no balanceamento da carga de processamento. Para atender essa classe de problemas são necessários algoritmos que possibilitem a migração de tarefas entre processadores. Esse é um problema extremamente complexo para o qual poucos trabalhos têm sido desenvolvidos.

No caso dos SDTRs o tempo de comunicação é um parâmetro importante. Nesses sistemas, o escalonamento de tarefas deve ser integrado ao escalonamento das mensagens de modo a viabilizar o cumprimento das restrições temporais associadas a aplicação.

2.4 Mecanismos de Comunicação e Sincronização em SDTRs

A comunicação entre os processos de um SDTR, independente da localização, ocorre através de troca de mensagens. O mecanismo de troca de mensagens possui a característica de simultaneamente conduzir o dado e implementar a sincronização. Esse mecanismo, ao contrário dos mecanismos de comunicação através de memória compartilhada, não impõem uma dependência em relação à estrutura de hardware utilizada, durante o desenvolvimento de uma aplicação.

Os mecanismos de comunicação e sincronização através de troca de mensagens constituem-se no envio (SEND) de mensagens e na recepção (RECEIVE) de mensagens. A comunicação é caracterizada pela troca de informações entre processos, enquanto que a sincronização é identificada através da ordem em que os eventos ocorrem. Em todos os sistemas baseados em mensagens existe uma sincronização

implícita em que, uma mensagem somente pode ser recebida por um processo após a mesma ter sido enviada.

O envio de uma mensagem ocorre através da execução de um comando

SEND mensagem TO destino

enquanto que o recebimento de uma mensagem realiza-se pela execução do comando

RECEIVE mensagem FROM origem

2.4.1 Tipos de Sincronização

As primitivas para troca de mensagens podem ser implementadas segundo algumas variações no modelo de sincronização entre processos, que se originam das semânticas da operação SEND, que podem ser, genericamente, classificadas como:

- Primitivas Assíncronas:
  - — Uma primitiva é dita assíncrona ou não-bloqueante quando o processo que a executa continua a sua execução independentemente da ocorrência de qualquer evento.
- Primitivas Síncronas:
  - — Uma primitiva é dita síncrona ou bloqueante quando o processo que a executa fica bloqueado até que a mensagem seja recebida pelo processo destino.

2.4.1.1 Mecanismos de Comunicação Síncronos

Os mecanismos de comunicação e sincronização síncronos realizam a sincronização e a transferência de dados de maneira simples, sem a obrigatoriedade de uso de buffers, uma vez que a transmissão poderá ser feita somente quando ocorrer a sincronização entre os processos.

Existem três tipos de mecanismos síncronos [45]:

- "Rendez-Vous";
- "Rendez-Vous" Estendido;
- Chamada Remota de Procedimento (CRP).

O "rendez-vous" é obtido através de primitivas de emissão e de recepção bloqueantes, colocadas em processos distintos. A execução dessas primitivas, em tempos diferentes, fará com que o processo que executar a primitiva antes do outro fique bloqueado até que ocorra a sincronização entre os dois processos, e a consecutiva transferência da mensagem; em seguida, ambos os processos continuarão sua execução concorrentemente. No caso extremo de ambas as primitivas serem executadas ao mesmo tempo, os dois processos utilizarão o tempo necessário para a sincronização e a transferência da mensagem para, então, continuarem simultaneamente sua execução normal.

O "rendez-vous" estendido, como o nome indica, é uma extensão do mecanismo anterior, com a diferença de que o processo receptor somente envia uma resposta ao processo emissor, após a execução de um corpo de comandos, que opera sobre a mensagem recebida. Essa resposta enviada pode conter parâmetros que indicam o resultado da computação realizada pelo processo receptor. O processo emissor fica bloqueado até o término do corpo de comandos no processo receptor, quando então o "rendez-vous" se completa.

A CRP é um mecanismo que é uma variante de "rendez-vous" estendido e que foi proposta na linguagem "Distributed Processes" apresentada por Brinch Hansen[11]. Esse mecanismo é semelhante ao "rendez-vous" estendido, com a diferença de que, nesse caso, o processo servidor não fica bloqueado na recepção de mensagens. Os procedimentos que podem ser chamados externamente, apesar de estarem contidos no processo, não fazem parte do seu processamento, sendo declarados no início, juntamente com as variáveis locais. No momento em que esses procedimentos são invocados, os mesmos são executados em exclusão mútua, de uma forma intercalada e não determinística com a execução do corpo do processo.

2.4.1.2 Mecanismos de Comunicação Assíncronos

Uma comunicação assíncrona caracteriza-se pelo não bloqueio dos processos que executam as primitivas. Um processo emissor continua a sua execução imediatamente após a operação de envio da mensagem. Uma das vantagens no envio não bloqueante de mensagens é a implementação de mensagens do tipo difusão ("broadcasting"), onde a mensagem é enviada a vários destinatários simultaneamente. No caso do uso da primitiva de recepção assíncrona, o processo que a executa, não é bloqueado caso naquele instante a mensagem não se encontre disponível.

A utilização de primitivas não bloqueantes permite altas taxas de paralelismos entre os processos, pelo fato de não exigir sincronização. Esse tipo de comunicação, entretanto, apresenta o incoveniente introduzido pelo tratamento de filas de mensagens. Esse inconveniente pode ocorrer devido ao fato dos processos cooperantes serem totalmente assíncronas e de ser possível que um determinado processo envie uma série de mensagens para um outro, sem que o processo receptor esteja pronto para recebê-las; ocorrendo esse caso, o núcleo deve fornecer um tratamento adequado e guardá-las em uma fila até que o processo receptor efetue as recepções pendentes. Esse tratamento é crítico pelas conseqüências que podem advir em sistemas tempo real.

2.5 Contexto e Contribuições deste Trabalho

A versão centralizada do ambiente STER, a partir da qual foi implementada a versão distribuída, é apresentada, bem como as contribuições proporcionadas por este trabalho.

2.5.1 O Ambiente de Desenvolvimento STER

O ambiente STER privilegia uma metodologia que explora o conceito de módulo, a qual mostra-se especialmente adequada aos sistemas de controle de processos de tempo real ("SOFT REAL TIME") projetados para ter um longo tempo de vida útil, durante o qual estão sujeitos a mudanças.

Devido ao fato dos módulos apresentarem interfaces bem definidas, através de portas lógicas de comunicação utilizadas para a troca de mensagens, uma expansão ou reestruturação corresponde à inclusão, exclusão ou reconexão de módulos, através de mudanças nas ligações entre as suas interfaces.

O ambiente STER fornece um conjunto de ferramentas para a compilação de programas, configuração da aplicação e depuração de erros. Em tempo de compilação, ligação e execução da aplicação, as checagens de erros são proporcionadas de maneira a assegurar a compatibilidade de operação e de troca de mensagens entre os componentes de software.

O modelo que serve de base ao ambiente STER baseia-se no projeto CONIC [35], desenvolvido no Departamento de Computação do Imperial College (Londres).

A construção de um programa de aplicação, nesse ambiente, é composta de duas etapas: a primeira etapa consiste da programação dos módulos que implementam as funções do sistema através da Linguagem de Programação de Módulos (LPM) e a segunda etapa consiste da configuração da aplicação a partir dos módulos disponíveis através da Linguagem de Configuração de Módulos (LCM).

A LPM [43] é uma extensão da linguagem PASCAL [31], que foi estendida para suportar modularidade, mecanismos de comunicação e sincronização e primitivas de troca de mensagens, permitindo:

- Declaração de tipos de módulos;
- Declaração de portas de entrada e saída;
- Declaração de mensagens;
- Envio e recepção de mensagens de comunicação e sincronização através de portas;
- Suspensão da execução de um módulo por um período de tempo;
- Mudança da prioridade de um módulo dinamicamente;
- Tratamento de interrupções lógicas em alto nível.

Um módulo consiste de um programa seqüencial que implementa uma ou mais funções, podendo se comunicar com outros módulos enviando e recebendo mensagens síncronas e assíncronas através de interfaces bem definidas.

Em tempo de configuração, instâncias de módulos são criadas a partir de tipos de módulos. Instâncias de módulos trocam mensagens e executam funções particulares, tais como controle de um dispositivo ou gerenciamento de um recurso.

A interface de um módulo é definida em termos de portas de comunicação fortemente tipificadas, que especificam todas as informações requisitadas pelo módulo. Nos instantes em que um determinado módulo necessita enviar ou receber mensagens, ele faz referência a sua própria porta de saída ou de entrada; ou seja, o uso das portas de um módulo é de domínio local ao módulo. Uma porta de saída denota a interface em que uma transação de mensagem pode ser iniciada. Uma porta de entrada denota a interface em que várias transações de mensagens podem ser recebidas. A declaração de uma porta define os tipos de operações nas quais ela será empregada.

A LPM suporta primitivas de comunicação para enviar uma mensagem através de uma porta de saída ou para receber uma mensagem a partir de uma porta de entrada. O tipo de comunicação envolvida, síncrona ou assíncrona, também é definida na declaração da porta. O tipo da mensagem precisa ser o mesmo do tipo da porta através da qual será transmitida.

Unidades de Definição (UD) são usadas por módulos que se comunicam através de troca de mensagens e, devido a isso, requerem o uso de uma mesma definição em várias declarações de portas e mensagens. A UD é um arquivo onde os tipos referenciados em declarações de portas e mensagens devem ser declarados, exceto os tipos elementares do PASCAL. Esse arquivo é compilado separadamente e importado nos módulos através do comando USE.

O ambiente STER é particularmente forte em suas facilidades de configuração. Uma Linguagem de Configuração (LCM) separada é empregada para especificar a configuração de componentes de software em uma estação. Isto proporciona uma descrição de configuração concisa e facilita o reuso de componentes da aplicação (módulos) em configurações diferentes. As aplicações são construídas como conjuntos de um ou mais módulos interconectados formando estações lógicas.

Em muitas aplicações, em particular as aplicações de tempo real não críticas, a que esse ambiente se propõe atualmente a atender, é muito útil ter-se um conjunto de componentes para serem usados em

configurações diferentes ou para futuramente auxiliarem no tratamento de recuperação de falhas e reconfiguração dinâmica, cujos componentes podem ser reconfigurados separadamente.

O Suporte de Tempo Real implementa os serviços que atendem, em tempo de execução, aos comandos, procedimentos e funções pré-declaradas da LPM que não fazem parte do PASCAL padrão [1].

A modularidade, a possibilidade de reutilização de componentes de software e a configuração flexível dos componentes, proporcionados pelo ambiente STER, são facilitados pela separação da programação dos componentes de software individuais da configuração geral da aplicação.

O ambiente STER, em suas duas versões, centralizada e distribuída, tem sido utilizado, na prática, para a construção de um grande número de aplicações, que se estendem desde utilitários de suporte ao sistema e serviços, até aplicações utilizando algoritmos distribuídos e aplicações distribuídas. Como exemplo do primeiro caso, tem-se protocolos de comunicação, prototipagem de especificações em LOTOS, expansões da versão distribuída para sistemas tempo real com restrições severas de tempo ("Hard Real Time"), simuladores de eventos discretos, tais como, simulador de células flexíveis de manufatura, simulador de elevadores, simulador de metrô, e, como exemplo do segundo caso, tem-se um simulador de monitoramento de pacientes em um hospital.

2.5.2 Definição do Sistema de Computação Distribuída

Nesta dissertação, o termo Sistema Distribuído refere-se a um sistema de computação que tem as seguintes características:

- A Arquitetura de hardware do sistema consiste de múltiplas estações autônomas de processamento, homogêneas, fracamente acopladas, constituídas de um número arbitrário de processadores independentes com memória local e, opcionalmente, dispositivos periféricos de comunicação com o meio externo. Essas estações estão interligadas por uma rede serial, assíncrona, com velocidade de 9600 bps (bits por segundo);
- A Arquitetura de Software do sistema separa a programação dos componentes de software individuais (módulos) da configuração da aplicação. O número de módulos é fixado em tempo de configuração. Essa arquitetura é suportada por um sistema operacional distribuído desenvolvido para o ambiente "SOFT REAL TIME", com uma abordagem mais próxima às linguagens de programação;
- Configuração estática das aplicações distribuídas;
- O controle de processos em tempo real é realizado por um número arbitrário de módulos sem qualquer relação hierárquica entre eles. É realizada uma cooperação, através de troca de mensagens, entre os módulos independentes.

Nos próximos capítulos o termo *processo* é referenciado como o processo físico utilizado em controle de processos tempo real e o termo *módulo* ou *instância de módulo* é referenciado segundo o conceito de processo utilizado em Sistemas Operacionais (SOs), que pode ser definido como um programa em execução.

2.5.3 Contribuições desta Pesquisa

Esta pesquisa proporciona contribuições no campo de SOs distribuídos no domínio de controle de processos de tempo real.

O ambiente projetado permite a sua conexão com qualquer rede local disponível no mercado.

O trabalho envolveu cinco aspectos:

- A extensão da metodologia associada ao ambiente (capítulo 3).
- A extensão da linguagem LCM (capítulo 4).
- A implementação da extensão do núcleo de tempo real (capítulo 5).
- A implementação de uma rede local simplificada para validação e testes do ambiente (apêndice D).
- A implementação de um exemplo de aplicação distribuída (capítulo 6).

Capítulo 3

O Sistema Distribuído STER

Este capítulo apresenta a versão distribuída do Ambiente STER detalhando as soluções adotadas para a arquitetura de software, desenvolvida como uma extensão da versão centralizada.

O ambiente implementado procura atender os requisitos essenciais para a programação de software distribuído, orientado para aplicações de tempo real com restrições não críticas de tempo. O ambiente STER consiste de uma metodologia, que explora o conceito de módulo orientada ao projeto de sistemas distribuídos; de duas linguagens: a Linguagem de Programação de Módulos (LPM) e a Linguagem de Configuração de Módulos (LCM) e de um Núcleo Operacional Distribuído (NOD) que fornece o suporte necessário às linguagens e aos serviços oferecidos.

As linguagens e o núcleo são apresentados nos próximos capítulos.

3.1 Arquitetura de Software do Sistema Distribuído STER

A arquitetura de software dos sistemas distribuídos precisa ser projetada cuidadosamente para explorar as vantagens potenciais da distribuição e melhor refletir os seus princípios básicos, considerando os problemas de erros, falhas e atrasos causados pelo meio de comunicação.

O que se deseja é uma arquitetura de software que constitua uma abstração da máquina física utilizando transparentemente as características do hardware.

O objetivo da arquitetura de software adotada é separar a programação dos componentes de software individuais que implementam as funções do sistema, da configuração geral da aplicação. A modularidade, a possibilidade de reutilização e a configuração flexível dos componentes de software é facilitada por essa separação.

A existência de fases distintas para o desenvolvimento de aplicações no STER, programação de módulos e configuração de aplicações, propicia uma transparência a nível de projeto e conduz à independência dos módulos em relação à utilização de uma arquitetura centralizada ou distribuída para sua execução. Nesse sentido, o desenvolvimento de cada componente deve ser independente da estrutura do hardware e, somente na etapa de configuração, tal estrutura deve ser considerada.

A independência relativa ao hardware pode ir de um extremo, onde todos os módulos da aplicação residem em uma única estação, até o outro, onde cada módulo reside fisicamente em uma estação distinta

da rede. Essa flexibilidade deve ser obtida sem a necessidade de mudanças nos módulos de software que implementam cada uma das funções de controle, não havendo necessidade, inclusive, de recompilação destes. O engenheiro de processo obtém, assim, a possibilidade de desenvolver, testar, depurar uma aplicação em ambiente centralizado e de reconfigurar a mesma aplicação em ambiente totalmente distribuído.

A metodologia de projeto de software, associada ao STER, compõe-se, basicamente, das seguintes etapas:

a) Decomposição Funcional do Sistema em Módulos
A decomposição funcional do sistema em módulos segue o conceito de "independência de módulos" [51], podendo ser avaliada segundo dois métodos para medição de qualidade : coesão e acoplamento. É desejável decompor o sistema em módulos com alta coesão e baixo acoplamento. Na metodologia adotada os módulos não compartilham áreas de dados, reduzindo o grau de acoplamento dos mesmos. Esses módulos podem ser usados como componentes reusáveis na construção de diferentes sistemas.

b) Definição das Interfaces dos Módulos
As interfaces dos módulos são constituídas por portas lógicas de entrada e saída. Uma porta lógica é de domínio exclusivo do módulo que a contém. Esse conceito permite o desenvolvimento independente de cada módulo, sem levar em consideração a estrutura do hardware onde a aplicação será implementada.

c) Interligação dos Módulos
A interligação dos módulos constitui a etapa de configuração lógica da aplicação. A interligação de módulos é feita conectando-se portas de saída a portas de entrada do mesmo tipo. Nessas ligações, as mensagens fluem entre módulos.

A figura 3.1 apresenta a estrutura de uma aplicação no ambiente STER, desenvolvida segundo essa metodologia.

As etapas citadas independem da arquitetura alvo da aplicação ser centralizada ou distribuída. Ao

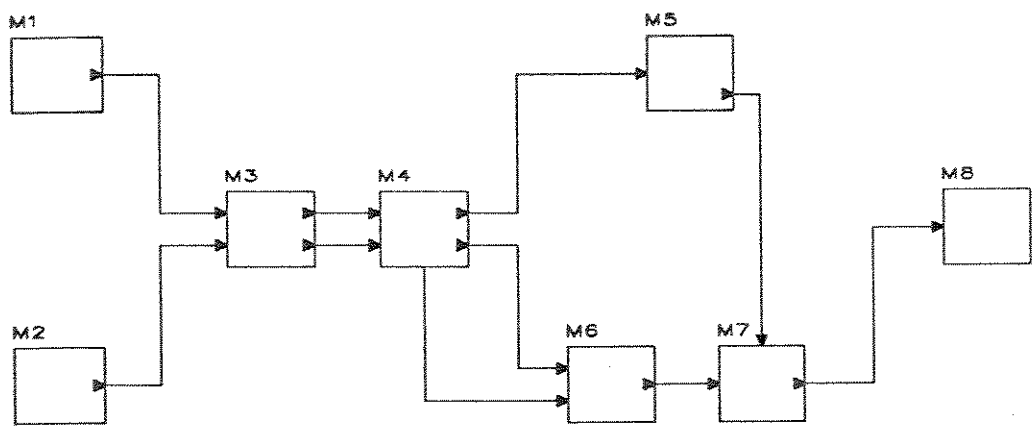

Figura 3.1 : Estrutura de uma Aplicação STER

concluir estas etapas, o projetista dispõe da configuração lógica do sistema independente da arquitetura.

A implementação da aplicação dar-se-á em duas fases: programação dos módulos e configuração da aplicação.

A funcionalidade de uma aplicação é implementada por módulos e unidades de definição usando a linguagem LPM.

Se o projetista visa a implantação do aplicativo num sistema distribuído, durante a configuração da aplicação uma etapa adicional (d) deve ser considerada:

d) Alocação dos Módulos às Estações
A alocação dos módulos às estações pertencentes à rede de comunicação é feita observando-se, basicamente, os seguintes aspectos: balanceamento de carga de processamento nas estações, fluxo de mensagens entre módulos, disponibilidade de periféricos, proximidade dos módulos aos processos controlados, tolerância a falhas, etc.

Aplicações distribuídas no ambiente STER são construídas através da linguagem LCM, estabelecendo uma estrutura hierárquica entre redes, estações e módulos. Os módulos obtidos (primeiro nível hierárquico), através da LPM, podem ser combinados em grupos de instâncias de módulos formando estações (segundo nível hierárquico) através da LCM. Essas estações constituem a unidade de distribuição da rede, podendo ser criadas instâncias de estações formando a estrutura física da rede (terceiro nível hierárquico).

O suporte à programação/especificação da etapa adicional requerida (d), introduz os conceitos lógicos de estação e rede. Uma estação lógica é uma abstração que contém uma maneira poderosa de estruturar os módulos que constituem essa estação. Da mesma maneira, uma rede lógica é uma abstração que contém as estações lógicas que participam dessa rede. Uma estação lógica é constituída por um conjunto de módulos, interconectados entre si, provida de uma interface semelhante às interfaces dos módulos. Como os módulos, as estações lógicas e as redes lógicas são tipos, no sentido de que mais do que uma instância de estação e rede podem ser criadas a partir do arquivo de código, representando um tipo de estação lógica e um tipo de rede lógica. Uma vez definidos, esses tipos lógicos de estações podem ser instanciados no momento da carga da aplicação na rede a partir da especificação da configuração, quando então é realizado o mapeamento das estações lógicas, em endereços físicos do hardware, bem como a interconexão das instâncias de estações, através da ligação de suas portas.

Novas configurações podem ser construídas, utilizando-se esses tipos de estações e formando tipos de redes. Na versão atual do STER existe um único tipo de rede, mas as extensões permitirão que esses sejam instanciados e conectados a outros tipos de redes e "gateways", formando configurações mais complexas.

Uma estação lógica pode, atualmente, executar em equipamentos compatíveis com o IBM-PC. Com as extensões, cada estação lógica poderá executar em máquinas heterogêneas, dependendo unicamente dos módulos de suporte que serão implementados.

A arquitetura adotada pelo Sistema Distribuído STER (SDS) fornece o suporte necessário às aplicações distribuídas de tempo real. Essa arquitetura proporciona as ferramentas LPM e LCM que combinam a simplicidade e segurança de uma linguagem de alto nível, com a flexibilidade de um Núcleo Operacional Distribuído (NOD).

Os próximos capítulos detalham as linguagens e o NOD.

A figura 3.2 apresenta uma visão hierarquizada da estrutura de uma aplicação no STER, onde são ilustrados os conceitos lógicos de estação e rede introduzidos na metodologia.

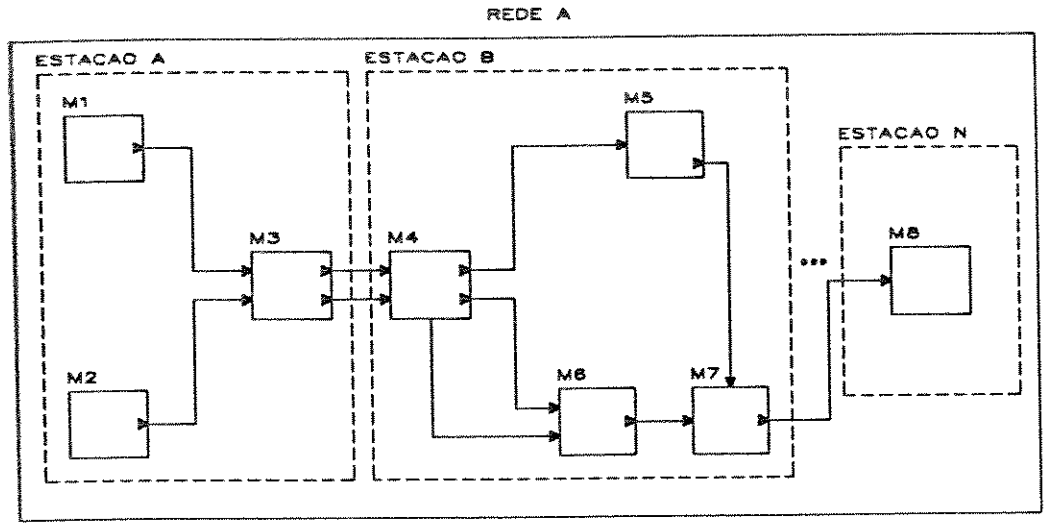

Figura 3.2 : Estrutura Lógica de uma Aplicação Distribuída

Capítulo 4

Linguagens de Programação e Configuração de Módulos

Em razão da metodologia incorporada ao ambiente STER ser caracterizada pela transparência com relação à arquitetura de hardware , a LPM não requer qualquer modificação com relação à versão desenvolvida anteriormente [43,44].

As primitivas de comunicação disponíveis na LPM, de envio e recepção de mensagens, mantêm a mesma sintaxe e semântica em relação à versão centralizada do STER, independentemente da comunicação tratada ser local ou remota.

No Apêndice A, é apresentada uma descrição BNF ("Bakus Normal Form") completa das linguagens LPM e LCM. No caso da LCM, foi necessário estendê-la, com o objetivo de incorporar os conceitos lógicos de estação e rede mencionados no capítulo anterior, conduzindo, dessa forma, à especificação de uma configuração estruturada em dois níveis.

Este capítulo apresenta uma visão geral da extensão feita na LCM, abordando os principais aspectos sintáticos e semânticos. Para a representação da sintaxe da linguagem são utilizados diagramas de sintaxe, onde as figuras circulares envolvem os símbolos terminais. Os símbolos não-terminais são representados pelas figuras retangulares. As semi-retas orientadas indicam a seqüência na qual os componentes da linguagem estão dispostos.

4.1 Linguagem de Configuração de Módulos (LCM)

A configuração de uma aplicação corresponde a um programa em LCM, onde são declarados os módulos que compõem a aplicação, as instâncias desses módulos agrupadas em nível de estações, as instâncias de estações agrupadas em nível de redes, e as conexões locais e remotas de suas portas.

Na atual versão de implementação, a LCM suporta apenas configuração estática, podendo o sistema ser implementado em ambiente centralizado ou distribuído.

A especificação de uma configuração corresponde a dois níveis independentes: *Construção STATION* e *Construção NETWORK*. Essas construções fornecem uma visão estruturada da especificação, onde são associadas uma configuração lógica com uma estrutura física de hardware.

A *Construção STATION* (primeiro nível) é usada para descrever uma estação lógica da rede, correspondendo a um programa escrito em Linguagem de Configuração a Nível de Estação (LCMS). Quando uma estação é especificada e processada, ela pode ser instanciada e interconectada a outras estações na fase da *Construção NETWORK* (segundo nível), correspondendo a um programa escrito em Linguagem de Configuração a Nível de Rede (LCMN), onde o usuário visualiza apenas as estações lógicas com as suas correspondentes interfaces e a estrutura física, onde a aplicação vai ser carregada e executada. Cada especificação de estação é compilada separadamente, gerando um arquivo descritor de estação. Esse arquivo é utilizado durante o processamento da *Construção NETWORK*, quando da instanciação e criação da estação na rede, para a geração de arquivos de saída, contendo tabelas de configuração específicas para cada estação pertencente à rede.

4.1.1 Vocabulário e Elementos Básicos da Linguagem

A seguir apresenta-se o conjunto de elementos básicos que compõem a linguagem. A representação usada na descrição do vocabulário é a BNF.

< vocabulário > ::= < vocabulário LCM >

< vocabulário LCM > ::= < palavras reservadas > | <símbolos LCM especiais>

< palavras reservadas > ::= NETWORK | STATION | CREATE | END_NETWORK | END_STATION | TO | INSTANCE | LINK | AT | NODE | ASSOCIATE | WITH

< símbolos LCM especiais > ::= "[" | "]" | "{" | "}" | "." | "," | ";" | ":" | "= "

A representação de um comentário em LCM é equivalente à representação usada no PASCAL, ou seja, qualquer seqüência de caracteres delimitada por "{" e "}" ou "(*" e "*)".

Um identificador LCM pode ser qualquer seqüência de letras, dígitos ou "_", que comece por uma letra e que não seja uma palavra reservada.

4.1.2 A Construção STATION

Uma aplicação distribuída no ambiente **STER** é construída a partir de tipos lógicos de estações pré-compilados. Cada tipo lógico de estação é especificado em um arquivo descritor de estação, através da LCMS.

A especificação de uma estação define um tipo, a partir do qual múltiplas instâncias podem ser criadas.

Na especificação **STATION** são descritas:

a) as portas que constituem a interface de uma estação lógica;

b) os tipos de módulos componentes e as respectivas instâncias desses módulos;

c) as conexões das instâncias de módulos através da interligação de portas;

d) a associação de portas internas dos módulos às portas da interface da estação, possibilitando assim, a exportação de nomes de portas internas de instâncias de módulos para o nível da estação, e permitindo o uso dessas portas na especificação da configuração distribuída.

Quando o usuário for especificar a *Construção NETWORK*, não precisará conhecer detalhes dos módulos e portas internas da estação lógica. Uma estação lógica é tratada como um módulo unitário, provido de sua respectiva interface.

Na criação das instâncias dos módulos pertencentes a uma estação lógica, os valores dos parâmetros reais das instâncias deverão ser determinados. Opcionalmente, podem ser usadas diretivas para definição da prioridade inicial das instâncias e do tamanho das áreas de memória que cada instância necessita para pilha e "heap".

A interface da estação é definida de maneira semelhante à declaração de portas de um módulo. As portas internas dos módulos podem ser associadas aos elementos da interface da estação, através de uma diretiva que associa a porta da instância do módulo a uma porta da interface.

4.1.2.1 Estrutura de um Programa de Configuração para a Construção STATION(LCMS)

A figura 4.1 apresenta a estrutura de um programa de configuração a nível da Construção STATION.

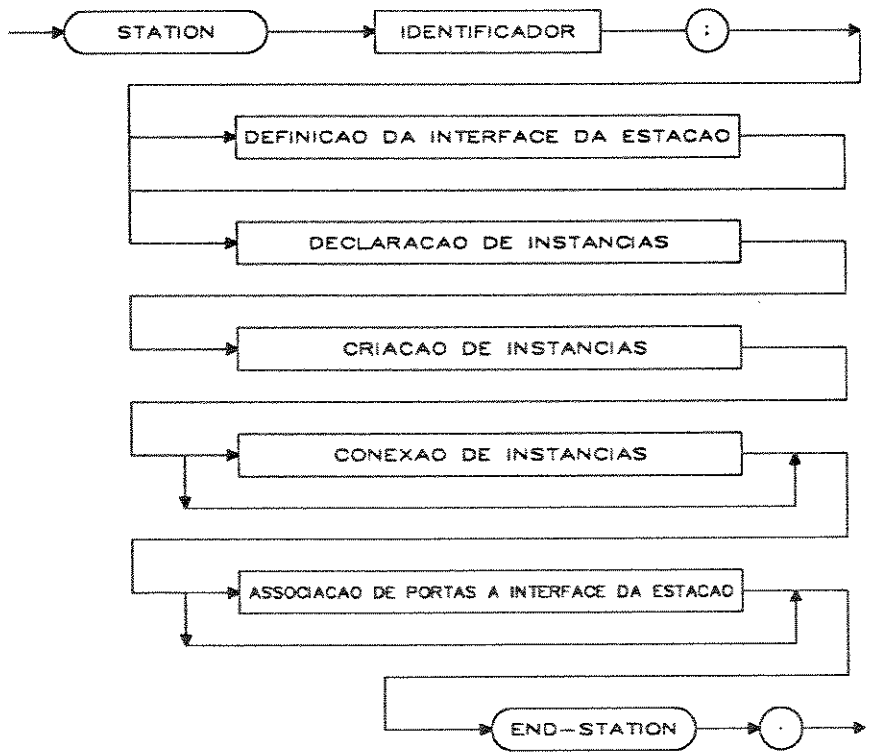

Figura 4.1: Estrutura de um Programa LCMS

4.1.2.2 Definição da Interface da Estação

A declaração da *Construção STATION* pode ter elementos de interface que são especificados através da declaração de portas (figura 4.2). Essas portas podem ser conectadas às portas de instâncias de estações, quando da especificação da *Construção NETWORK*. Portanto, as portas estabelecem a interface para a comunicação entre estações (portas de entrada e saída), de maneira semelhante à declaração de portas na declaração de módulos.

A cada porta é associado um tipo. Essa associação de tipos possibilita a realização de consistências nos comandos de troca de mensagens e na conexão de portas na fase de configuração de estações.

A declaração de uma porta define os tipos de operações de comunicação na qual ela será empregada. As operações primitivas em portas são: o envio de uma mensagem através de uma porta de saída e a recepção de uma mensagem a partir de uma porta de entrada. O tipo de comunicação envolvida, síncrona ou assíncrona, também é definido na declaração da porta.

Uma família de portas pode ser declarada, através da especificação do intervalo de variação do índice, após o identificador da porta. Essa construção é análoga à declaração de arrays em PASCAL.

O tipo associado a uma porta ou família de portas, pode ser elementar ou estruturado.

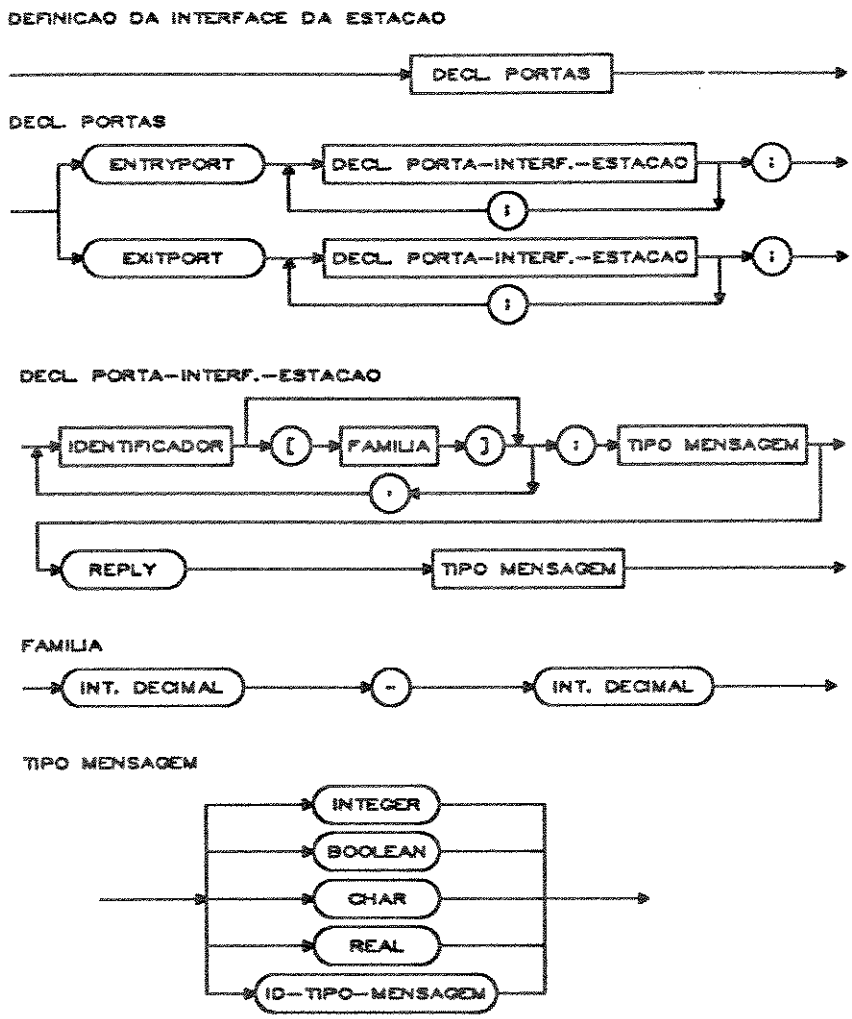

Figura 4.2 : Definição da Interface da Estação

4.1.2.3 Declaração de Instâncias de Módulos

A declaração de instâncias possui sintaxe e semântica equivalentes às descritas na versão LCM centralizada.

Nessa declaração, são associados nomes às várias instâncias dos módulos da aplicação (figura 4.3).

O nome associado a cada instância deve ser único e corresponde ao identificador que sucede a palavra reservada INSTANCE. Quanto ao identificador que indica o tipo do módulo, ele deve corresponder ao nome de um módulo da linguagem LPM, previamente compilado.

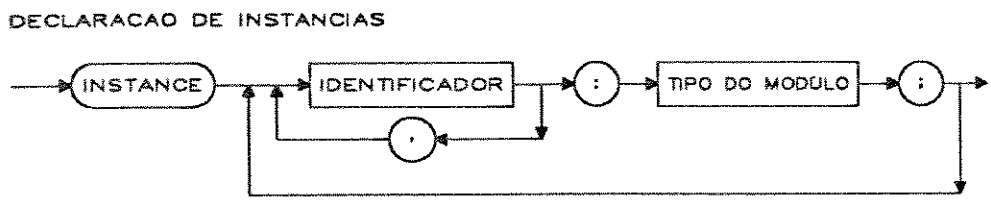

Figura 4.3 : Declaração de Instâncias

4.1.2.4 Criação de Instâncias de Módulos

A criação de instâncias possui sintaxe e semântica equivalentes às descritas na versão LCM centralizada.

Na LCM, os módulos são vistos como tipos, a partir dos quais várias intâncias podem ser criadas (figura 4.4).

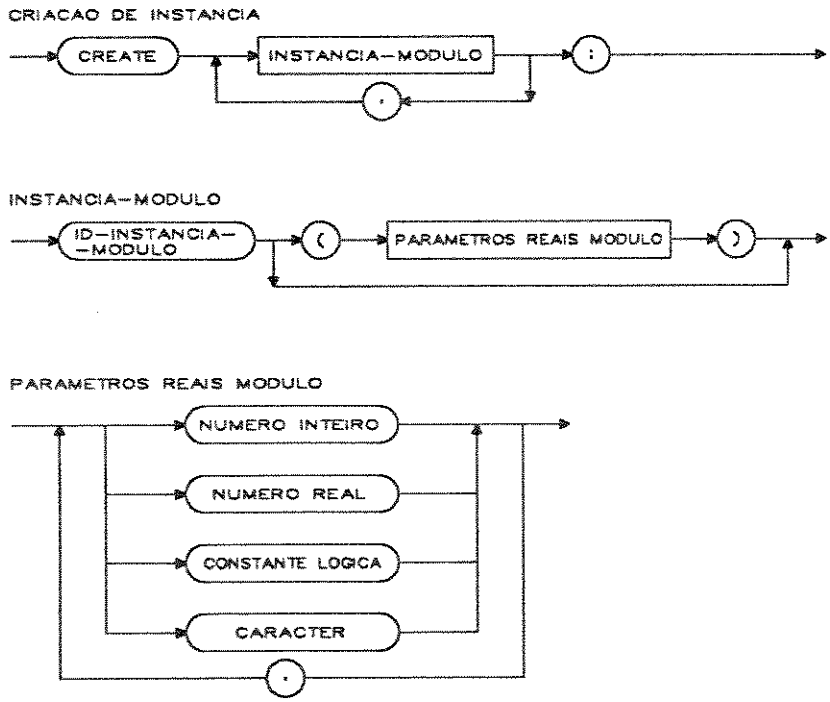

Figura 4.4 : Criação de Instâncias

Caso a especificação LPM de um módulo possua parâmetros formais, os parâmetros reais correspondentes devem ser especificados na criação das intâncias.

Na criação de instâncias é permitido ao usuário a especificação de algumas diretivas de configuração, para definição da prioridade inicial da instância e a área de memória reservada para pilha e "heap". Muito embora as diretivas não estejam definidas no diagrama sintático da figura 4.4, elas são definidas como:

/P é usada para definir uma prioridade inicial de um módulo;

/S é usada para definir a área reservada para pilha;

/H para definir a área reservada para "heap".

4.1.2.5 Conexão de Instâncias de Módulos

A conexão de instâncias possui sintaxe e semântica equivalentes às descritas na versão LCM centralizada.

Para que um módulo possa se comunicar com outro, é necessário que exista uma conexão lógica entre eles.

A figura 4.5 mostra o diagrama sintático do comando LCM, que faz conexões lógicas através de uma ligação direcionada de uma porta de saída para uma porta de entrada.

Os seguintes tipos de conexão são permitidos:

a) Uma porta de saída a uma porta de entrada, permitido para portas assíncronas e síncronas;

b) Uma porta de saída a várias portas de entrada, permitido somente para portas assíncronas;

c) Várias portas de saída a uma porta de entrada, permitido para portas assíncronas e síncronas.

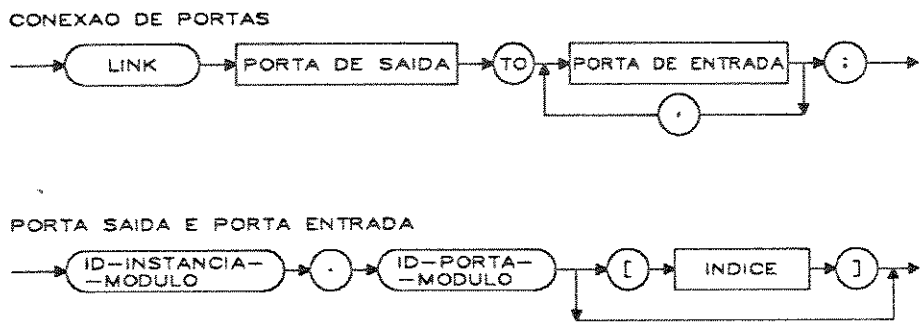

Figura 4.5 : Conexão de Instâncias

4.1.2.6 Associação de Portas à Interface da Estação

Portas internas dos módulos podem constituir elementos da interface da estação, através de uma diretiva que associa uma porta de uma instância de módulo (entrada/saída) a uma porta componente da

interface da estação (figura 4.6).

A porta interna do módulo e a porta da interface da estação devem ser do mesmo tipo, podendo ser de entrada ou saída, mas, se uma delas for de entrada a outra também deverá ser de entrada, o mesmo valendo para portas de saída.

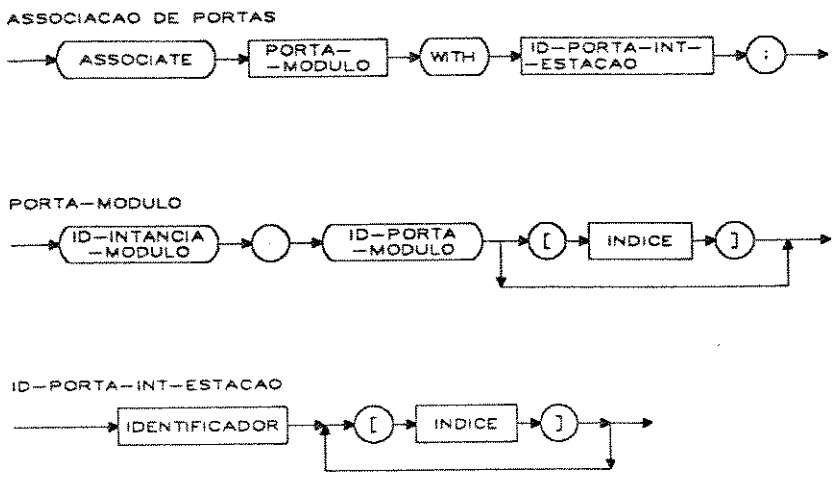

Figura 4.6 : Associação de Portas a Interface da Estação

4.1.2.7 Exemplo da LCMS

A figura 4.7 mostra a configuração de uma instância de estação, no caso a estação ENFERMARIA.

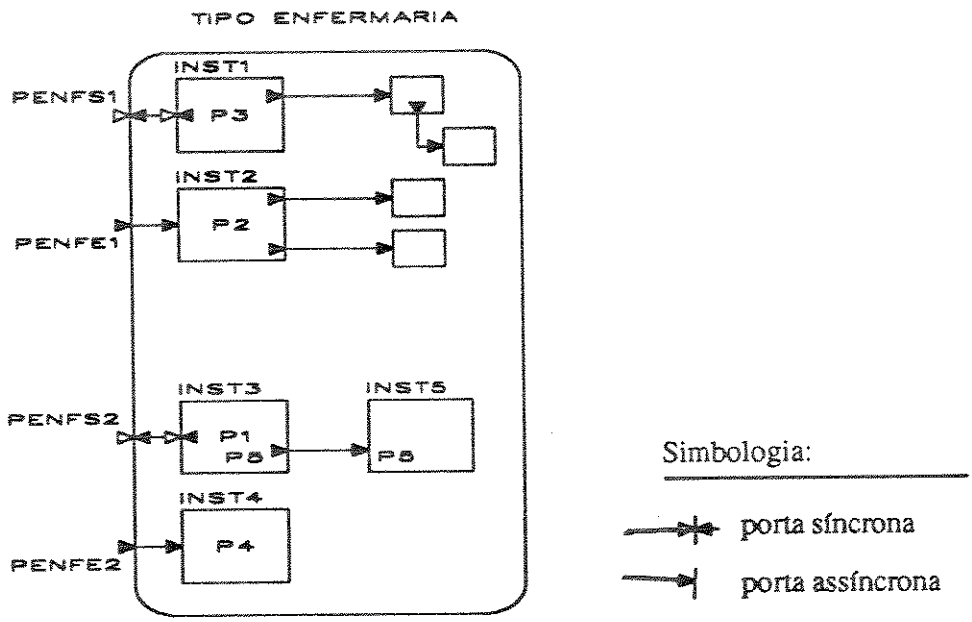

Figura 4.7 : Configuração de Instância de Estação à Nível da Construção STATION

especificada a nível da *Construção STATION*, com seus elementos de interface associados a portas internas dos módulos nas estações. Os módulos sem identificadores, representados na figura 4.7, não pertencem ao exemplo ilustrado na figura 4.8, sendo apresentados para ressaltar a possibilidade de ligações dos módulos pertencentes à interface com outros módulos da estação.

A figura 4.8 corresponde a uma parte da aplicação descrita no capítulo 6 e tem como objetivo apresentar as principais características da linguagem de configuração, relativa à *Construção STATION*, ressaltando-se a semântica correspondente aos elementos de interface e a associação de portas internas dos módulos aos elementos da interface da estação.

```
STATION ENFERMARIA;
        (* elementos de interface de estação *)
        EXITPORT
                Penfs1: tiposignal REPLY tipoleito;
                Penfs2: tiposignal REPLY tipoleito;
        ENTRYPORT
                Penfe1: tipoalarme;
                Penfe2: tiposignal;
        (* declaração de instâncias de módulos desta estação *)
        INSTANCE
                inst1: pdisp;
                inst2: psel;
                inst3: palarm;
                inst4: pconsole;    inst5: gersaida;
        (* criação de instâncias de módulos desta estação *)
        CREATE
                inst1(600)/S=64/H=2,
                inst2/P=lowpr/S=64/H=2,
                inst3(1),      inst5/S=64/H=2,
                inst4(600)/S=64/H=2;
        (* conexão de instâncias de módulos desta estação *)
        LINK
                inst3.P5     TO    inst5.P5;
        (* associação de portas a interface da estação *)
        ASSOCIATE
                inst1.P3  WITH  Penfs1;
                inst2.P2  WITH  Penfe1;
                inst3.P1  WITH  Penfs2;
                inst4.P4  WITH  Penfe2;
END_STATION.
```

Figura 4.8 : Linguagem de Configuração Nível Construção STATION

4.1.3 A Construção NETWORK

A configuração de uma aplicação distribuída é especificada, em nível de rede, através da *Construção* ***NETWORK***, que consiste em:

a) declarar as instâncias das estações componentes;

b) criar as instâncias dessas estações, fornecendo as informações relativas ao endereço físico da estação, à rede de comunicação empregada, ao número de comunicações que podem ser suportadas em paralelo pelo **SCR**;

c) especificar as interconexões entre estações, através das ligações das portas constituintes das interfaces das estações.

O processamento de uma *Construção* ***NETWORK*** produz vários arquivos (tabelas de configuração) para cada estação especificada na LCMN, correspondendo ao tipo de rede, que deverão ser transportados e carregados nas estações físicas correspondentes. O transporte, na versão atual, é feito manualmente para cada estação da rede. Uma extensão seria a carga remota das estações, através da rede de comunicação.

4.1.3.1 Estrutura de um Programa de Configuração para a Construção NETWORK (LCMN)

A figura 4.9 apresenta a estrutura de um programa de configuração a nível da *Construção NETWORK*.

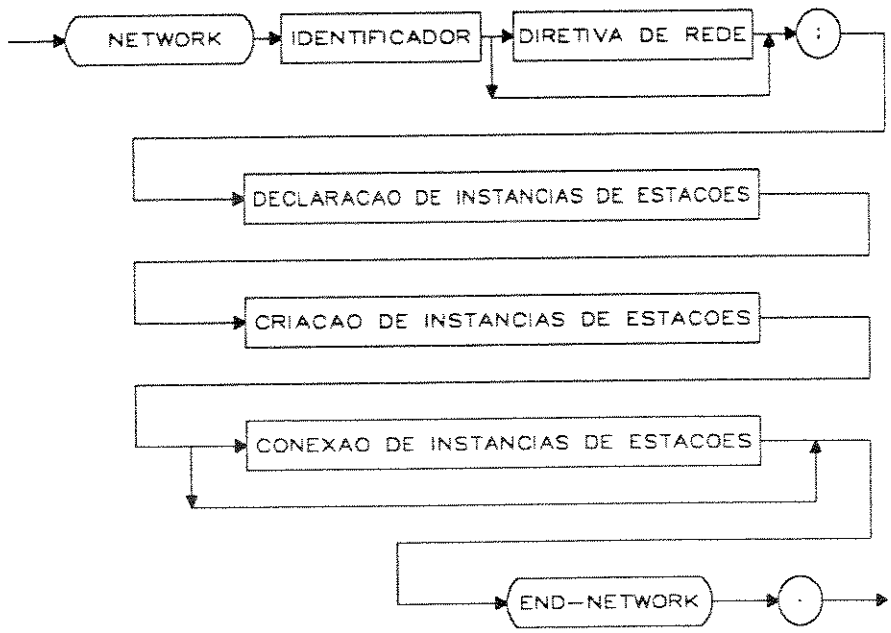

Figura 4.9 : Estrutura de um Programa LCMN

4.1.3.2 Diretiva de Rede

A diretiva especifica a rede de comunicação local a ser utilizada pela aplicação, incorporando os módulos da rede escolhida aos módulos da aplicação distribuída, em tempo de carga dos módulos pelo NOD. O seu objetivo é permitir ao usuário optar por uma das redes disponíveis, supondo que a implementação tem várias opções de rede de comunicação em bibliotecas. A figura 4.10 ilustra o diagrama sintático da diretiva de rede.

A não especificação dessa diretiva indica que o usuário não quer utilizar as redes disponíveis; possívelmente, neste caso, o usuário utilizará como sistema de comunicação os seus próprios módulos de rede incluídos aos módulos da sua aplicação.

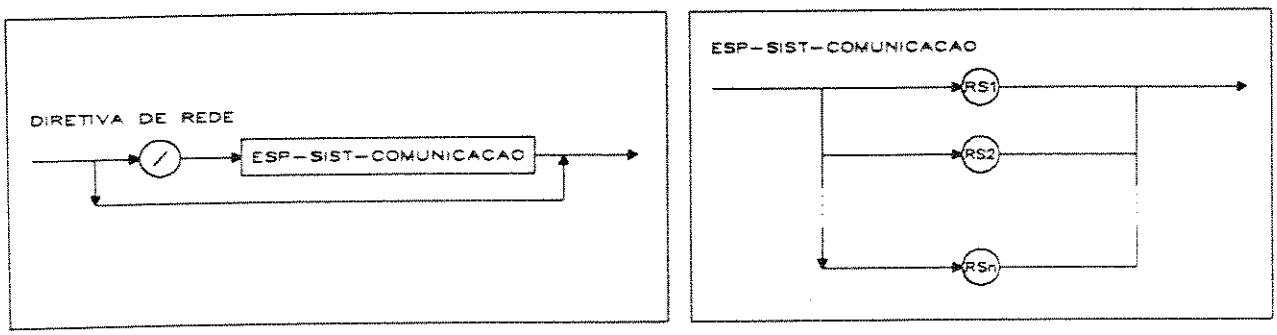

Figura 4.10 : Diretiva de Rede

4.1.3.3 Declaração de Instâncias de Estações

Nesta declaração são associados nomes às várias instâncias de estações da aplicação. Os mapas de configuração a serem carregados nas estações físicas de hardware, para a execução da aplicação, terão os mesmos nomes.

A sintaxe, conforme mostra a figura 4.11, é similar à declaração de instâncias na *Construção STATION*, sendo o tipo da estação um identificador do arquivo gerado pela *Construção STATION*.

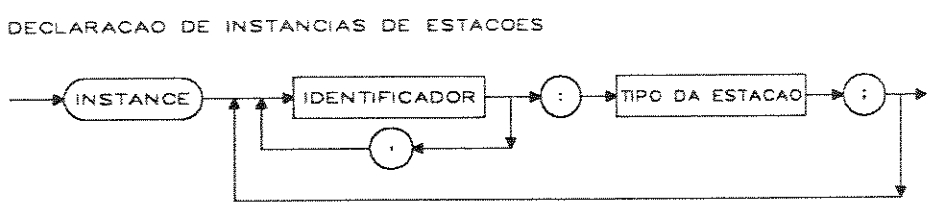

Figura 4.11 : Declaração de Instâncias de Estações

4.1.3.4 Criação de Instâncias de Estações

A estrutura sintática apresentada na figura 4.12, é usada para criar as instâncias de estações que compõem a rede.

Nesta construção, o usuário deverá fornecer o endereço físico da estação na rede através do número # <inteiro decimal> , bem como poderá estabelecer o número de buffers da rede, que correspondem ao número de comunicações que podem ser suportadas em paralelo pelo SCR. Caso esse número não seja especificado, o valor padrão característico da versão de implementação, será assumido.

O inteiro decimal definido para endereço de estação deve ser, no máximo, o valor configurado para a versão.

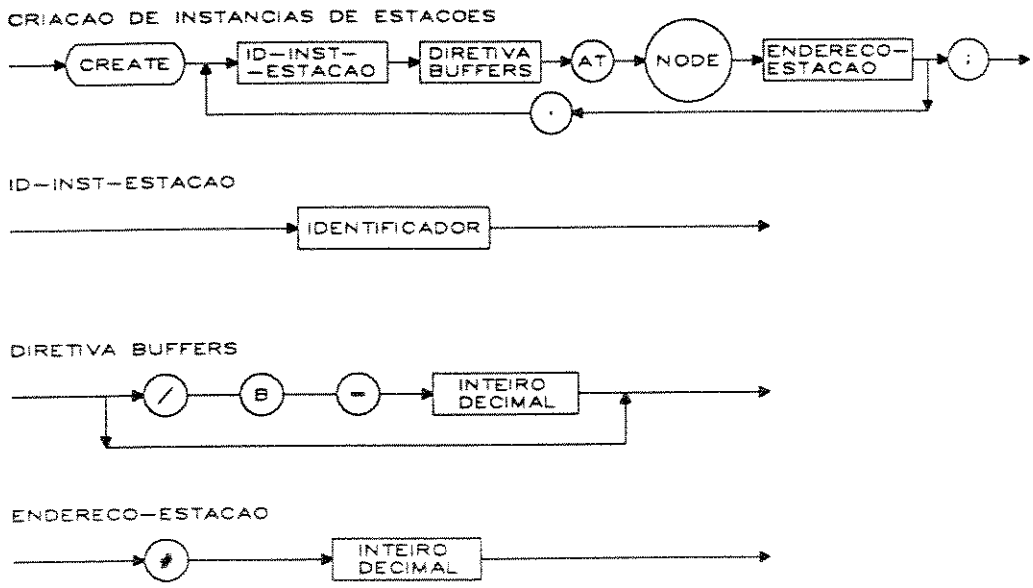

Figura 4.12 : Criação de Instâncias de Estações

4.1.3.5 Conexão de Instâncias de Estações

A sintaxe do comando que realiza as conexões entre as estações é similar às conexões realizadas na *Construção STATION*.

A figura 4.13 ilustra o diagrama sintático do comando "LINK".

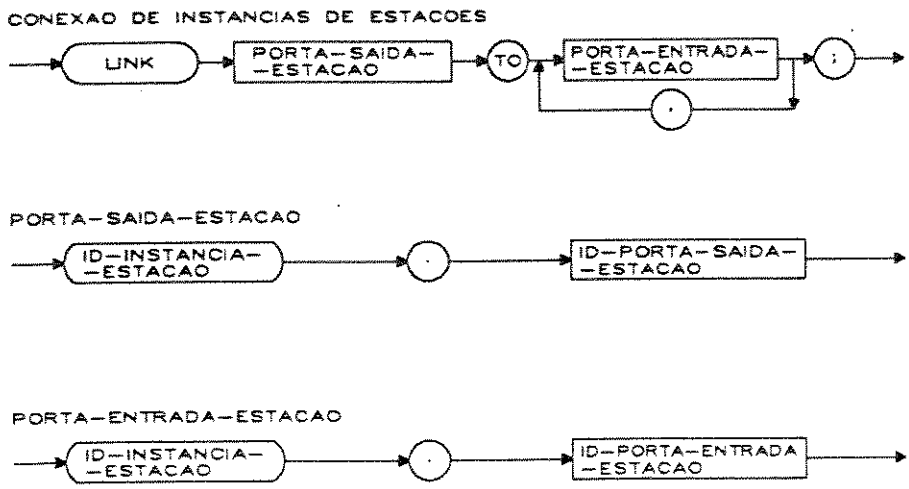

Figura 4.13 : Conexão de Instâncias de Estações

4.1.3.6 Exemplo de LCMN

A figura 4.14 mostra a configuração, a nível NETWORK, de uma aplicação distribuída, com três instâncias de estações interconectadas, através das ligações entre suas portas de interface.

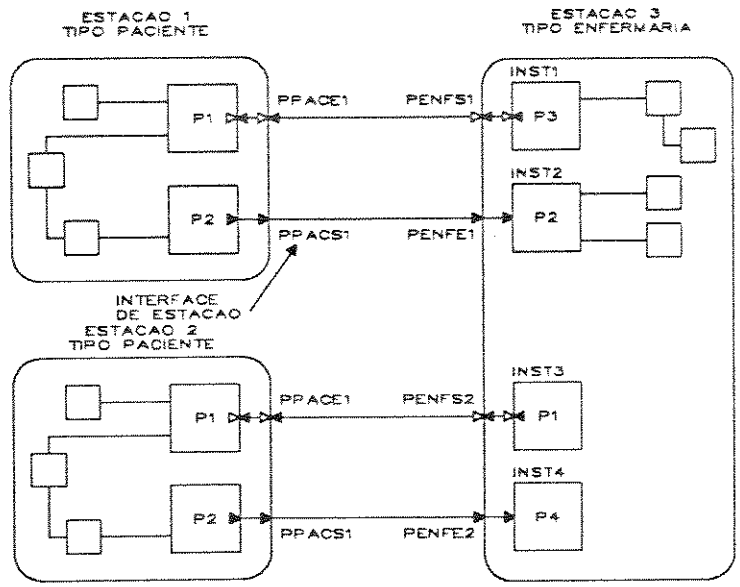

Figura 4.14 : Configuração de uma Aplicação Distribuída

Os módulos sem identificadores, representados nas estações, são apresentados para ressaltar a possibilidade de ligações dos módulos pertencentes à interface com outros módulos da estação.

A figura 4.15 corresponde a uma parte da aplicação descrita no capítulo 6, tendo como objetivo apresentar a sintaxe da *Construção NETWORK* e o mapeamento da configuração lógica para a configuração física, bem como a interconexão das várias estações, através das ligações entre portas.

```
(* especificação de rede de comunicação local : RS2 *)

NETWORK rede / RS2;

(* declaração de instâncias de estações *)
        INSTANCE
                estacao1 : PACIENTE;
                estacao2 : PACIENTE;
                estacao3 : ENFERMARIA;
(* criação de instâncias de estações *)
        CREATE
                estacao1  AT NODE # 1,
                estacao2  AT NODE # 2,
                estacao3  AT NODE # 3;
(* conexão de instâncias de estações *)
        LINK
                estacao3.Penfs1  TO  estacao1.Ppace1;
                estacao3.Penfs2  TO  estacao2.Ppace1;
                estacao1.Ppacs1  TO  estacao3.Penfe1;
                estacao2.Ppacs1  TO  estacao3.Penfe2;

END_NETWORK.
```

Figura 4.15 : Linguagem de Configuração Nível Construção NETWORK

4.2 Geração de um Programa de Aplicação Distribuído

A geração de um programa de aplicação para o ambiente STER distribuído compreende duas etapas: a primeira consiste na compilação independente dos módulos construídos em LPM e a segunda na configuração escrita em LCM da aplicação a partir dos módulos disponíveis.

4.2.1 Compilação de um Módulo da Aplicação

A compilação de um módulo LPM (figura 4.16) é descrita a seguir.

O programa fonte do módulo escrito pelo usuário (M1.LPM) é submetido ao pré-processador LPM. A implementação do pré-processador LPM consiste de dois componentes: o processador de unidades de definição e o pré-compilador LPM.

Antes de ser referenciada por um comando LPM, uma unidade de definição deve ser previamente compilada pelo processador de unidade de definição, cuja função é verificar a especificação fornecida, obter os tamanhos correspondentes aos tipos das mensagens e gerar os arquivos: unidade.TIP com os tamanhos dos tipos correspondentes às mensagens e unidade.INC com os tipos a serem incluídos por um comando LPM de importação de tipos.

O pré-compilador LPM converte o programa fonte em LPM em um programa fonte em linguagem PASCAL (M1.PAS) e em uma estrutura de dados chamada qualificador do módulo (M1.QLF). As informações contidas no qualificador de um módulo descrevem as portas de entrada e saída do módulo pré-compilado e os tipos dessas portas, de modo a permitir testes de consistência durante a fase de configuração.

A implementação do pré-compilador LPM possui como estratégia básica a cópia direta, para o arquivo de saída, dos comandos e declarações do PASCAL encontradas no corpo do módulo. As extensões à linguagem PASCAL, compostas basicamente dos comandos e declarações LPM (SELECT, SEND, RECEIVE, DELAY, INTALLOC, WAITIO, declarações de portas e mensagens, etc.), são traduzidas pelo pré-compilador LPM em chamadas a serviços do STRE, representados como procedimentos no programa PASCAL resultante da pré-compilação, e em estruturas de dados a serem utilizadas na fase de configuração. Durante a fase de pré-compilação são também realizados testes de consistência com o objetivo de detetar erros na programação dos módulos.

O código objeto (M1.OBJ), resultante da compilação do programa PASCAL gerado, será submetido ao ligador que gerará o código executável do módulo (M1.EXE). A figura 4.16 ilustra o código objeto do módulo ligado à vários outros arquivos objeto a fim de gerar o código executável do módulo.

O arquivo INIM.OBJ contém uma série de iniciações que antecedem a execução do módulo, tais como a iniciação do mapa de memória do módulo e o contexto inicial do módulo. O arquivo ISTR.OBJ contém as rotinas de interfaceamento do módulo ao núcleo STRE. O arquivo *.OBJ representa um número qualquer de rotinas escritas em outras linguagens (linguagens de montagem, FORTRAN, C, etc.) e declaradas como externas no módulo em LPM (M1.LPM). O arquivo PASCAL.LIB contém a biblioteca PASCAL.

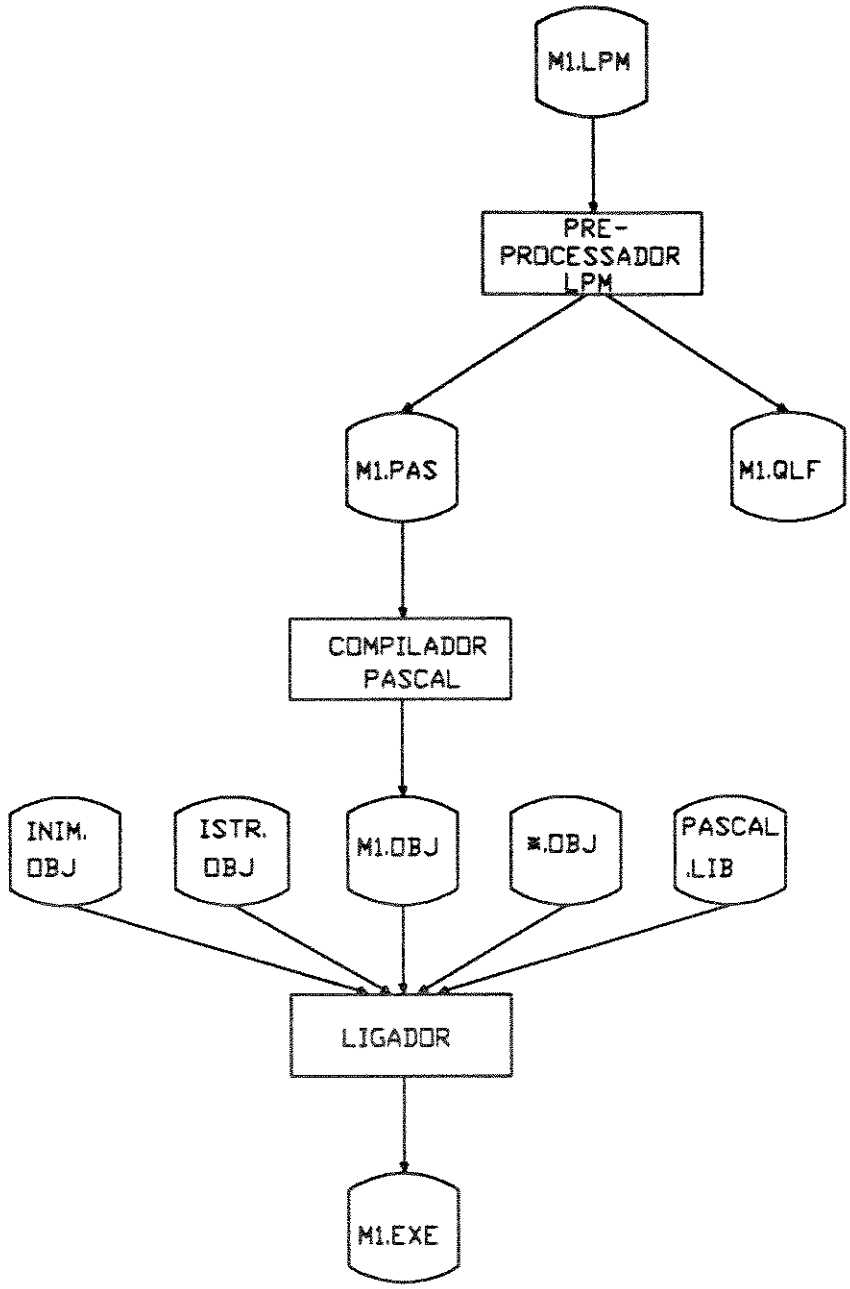

Figura 4.16 : A Compilação de um Módulo

4.2.2 Configuração de uma Aplicação Distribuída

A etapa de configuração de uma aplicação distribuída é construída através da linguagem LCM e pressupõe que todos os módulos envolvidos, escritos em LPM, já foram previamente compilados, conforme descrito no item anterior.

A especificação de uma configuração distribuída consiste na programação de dois níveis independentes: o primeiro nível corresponde a um programa escrito em LCMS (Construção STATION) e o segundo nível corresponde a um programa escrito em LCMN (Construção NETWORK).

4.2.2.1 A Configuração de uma Aplicação a Nível STATION

A Construção STATION, suportada pela ferramenta LCMS, constitui uma estruturação lógica de um tipo de estação, a partir do qual pode-se criar várias instâncias, durante a Construção NETWORK.

O Tradutor LCMS processa um programa de configuração que especifica um tipo de estação com suas portas de interface, os tipos de módulos e instâncias componentes da estação, as características particulares que cada instância apresentará e o relacionamento existente entre as instâncias via a interligação local de suas portas. As características particulares de cada instância de módulo consistem da especificação do valor de seus parâmetros reais; da quantidade de memória alocada para "heap" e pilha; e da sua prioridade inicial.

Conforme ilustra a figura 4.17, o programa de configuração a nível STATION é especificado através de um programa fonte em linguagem LCMS (E1.LCM). A partir desse programa fonte e dos qualificadores dos módulos envolvidos (Mn.QLF), o tradutor LCMS obtém as informações necessárias à geração de um arquivo descritor de estação (E1.CFS), que é utilizado durante o processamento da Construção NETWORK.

À medida que o reconhecimento da especificação é processado, testes de consistência são realizados. Os parâmetros reais especificados no programa LCM serão válidos se compatíveis com os parâmetros formais definidos no módulo LPM. Uma conexão será válida se for orientada de uma porta de saída a uma porta de entrada do mesmo tipo.

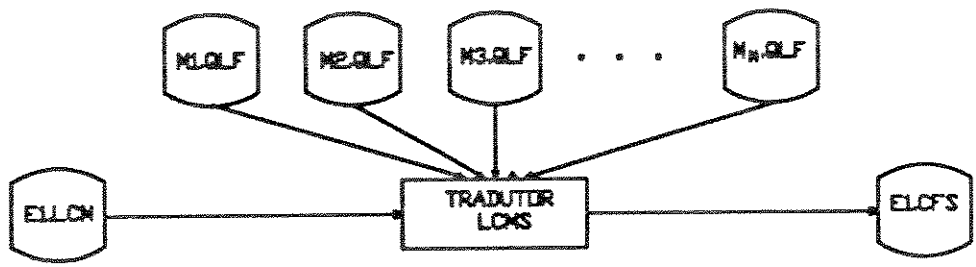

Figura 4.17 : Configuração de uma Aplicação Nível STATION

4.2.2.2 A Configuração de uma Aplicação a Nível NETWORK

A configuração a nível NETWORK de uma aplicação pressupõe que todos os tipos de estações envolvidos, escritos em LCM, já foram previamente processados pelo tradutor LCMS.

A Construção NETWORK, suportada pela ferramenta LCMN, permite que tipos de estações sejam instanciados e interconectados a outras instâncias de estações, produzindo uma tabela de configuração específica para cada estação lógica pertencente à rede de comunicação. Essa tabela de configuração consiste de uma tabela estruturada, que contém as informações requisitadas pelo núcleo (STRE) para efetivar a configuração física da aplicação.

O tradutor LCMN processa um programa de configuração que especifica as instâncias de estações componentes da rede; as informações quanto ao endereço físico da rede; o número de buffers utilizados na rede de comunicação; e as interconexões entre as estações através das ligações de portas de interface das estações.

Conforme ilustra a figura 4.18, o programa de configuração a nível NETWORK é especificado através de um programa fonte em linguagem LCMN (REDE.LCM). A partir desse programa fonte e dos arquivos descritores dos tipos de estação envolvidos (En.CFS), o tradutor LCMN obtém as informações necessárias para a geração da tabela de configuração, gerada em arquivos específicos para cada estação pertencente a rede, que definem a imagem da estação a ser configurada (ESTn.CNF).

À medida que o reconhecimento da especificação é processado, testes de consistência são realizados. Esses testes consistem em verificar se as conexões especificadas correspondem a portas do mesmo tipo e se a conexão é orientada de uma porta de saída a uma porta de entrada.

As tabelas de configuração geradas são passadas ao núcleo NOD de cada estação física da rede, quando se desejar executar a aplicação. A execução da aplicação é precedida pelas iniciações realizadas pelo NOD, que varre a tabela de configuração, onde obtém as informações relativas às estações componentes, criando dinamicamente as estruturas de dados necessárias e realizando a carga da aplicação. Esse procedimento é descrito no item 5.2.2. Nesse momento, tem-se a imagem executável da aplicação correspondente à configuração física especificada, em cada computador da rede. O módulo de maior prioridade, especificado pelo usuário, recebe o controle do NOD e inicia a sua execução.

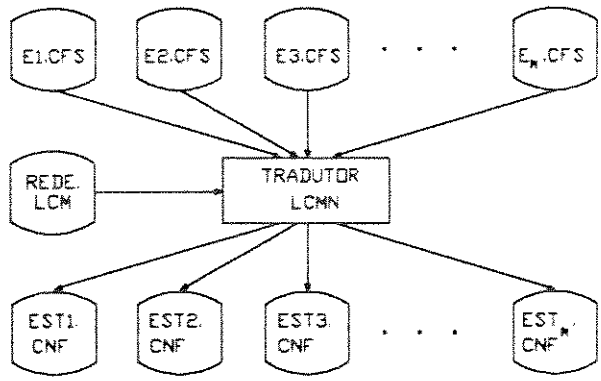

Figura 4.18 : Configuração de uma Aplicação Nível NETWORK

Capítulo 5

Núcleo Operacional Distribuído (NOD)

O NOD oferece suporte à interação e comunicação entre estações de tal forma que o resultado de um processamento em uma estação possa interferir, de uma maneira controlada e desejada, nos resultados de processamento em outras estações.

Cada processador, em uma estação da rede, executa as instâncias de módulos armazenadas na sua memória local, de forma independente do que ocorre nos processadores de outras estações.

Cada processador de uma estação executa uma cópia privativa do código do **NOD** armazenada em sua memória local, durante a carga do **SDS**.

A figura 5.1 apresenta o **NOD** de cada estação que é caracterizado como uma máquina composta de uma série de camadas ou níveis de abstração. Cada camada implementa uma máquina virtual, acrescentando novas abstrações ao conjunto oferecido pelas camadas sobre as quais é construída.

A primeira camada, Camada de Funções Básicas, corresponde ao suporte da versão centralizada do ambiente **STER**. A segunda camada corresponde a uma extensão da camada **STR** (Núcleo de Suporte de Tempo Real Local) da versão centralizada do ambiente **STER** e denominada, a partir deste ponto, de **STRE** (STR Estendido). A terceira "camada" é formada pelo Subsistema de Comunicação Remota entre Módulos (**SCR**).

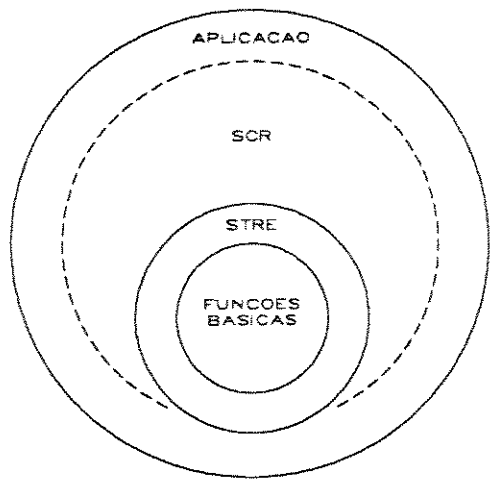

Figura 5.1 : Estrutura em Camadas do NOD

Os itens a seguir discutem as características das camadas de **Funções Básicas,STRE e SCR.**

5.1 Funções Básicas

A Camada de Funções Básicas consiste do Suporte de Tempo Real (STR) [1] que implementa os serviços que atendem, em tempo de execução, aos comandos, procedimentos e funções pré-declarados da LPM que não fazem parte do PASCAL padrão. Inclui, também, os procedimentos que correspondem à implantação do STR em uma arquitetura específica de hardware. Os serviços são implementados em PASCAL e compartilham as estruturas de dados que representam os objetos da aplicação (módulos, portas). Outros procedimentos, em menor porcentagem, são implementados em Assembler e realizam funções que dependem diretamente do hardware.

Convém lembrar que cada serviço do STR não é diretamente acessível ao usuário; este faz a programação dos seus módulos usando as construções da LPM apresentadas no Apêndice A. As chamadas aos serviços do STR, apresentadas no Apêndice B, são geradas pelo pré-processador da LPM como resultado da tradução dos módulos fonte em LPM para programas fonte em PASCAL.

A organização do STR é composta de serviços que pertencem à seguinte estrutura:

- Gerência de Módulos;
- Gerência de Tratamento de Interrupções;
- Comunicação e Sincronização entre Módulos através de Troca de Mensagens;
- Gerência de Temporização;
- Gerência de Memória;
- Gerência de Exceções.

Os principais serviços oferecidos nesta estrutura são descritos a seguir.

5.1.1 Gerência de Módulos

- Implementa a política de escalonamento preemptivo por prioridades. A preempção por prioridade é a política mais comum nas aplicações de tempo real ("SOFT REAL TIME"), procurando assegurar um atendimento rápido dos módulos tratadores de situações urgentes;

- Serviço de Mudança de Prioridade de um Módulo
(Comando em LPM : SETPRIORITY).

5.1.2 Gerência de Tratamento de Interrupções

Em aplicações de controle de tempo real a ocorrência de eventos assíncronos é uma das atividades mais importantes. O STR fornece o suporte à LPM para manipular a ocorrência de interrupções em alto nível através dos serviços:

- Mapeamento de um elemento do vetor físico de interrupções em um elemento do vetor lógico de interrupções
  (Comando em LPM : INTALLOC);
- Espera pela ocorrência de uma interrupção
  (Comando em LPM :WAITIO);
- Verificação da ocorrência de interrupção lógica
  (Comando em LPM : OCORREINT);
- Reiniciação do indicador de ocorrência de uma interrupção lógica
  (Comando em LPM : LIMPAINT);
- Dealocamento de interrupção lógica
  (Comando em LPM : DEALOCINT);

5.1.3 Comunicação e Sincronização entre Módulos através de Troca de Mensagens

Os módulos do sistema se comunicam com outros módulos através da troca de mensagens nas portas de entrada e saída pertencentes aos módulos.

A sintaxe dos serviços de comunicação e sincronização pode ser observada nos apêndices A e B.

Os serviços disponíveis nesse contexto são:

- Envio Assíncrono de uma Mensagem;
- Envio Síncrono de uma Mensagem;
- Envio Síncrono de uma Mensagem com cláusula para Tratamento de Falha;
- Recepção Bloqueante de uma Mensagem;
- Envio de uma Mensagem de Resposta;
- Desvio de uma Comunicação Síncrona;
- Recepção Seletiva de uma Mensagem (priorizada e randômica);
- Cancelamento de uma Comunicação Síncrona;
- Teste de Conexão de uma Porta de Saída;
- Determinação da Quantidade de Mensagens enfileiradas em uma Porta de Saída;
- Determinação da Razão de uma Falha.

5.1.4 Gerência de Temporização

O STR mantém um contador de tempo (relógio) usado para detetar o fim dos períodos de espera dos módulos em função de um pedido de retardo ("DELAY") na execução de um módulo ou, ainda, na execução de comunicações com cláusulas de temporização.

Os serviços disponíveis envolvendo temporização são:

- Leitura do Relógio do STR
  (Comando em LPM : TIME);
- Retardamento de um Módulo
  (Comando em LPM : DELAY).

5.1.5 Gerência de Memória

O Gerenciador de Memória implementado é simples e utiliza um alocador de memória dinâmica que usa a memória em função das necessidades do procedimento requisitante. Do modo como está implementado nesta versão, o gerenciador não permite que se faça a dealocação de memória, o que, neste momento, não é necessário. No caso do ambiente STER vir a incorporar o conceito de reconfiguração dinâmica da aplicação, será necessário melhorar o gerenciador em uso, de forma que este possa fazer a dealocação dinâmica de memória.

5.1.6 Gerência de Exceções

A gerência de exceções implementa o controle das medidas utilizadas quando da ocorrência de um evento que exija a interrupção do ciclo normal de execução.

Exceções podem ser provocadas por falhas de hardware, erros da aplicação (divisão por zero, overflow, etc), interrupção de hardware (timer,I/O), interrupção por software (trap).

Para informar a aplicação sobre as exceções, o STR imprime mensagens de erro no terminal.

5.2 STRE - STR Estendido

A extensão do STR da versão centralizada consiste dos itens descritos a seguir:

a) Estrutura de Dados do STRE;

b) Iniciação do STRE;

c) Suporte fornecido pelo STRE ao Subsistema de Comunicação Remota entre Módulos (SCR).

5.2.1 Estrutura de Dados do STRE

O STRE contém, basicamente, as mesmas estruturas de dados implementadas no núcleo para suporte ao ambiente STER da versão centralizada. Algumas estruturas de dados foram estendidas e outras criadas com o objetivo de incorporar os conceitos de reentrância de código, estação e rede, para utilização no ambiente STER da versão distribuída.

Essas estruturas são formadas de uma maneira hierárquica, conforme ilustra a figura 5.2, em cada uma das estações que compõem o SDS. A figura 5.2 ilustra, em particular, as estruturas de dados do núcleo no caso da estação B do SDS.

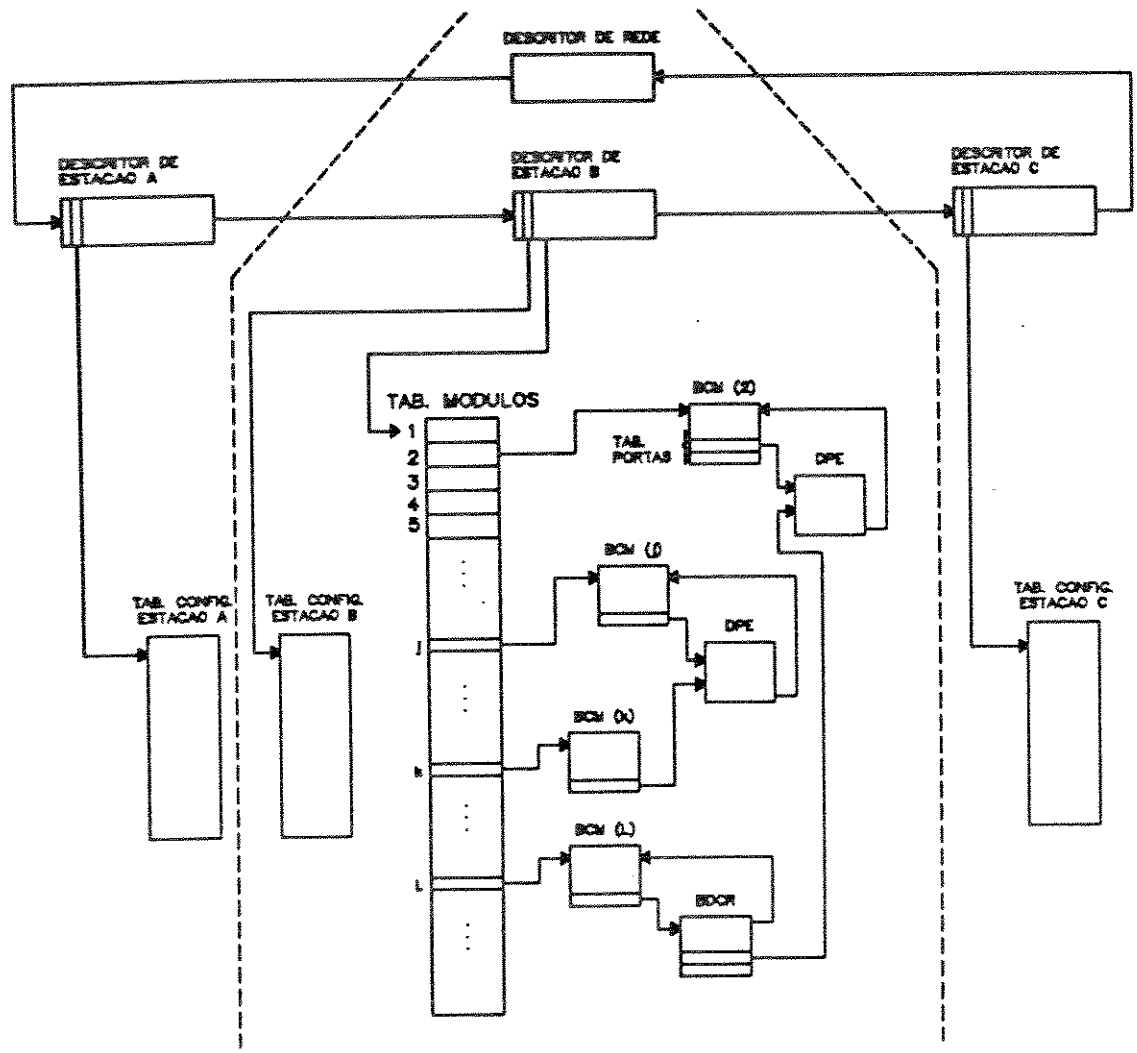

Figura 5.2 : Estruturas de Dados do Núcleo (ESTAÇÃO B)

A *Estrutura Descritor de Rede* corresponde à estrutura geral de uma aplicação distribuída na rede, e contém um apontador para uma lista de estruturas descritoras de cada estação da rede.

A *Estrutura Descritor de Estação* corresponde a representação de cada estação pertencente à rede que contenha elementos da aplicação distribuída, e contém dados estáticos que descrevem a estação que representa, bem como dados dinâmicos que descrevem o seu estado em um determinado instante.

As informações correspondentes a dados estáticos são as seguintes:

- Identificação da Estação;
- Nome do arquivo que contém a Tabela de Configuração da estação;
- Tipo da Estação: estação local ou estação remota;
- Se a estação é local:
  - Apontador para uma Tabela de Apontadores aos BCMs (TABELA de MÓDULOS), correspondentes às instâncias de módulos definidos pertencentes a estação;

— identificação de cada estação remota conectada a esta estação local;

- Se a estação é remota :

  — Apontador para uma Tabela de Configuração desta estação;

- Apontador para a próxima estrutura de estação.

Exemplos de dados dinâmicos são:

- Estado da estação: ativa ou não ativa;
- Tempo que o módulo Gerente de Comunicação deve esperar até avisar que a estação não está ativa;
- Contador indicando o número de vezes que a estação falhou durante as comunicações através da rede.

A *Estrutura da Tabela de Módulos* reúne os apontadores aos BCMs das instâncias de módulos, definidas pela aplicação. Esta estrutura permanece à mesma da versão centralizada.

Cada instância de módulo é descrita por um *Bloco de Contexto de Módulo* (BCM). Um BCM contém dados estáticos e dados dinâmicos, alguns dos quais foram acrescentados para suporte à implementação de reentrância de código.

Exemplos de dados estáticos relativos a uma instância de módulo são os seguintes:

- Identificação do módulo;
- Nome do módulo;
- Quantidade de memória usada para pilha e heap;
- Apontador a sua base de dados;
- Identificação do pai do módulo;
- Apontador para a estrutura de código do módulo;
- Tamanho de seu segmento de dados;
- Valor de seus parâmetros reais;
- Descrição de suas portas.

Exemplos de dados dinâmicos são os seguintes:

- O valor da prioridade do módulo;
- O tempo que o módulo deve esperar para passar ao estado de pronto, quando suspenso.

A *Estrutura de Código* do módulo foi construída para suporte a informações:

- Endereço do início do código do módulo;
- Tamanho do código;
- Contador de referências ao código.

A *Estrutura da Tabela de Portas* de um BCM contém, em cada posição, um apontador ao descritor da porta de número correspondente à sua posição na tabela. Assim, na primeira posição da tabela de portas,

há um apontador ao descritor da porta de número um; na segunda posição, há um descritor da porta de número dois e assim por diante.

Se uma dada porta for de entrada, na posição que lhe corresponde na tabela de portas, haverá um apontador a uma estrutura *Descritor de Porta de Entrada* (DPE) que representará a porta. Caso uma porta de entrada tenha um buffer, este será parte de seu DPE.

Se uma dada porta for de saída haverá duas opções:

- Se a porta de saída pertencer a um módulo envolvido em uma comunicação local, na posição que lhe corresponde na tabela de portas, haverá um apontador ao DPE da porta de entrada à qual está conectada.
- Se a porta de saída pertencer a um módulo envolvido em uma comunicação remota, na posição que lhe corresponde na tabela de portas, haverá um apontador ao Bloco Descritor de Conexões Remotas (BDCR).

A *Estrutura Bloco Descritor de Conexões Remotas* corresponde a estrutura, através da qual é criada uma interface transparente, entre o Módulo da Aplicação e o Subsistema de Comunicação Remota, durante a fase de iniciação do sistema, contendo as seguintes informações:

- Identificação da estrutura BDCR;
- Endereço de sistema da estação remota;
- Endereço de sistema da estação local;
- Apontador ao DPE da porta de entrada de um módulo do SCR (Subsistema de Comunicação Remota entre Módulos).

As estruturas de dados do STRE e do STR foram projetadas de maneira a minimizar o tempo envolvido na sua manipulação, o que é muito importante para sistemas de tempo real, pois favorece a implementação de serviços com tempo pequeno de resposta. Essas estruturas foram projetadas pensando-se em futuras extensões do ambiente, tais como, a carga das estações pela rede de comunicação e reconfiguração dinâmica.

Os nomes lógicos das estações, módulos e portas foram mapeados em identificadores inteiros durante as fases de compilação e de configuração dos módulos da aplicação, assim contribuindo para otimizar o acesso às estruturas de dados que representam essas entidades.

5.2.2 Iniciação do STR Estendido

Após a fase de configuração da aplicação , gerada pela LCM, resultam várias tabelas de configuração que definem o relacionamento existente entre as várias estações componentes da rede, bem como as características que particularizam cada uma delas.

Durante a etapa de iniciação das estações, e antes da carga dos módulos da aplicação, é carregada a tabela de configuração específica para cada estação pertencente à rede.

O STRE inicia as suas estruturas de dados , descritas no item 5.2.1, através das informações obtidas na tabela de configuração carregada na sua memória.

A estrutura da tabela de configuração é mostrada na figura 5.3. O descritor de rede (D-REDE) dará origem a *Estrutura Descritor de Rede*.O descritor de estação (D-ESTAÇÃO) dará origem a *Estrutura Descritor de Estação*, alocada dinamicamente e iniciada a partir dos dados contidos no descritor de estação. Cada descritor de módulo (D-MÓDULO) dará origem a um *Bloco de Contexto de Módulo* (BCM),

e a um número de *Descritores de Portas de Entrada* (DPE) igual à quantidade das portas de entrada pertencentes ao módulo (D-PRT-ENT). Os BCM's e DPE's são alocados dinamicamente e iniciados a partir dos dados contidos no descritor de módulo e no descritor de porta de entrada.

O descritor de conexão, da tabela de configuração, corresponde à representação de uma conexão local (D-CONEXÃO) ou de uma conexão remota (D-CONREDE). Ainda com relação à versão centralizada, a referência a uma porta é acrescida do identificador de estação, gerando a tupla:

(número de estação, número de módulo, número de porta),

correspondendo ao endereço de sistema da porta no STRE.

Cada descritor de conexão representa a ligação de uma porta de saída a uma porta de entrada. Um descritor de conexão no **STRE** é representado através da tupla:

[(IdEstação.IdMódulo.IdPorta),(IdEstação.IdMódulo.IdPorta)]

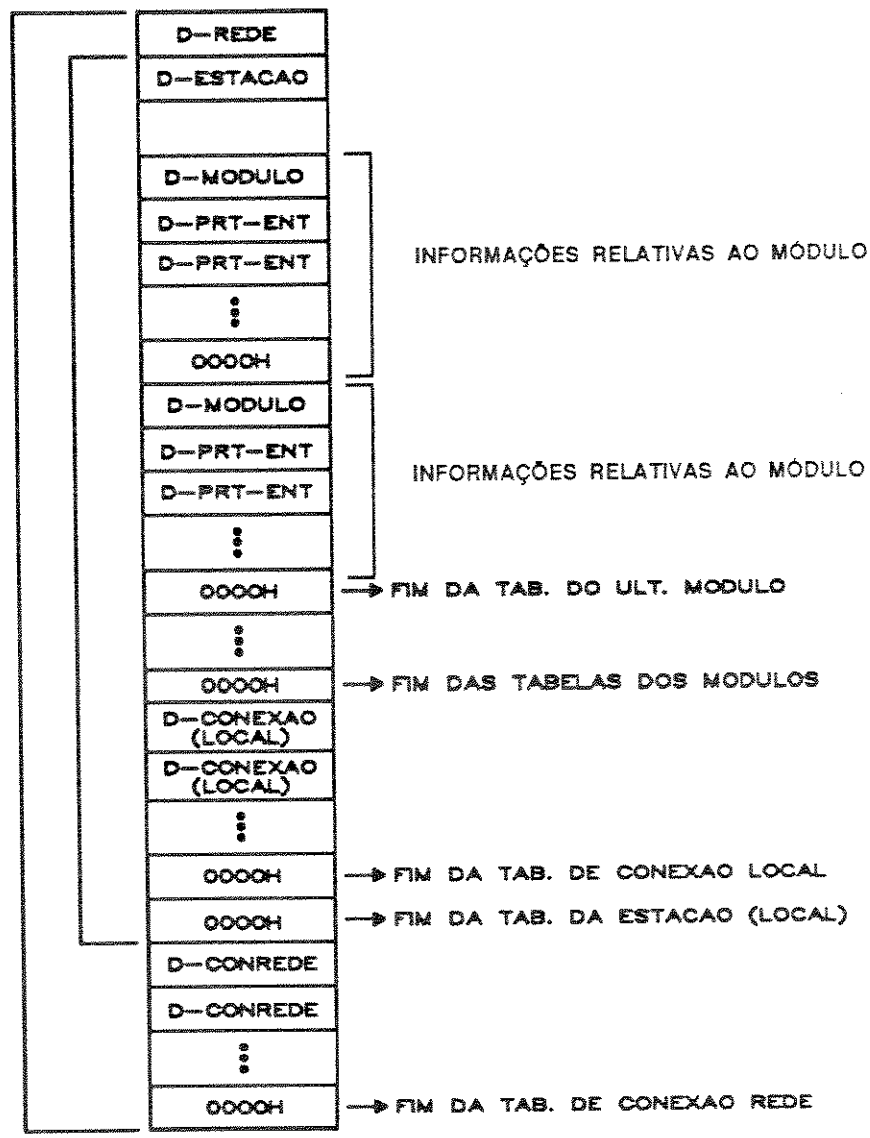

Figura 5.3 : Tabela de Configuração para uma Estação

O primeiro elemento identifica uma porta de saída origem e o segundo, uma porta de entrada destino local ou remota à qual a porta de saída está ligada.

A partir da informação acima,o STRE gera as estruturas de dados, que representam as conexões local e remota.

Quando uma porta de saída for ligada a uma porta de entrada local, os campos *IdEstação* dos endereços das portas têm o mesmo valor, e assim, são criadas as estruturas de dados relacionadas à comunicação local. A partir desse descritor de conexão, será colocado na posição correspondente à porta de saída do módulo origem da conexão um apontador ao *Descritor de Porta de Entrada* (DPE) da porta de entrada de destino.

Quando uma porta de saída for ligada a uma porta de entrada remota, os campos *IdEstação* dos endereços das portas não têm o mesmo valor, e então o STRE gera um *Bloco Descritor de Conexões Remotas*(BDCR) relacionado a conexão remota. A partir do descritor de conexão, será colocado na posição correspondente à porta de saída do módulo origem da conexão, um apontador ao BDCR, alocado dinamicamente, que recebe em seus campos os dois endereços de sistema origem e destino, bem como um apontador ao *Descritor de Porta de Entrada Síncrona* (DPE) da porta de entrada de um módulo do SCR (módulo Emissor), que faz a interface entre o NOD e o Serviço de Comunicação da Rede Local a ser utilizado.

A figura 5.4 ilustra as estruturas de dados para a conexão local e remota de uma aplicação organizada durante a etapa de iniciação de uma estação.

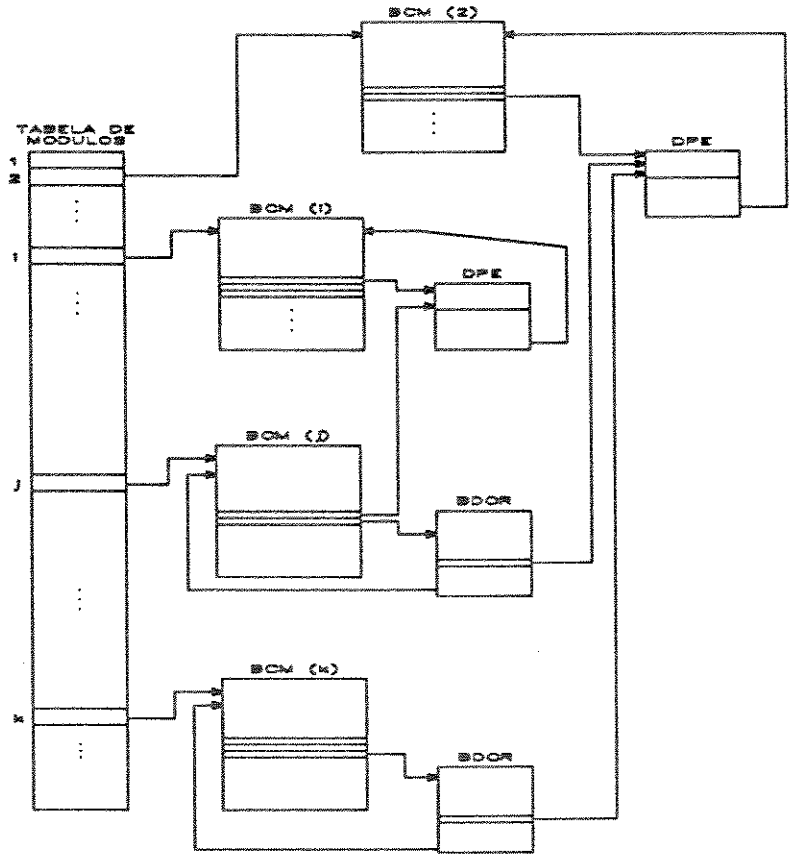

Figura 5.4 : Estruturas de Dados básicas do STRE, ilustrando conexões local e remota. Na conexão local, a porta de saída do módulo j está ligada diretamente ao DPE da porta de entrada do módulo i. Nas conexões remotas, as portas de saída dos módulos j e k estão ligadas ao BDCR.

O **STRE** continua com as iniciações correspondentes ao núcleo centralizado, culminando com a carga dos módulos da aplicação o que atualmente é feito de maneira reentrante. Cada instância de um tipo de módulo (módulo filho) contém somente seu segmento particular de dados enquanto compartilha o segmento de código do seu tipo de módulo (módulo pai).

A carga da aplicação compreende três atividades principais. A primeira consiste na carga do código objeto do módulo pai. A segunda consiste na criação de cada instância a partir de um módulo pai. A terceira consiste na criação do contexto inicial de cada instância de módulo.

A carga do código objeto dos módulos pai é feita pelo MS-DOS em atendimento a uma solicitação feita pelo STRE, que obtém os nomes dos arquivos contendo os módulos a serem carregados nos BCMs criados a partir da tabela de configuração. A carga de um módulo pai é totalmente independente da carga dos demais, sendo cada um visto pelo MS-DOS como a carga de um novo programa.

A criação de cada instância, a partir de um módulo pai, é feita após a carga de cada módulo pai e é executada pelo STRE que obtém os valores particulares para cada instância: parâmetros reais com que essas instâncias serão criadas; prioridade inicial; e tamanho das áreas de memória que cada instância deseja usar para pilha e "heap". De maneira diferente do módulo pai, cada instância não necessita que o STRE solicite ao MS-DOS a carga do código objeto, pois este é compartilhado pelos processos gerados a partir do módulo pai e das suas instâncias. As áreas de segmento de código e de segmento de dados de uma instância são completamente separadas; e o STRE fornece a cada instância uma cópia privativa do segmento de dados, contendo sua própria cópia de registradores e dados guardados para serem utilizados no momento de sua execução.

O contexto inicial de um módulo consiste do registrador base da sua área de dados estáticos (DS), do tamanho de sua área de dados, do apontador para a estrutura texto correspondente ao código do módulo, do tamanho do código do módulo, do seu ponto de entrada (CS:IP) e do apontador do topo da sua pilha (SS:SP). A criação do contexto inicial de um módulo é feita através da interação do STRE com o módulo, durante o processo de carga deste. A interação entre o STRE e o módulo é necessária, pois a criação de um módulo é independente da criação dos demais módulos e da iniciação do STRE, não havendo, em conseqüência, uma fase de ligação global de todos os componentes da aplicação, onde esse contexto seja previamente determinado.

O processo de carga descrito é repetido para todos os módulos da aplicação; sendo que, após isso, a aplicação estará totalmente carregada na memória e pronta para ser executada. O comportamento do sistema dependerá exclusivamente do programa de aplicação do usuário. O STRE intervirá apenas quando chamado pelos módulos do usuário ou pela ocorrência de alguma interrupção.

A figura 5.5 apresenta o mapa de memória de uma estação quando uma aplicação, configurada pela LCM, estiver em execução sob o suporte do NOD. Os primeiros 1024 bytes são reservados para o vetor de interrupções, seguido do MS-DOS, do código do NOD e dos módulos da aplicação.

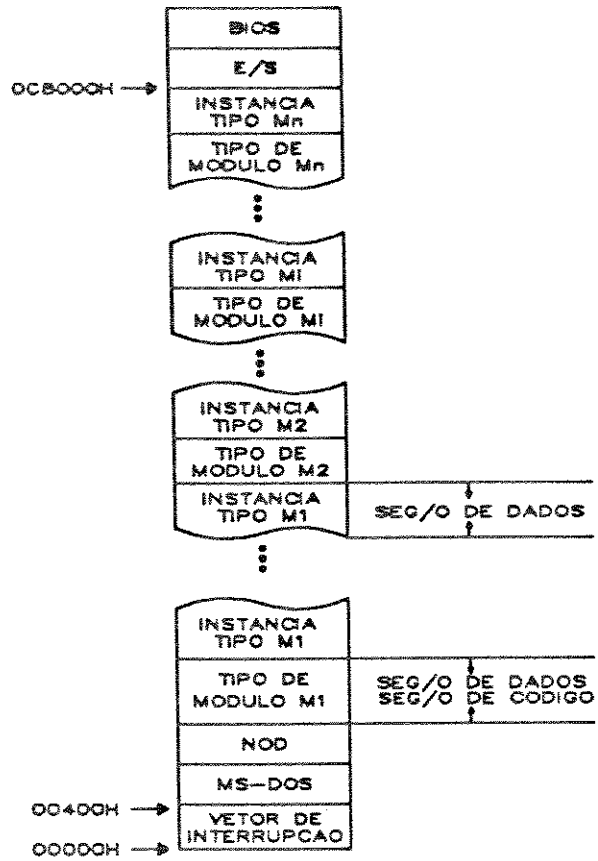

Figura 5.5 : Mapa de Memória de uma Estação em Execução

5.2.3 Suporte ao Subsistema de Comunicação Remota entre Módulos (SCR)

Embora logicamente transparente para a aplicação, as comunicações locais e remotas têm diferenças substanciais, do ponto de vista da implementação no NOD.

Na realidade, existem dois tipos de interfaces entre a Camada da Aplicação e o NOD.

O primeiro tipo corresponde a uma *interface explícita* entre a aplicação e o STRE. Essa interface, estática, implementa funções do STRE que não acionam o SCR, ou seja, não implicam em uma comunicação remota.

O segundo tipo de interface corresponde a uma *interface implícita*, considerada dinâmica, pois uma aplicação ao realizar uma troca de mensagens remota provoca a ativação, pelo STRE, de módulos pertencentes ao SCR.

Na figura 5.1, a interface implícita é indicada pela linha tracejada.

Existem duas possibilidades de cooperação entre módulos da aplicação que ocasionam a passagem por uma ou outra interface:

- Os módulos residem na mesma estação - comunicação local.
- Os módulos residem em estações diferentes - comunicação remota.

Deve ser observado que, no caso de uma comunicação local, o tipo de interface utilizada é a interface

explícita. O núcleo oferece o suporte local para a troca de mensagens entre os módulos, consistindo, basicamente, na cópia da mensagem da área origem para o "buffer" de entrada destino.

A figura 5.6 ilustra esse caso.

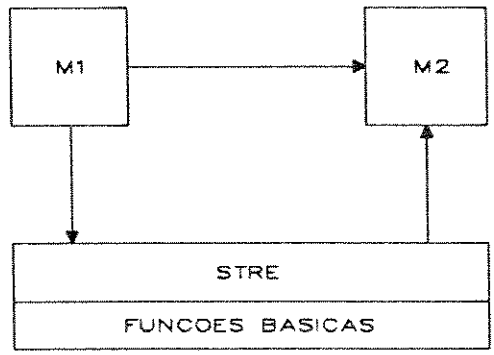

Figura 5.6 : Interface Explícita para uma Comunicação Local

No caso da comunicação remota o que ocorre é descrito a seguir.

Quando um módulo da aplicação da *ESTAÇÃO A* envia a mensagem e esta, em função do programa de configuração é destinada à *ESTAÇÃO B* remota, o suporte à troca de mensagens fornecido pelo STRE percebe que esta é para ser enviada através das camadas inferiores para o meio de comunicação. O STRE identificando essa mensagem como uma mensagem remota, envia-a para a *Camada SCR*, ativando-a através de um envio local normal. A comunicação local entre o STRE e o SCR, mais especificamente, o módulo Emissor do SCR, é executada através de uma porta de comunicação local síncrona. Dessa forma, o SCR garante que todas as mensagens (síncronas ou assíncronas) dos módulos fontes serão colocadas na fila da porta de entrada do módulo Emissor e copiadas para o seu "buffer" de comunicação, quando houver disponibilidade deste para transmitir nova mensagem.

No caso da mensagem da aplicação ser uma mensagem assíncrona, o STRE, interceptando-a, converte-a em uma mensagem síncrona com resposta nula ("signal"), ficando o módulo fonte da aplicação bloqueado à espera por esta resposta, emitida pelo módulo Emissor do SCR. Isso garante que nenhuma mensagem mais velha seja sobrescrita por outra mais nova de um módulo fonte diferente. O processo fonte somente continua a sua execução quando a sua mensagem tiver sido copiada para o "buffer" de comunicação do módulo Emissor do SCR.

No caso da mensagem da aplicação ser uma mensagem síncrona, o módulo fonte é bloqueado normalmente esperando por uma resposta.

A partir desse momento, o SCR se responsabiliza pelo envio da mensagem, através da utilização dos serviços oferecidos pela Camada do Serviço de Comunicação da Rede Local (CSCRL), implementados na rede local utilizada.

Esse envio é feito, após a execução de algumas funções necessárias para o sucesso dessa comunicação e após a formação do cabeçalho, que é colocado junto com a mensagem, específico para esta camada, para formar o protocolo equivalente da estação B.

A CSCRL, por sua vez, acrescenta um cabeçalho e uma terminação ao formato padrão transmitido pela Camada Superior, o qual contém informações para o protocolo da Camada equivalente da ESTAÇÃO B; e envia o quadro formado para a Camada Física.

A Camada Física não acrescenta cabeçalhos e transmite a mensagem para a estação B remota.

O quadro da mensagem chega ao CSCRL da *ESTAÇÃO B*, a qual retira o cabeçalho e a terminação do quadro recebido, que contém informações para ela. O restante é passado para a *Camada do SCR* que retira o seu cabeçalho e encaminha a mensagem para a aplicação.

A figura 5.7 ilustra a interface implícita.

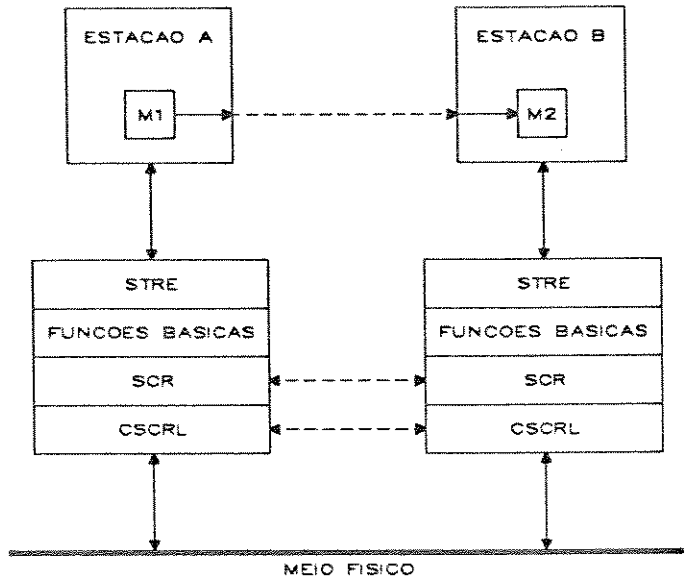

Figura 5.7 : Interface Implícita para uma Comunicação Remota

Deve ser mencionado, ainda, que todas as operações envolvidas na utilização da interface implícita são suportadas pelo STRE, através de funções reservadas, ou seja , não são acessíveis aos módulos da aplicação, já que essas funções têm acesso às estruturas de dados pertencentes ao STRE. Essas funções são utilizadas somente pelo SCR.

5.3 Subsistema de Comunicação Remota entre Módulos (SCR)

O SCR implementa um serviço do tipo DATAGRAMA, para a troca de mensagens entre módulos, através de uma rede local de comunicação.

Conforme citado anteriormente, quando um módulo da aplicação envia uma mensagem para um módulo remoto, o núcleo trata-a como se fosse um envio de mensagem normal, para uma porta de entrada local pertencente a um dos módulos do SCR. A mensagem é então acrescida de outras informações, para formar o padrão de mensagens utilizadas pelo SCR e transferida para a estação remota via o meio de comunicação. O SCR da estação remota direciona então a mensagem recebida para a porta de entrada do módulo destino da mensagem.

Tudo se passa como se cada camada SCR se comunicasse diretamente com a camada SCR da outra estação da rede, dentro de certas regras e convenções conhecidas como protocolos de comunicação da camada SCR. Esses protocolos são responsáveis pela interação de módulos de camadas SCR, localizadas no mesmo nível de hierarquia em estações diferentes.

A Camada SCR possui as seguintes funções:

- Fornecer serviços à Camada Superior (Camada da Aplicação) através da Interface Implícita;
- Utilizar os serviços oferecidos pela CSCRL através da interface fornecida pelos módulos de interface específicos para a rede utilizada.

O SCR proporciona esses serviços para as suas camadas adjacentes, através de interfaces bem definidas. Uma interface é necessária entre o SCR e a sua camada superior; e uma outra interface é necessária entre o SCR e a sua camada inferior.

A figura 5.8 mostra o modelo de camadas local de cada estação com a Camada de Aplicação no nível superior, a Camada do SCR, no nível intermediário e a CSCRL, no nível inferior. A figura ilustra, também, a interface para as camadas adjacentes e o protocolo da camada SCR utilizado entre estações remotas, de uma maneira geral.

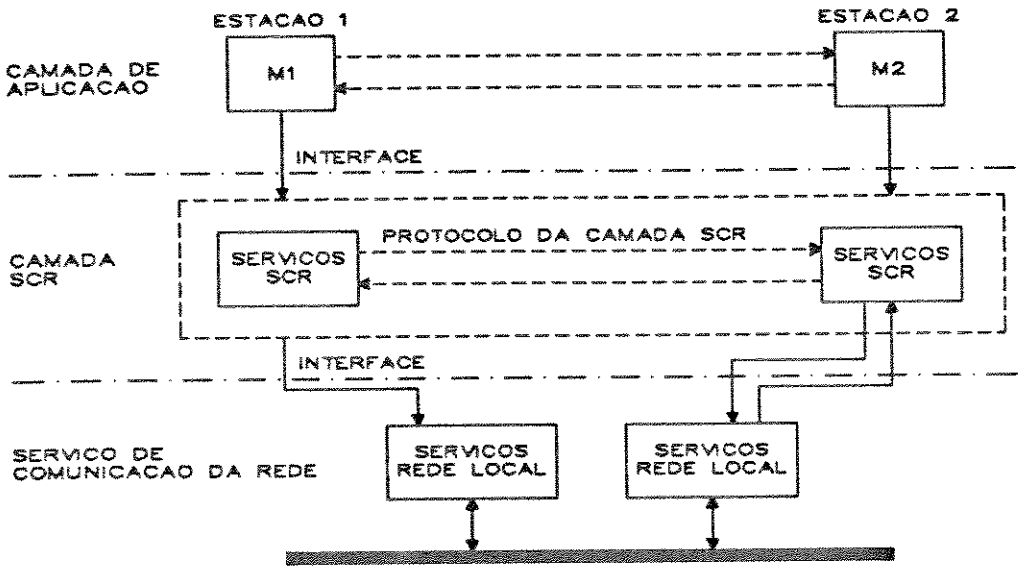

Figura 5.8 : Modelo em Camadas com Interface e Protocolo da Camada SCR

O SCR consiste de um conjunto de módulos responsáveis pela troca de mensagens remotas, de forma transparente à aplicação, entre módulos comunicantes localizados em estações diferentes. Os módulos do SCR são implementados usando a LPM e, portanto, utilizam-se das primitivas de comunicação normais disponíveis na linguagem. São módulos especiais, que executam com o nível de prioridade mais elevado, sendo autorizados a acessar estruturas de dados do núcleo através de chamadas a alguns serviços exclusivos.

A opção de implementação da camada SCR em módulos LPM, deve-se à facilidade de utilização das características de suporte à programação concorrente do ambiente STER. Aliada a essas características, o fato de utilizar uma ferramenta do próprio ambiente, torna a arquitetura de software melhor estruturada e mais homogênea, facilitando, dessa forma, a transportabilidade do STER para outras arquiteturas de hardware.

5.3.1 Descrição do Protocolo de Comunicação do SCR

O SCR implementa dois protocolos da camada SCR, para suportar os dois tipos de primitivas de comunicação entre módulos do STER. Estes são os protocolos *SÍNCRONO* e *ASSÍNCRONO*, cujo formato é próprio para o SCR.

Para suporte às primitivas de comunicação síncronas e assíncronas, entre módulos da aplicação, os protocolos do SCR utilizam o formato padrão de mensagens, denominado *UPSC* (Unidade do Protocolo do SCR), definido pela declaração PASCAL abaixo:

```
CONST
      maxtamsg = 128;
TYPE
      tipomsg     =     (urequest, ureply, unotify);
      endsys      =     RECORD
                              s, m, p: INTEGER;
                        END;
      bufdado     =     ARRAY [1.. maxtamsg] OF BYTE;
      aptUPSC     =     ADS OF tUPSC;
      tUPSC       =     RECORD
                              mensagem  : tipomsg;
                              transação : INTEGER;
                              valtempo  : INTEGER;
                              destino   : endsys;
                              fonte     : endsys;
                              tamdado   : INTEGER;
                              dadousuario : bufdado;
                        END;
```

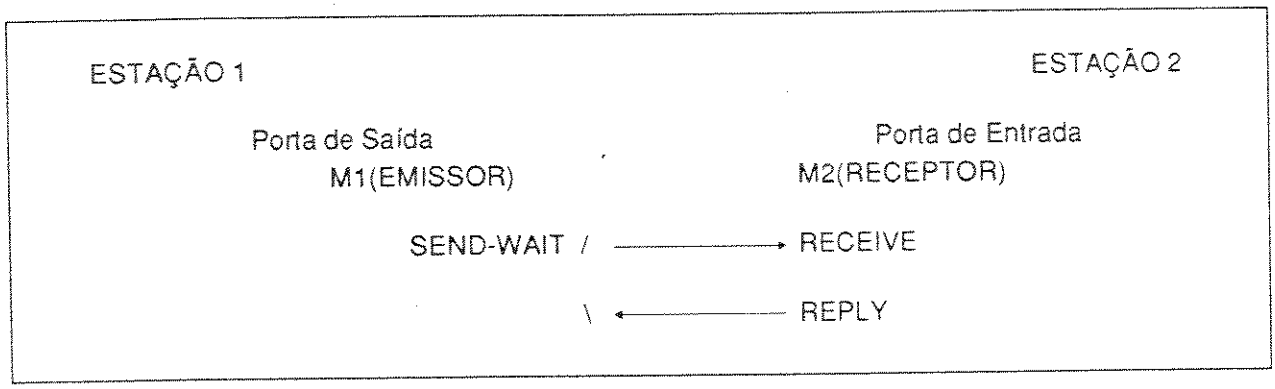

Figura 5.9 : Protocolo Síncrono

O *protocolo síncrono* é constituído de duas mensagens que estabelecem uma troca bidirecional de mensagens entre estações distintas, consistindo no envio de uma mensagem em um sentido e de uma resposta no outro (figura 5.9).

O UPSC gerado para as mensagens do tipo "urequest" e "ureply" contém as informações descritas nas tabelas abaixo.

PROTOCOLO : SÍNCRONO		
TIPO DE MENSAGEM	CAMPO	DESCRIÇÃO
REQUISIÇÃO SÍNCRONA	mensagem	contém o tipo de mensagem enviada : 'urequest'
	transação	identificador de mensagens gerado pelo núcleo
	valtempo	valor de "timeout" definido pelo módulo que executa uma primitiva 'SEND-WAIT'. Se não definido este é iniciado com um valor padrão
	destino	endereço de sistema da porta de entrada destino
	fonte	endereço de sistema da porta de saída fonte
	tamdado	tamanho da mensagem enviada pela aplicação
	dadousuario	mensagem da aplicação

Tabela 5.1 : Protocolo Síncrono (MSG = Requisição)

A mensagem de resposta gerada pelo SCR na estação remota, contém as informações descritas na tabela abaixo.

PROTOCOLO : SÍNCRONO		
TIPO DE MENSAGEM	CAMPO	DESCRIÇÃO
RESPOSTA DA REQUISIÇÃO SÍNCRONA	mensagem	contém o tipo de mensagem enviada : 'ureply'
	transação	identificador de mensagens identico a mensagem do tipo urequest associada
	valtempo	não utilizado
	destino	endereço de sistema da porta de saída que iniciou a comunicação síncrona (correspondente ao campo fonte da mensagem do tipo urequest associada)
	fonte	endereço de sistema da porta de entrada que está respondendo
	tamdado	tamanho da mensagem de resposta da aplicação
	dadousuario	mensagem de resposta da aplicação

Tabela 5.2 : Protocolo Síncrono (MSG = Resposta)

O *protocolo assíncrono* é constituido de uma mensagem que estabelece um envio unidirecional de mensagens entre estações distintas, consistindo do envio de uma mensagem para uma outra estação pertencente à rede (figura 5.10).

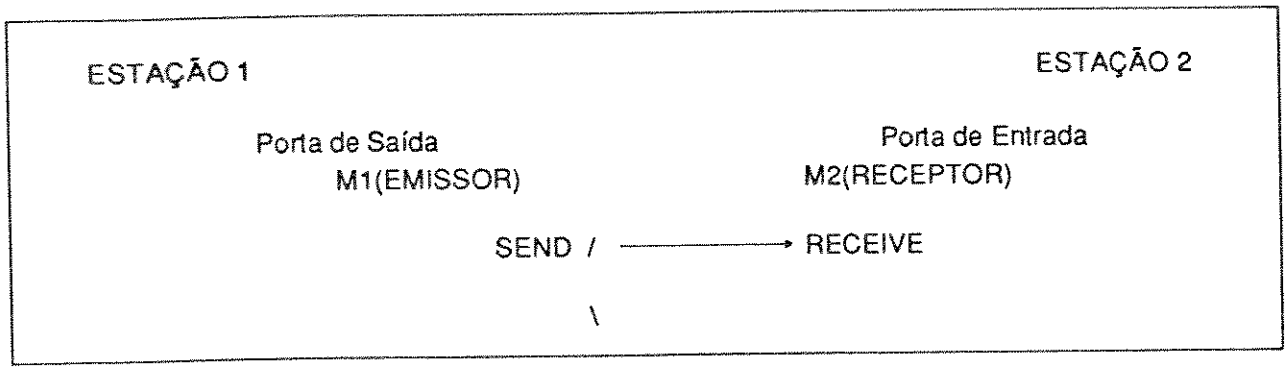

Figura 5.10 : Protocolo Assíncrono

O UPSC gerado para a mensagem do tipo "unotify" contém as informações descritas na tabela abaixo.

PROTOCOLO : ASSÍNCRONO		
TIPO DE MENSAGEM	CAMPO	DESCRIÇÃO
REQUISIÇÃO ASSÍNCRONA	mensagem	contém o tipo de mensagem enviada : "unotify"
	transação	campo não utilizado
	valtempo	campo não utilizado
	destino	endereço de sistema da porta de entrada destino
	fonte	endereço de sistema da porta de saída fonte
	tamdado	tamanho da mensagem enviada pela aplicaçao
	dadousuario	mensagem assíncrona da aplicação

Tabela 5.3 : Protocolo Assíncrono

5.3.2 Descrição da Interface do SCR

Uma interface entre duas camadas traduz-se pelo conjunto de serviços que a camada inferior oferece à superior. O ambiente STER contribui para a definição de interfaces, uma vez que podem, dentro do ambiente, ser implementadas através das portas de comunicação entre módulos.

Os módulos pertencentes à camada SCR comunicam-se com os módulos pertencentes às camadas adjacentes através de uma interface.

A interface implícita com a camada da aplicação (correspondendo à camada superior) e a interface com os módulos de interfaceamento da CSCRL (correspondendo à camada inferior) foram implementadas usando-se portas de comunicação entre módulos, como ilustrado na figura 5.11.

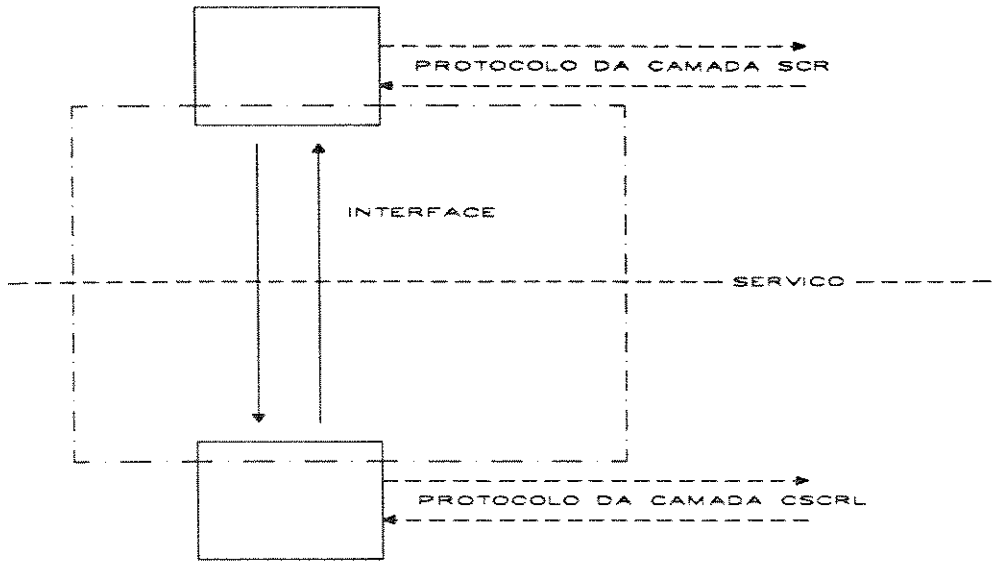

Figura 5.11 : Interface entre Camadas

5.3.3 Descrição dos Serviços Oferecidos pelo SCR

Os serviços oferecidos pelo SCR à sua Camada Superior consistem de funções encapsuladas em três módulos, escritos em LPM, responsáveis pelo gerenciamento (GERCOM), transmissão (EMISSOR) e recepção (RECEPTOR) de mensagens síncronas e assíncronas da aplicação.

O módulo GERENTE de COMUNICAÇÃO (GERCOM) é responsável pela iniciação da aplicação e do sistema local de comunicação, como também pelo gerenciamento de falhas ocorridas durante a execução da aplicação.

O módulo EMISSOR (ENVREM) acrescenta às mensagens recebidas em sua porta de entrada, informações de controle que permitem enviá-las à estação remota, usando o formato padrão de mensagens descrito adiante.

O módulo RECEPTOR (RECREM) trata as mensagens recebidas da CSCRL. Para isso, interpreta os campos de controle acrescentados pelo módulo ENVREM da estação emissora, procedendo as chamadas de funções do STRE necessárias ao envio da mensagem à porta de comunicação requisitada.

Os módulos citados são apresentados nos próximos itens de duas maneiras; uma descrição informal através do uso da linguagem natural para uma melhor compreensão de sua operação e uma especificação formal mais concisa e precisa. A especificação formal dos módulos é apresentada no Apêndice C.

Uma especificação formal pode ser feita através do uso de várias técnicas. Essas técnicas são classificadas em três categorias: modelos de transição, linguagens de programação e modelos mistos contendo as duas categorias anteriores.

Para a especificação formal, foi escolhida a técnica de modelos de transição. Nesta categoria classificam-se os modelos de Máquina de Estados Finita (MEF) [19], Redes de Petri [48], Linguagens Formais do tipo ESTELLE [64], entre outros.

Neste trabalho, utilizou-se a técnica da MEF para ilustração dos modelos de transição.

5.3.3.1 O Módulo EMISSOR (ENVREM)

O módulo ENVREM é um módulo privilegiado, (i.e., tem acesso a estruturas de dados do STRE), escrito em LPM, contendo três portas de comunicação, conforme ilustra a figura 5.12.

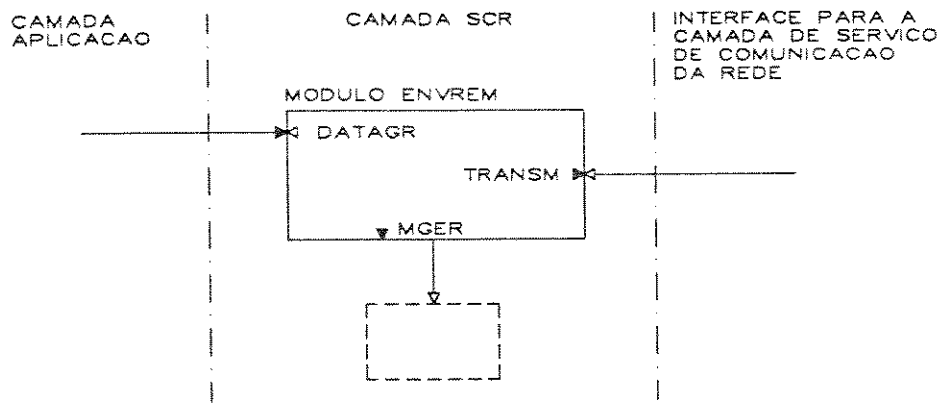

Figura 5.12 : Portas de Comunicação da Interface do Módulo ENVREM

A sua interface, em LPM, é a seguinte:

```
MODULE ENVREM;

    USE undefrs2.inc;

    ENTRYPORT
        DATAGR : bufdado REPLY tiposignal;

    EXITPORT
        TRANSM : aptUPSC REPLY tiposignal;
        MGER : tiposignal;

END_MODULE.
```

A sua porta de entrada "DATAGR" é uma porta síncrona, utilizada por ENVREM para esperar por mensagens da aplicação, encaminhadas através do STRE.

A sua porta de saída "TRANSM" é uma porta síncrona, utilizada por ENVREM no envio de um apontador do quadro formado, para o Módulo Transmissor de Mensagens da camada CSCRL.

A sua porta de saída "MGER" é uma porta assíncrona, utilizada por ENVREM para enviar um aviso para o módulo GERCOM.

ESPECIFICAÇÃO INFORMAL:

Quando um módulo da aplicação envia uma mensagem (síncrona ou assíncrona) para uma porta de saída, conectada a uma porta de entrada remota, o STRE procede como se tivesse executado uma primitiva de envio síncrono à porta de entrada "DATAGR" do módulo ENVREM local. O módulo fonte da mensagem fica bloqueado esperando por uma resposta. O módulo ENVREM recebe a mensagem a ser transmitida em um "buffer" interno de comunicação.

Sendo ENVREM um módulo privilegiado, ele obtém as informações necessárias para preenchimento dos campos do cabeçalho da mensagem UPSC a ser enviada ao módulo RECREM remoto, mediante o acesso às estruturas de dados internas do STRE . Para tanto, ENVREM utiliza-se dos serviços de acesso exclusivo, através da função especial do STRE, denominada "OBTSTNC". Essas informações são obtidas, por essa rotina, através do acesso às estruturas de dados do DPE de "DATAGR" e do Bloco de Contexto do Módulo Fonte (BCM). As informações citadas referem-se ao endereço de sistema da porta fonte da mensagem, endereço de sistema da porta destino da mensagem, tipo de mensagem (síncrona ou assíncrona), tamanho da mensagem e identificador de transação da mensagem.

Se a mensagem for assíncrona, ENVREM envia imediatamente uma mensagem de resposta, liberando o módulo emissor, uma vez que a mensagem já foi copiada no seu "buffer" de comunicação. Como a mensagem é assíncrona, o campo correspondente à mensagem no cabeçalho do UPSC, é preenchido com o tipo de mensagem "unotify".

Se a mensagem for síncrona, o valor do campo valtempo é definido por ENVREM, para uso na estação receptora, como sendo o valor de "timeout" especificado na primitiva de comunicação síncrona original, requisitada pela aplicação. Este é calculado, compensando o tempo que a mensagem espera até ser recebida pelo módulo de recepção (RECREM) da mensagem da camada. Como a mensagem é síncrona, o campo correspondente à mensagem no cabeçalho do UPSC é preenchido com o tipo de mensagem "urequest".

Para evitar operações de cópia, a mensagem no formato padrão de mensagens UPSC é mantida no módulo ENVREM.

Um apontador para a UPSC formada é enviado para a CSCRL através da porta de saída "TRANSM". Após a CSCRL enviar a mensagem (com sucesso ou fracasso) através do meio de transmissão, ela responderá indicando sucesso ou não e, liberará o módulo ENVREM para processar outra mensagem da aplicação, já que este não efetua retransmissões em caso de erro. Em caso de fracasso, ENVREM envia uma notificação de falha ao módulo GERCOM, através de sua porta "MGER".

A interação entre o SCR e a CSCRL é feita por módulos de interfaceamento descritos no item 5.3.4.

Na implementação atual é usada uma única instância do módulo ENVREM. Cada instância de ENVREM processa uma mensagem de cada vez, esperando até a sua transmissão pelo Serviço de Comunicação da Rede. Versões mais elaboradas, com mais de uma instância de ENVREM poderão ser objeto de trabalhos futuros.

5.3.3.2 O Módulo Receptor (RECREM)

O Módulo RECREM é um módulo priviligeado, (i.e.,tem acesso a estruturas de dados do STRE), escrito em LPM, contendo cinco portas de comunicação, conforme ilustra a figura 5.13.

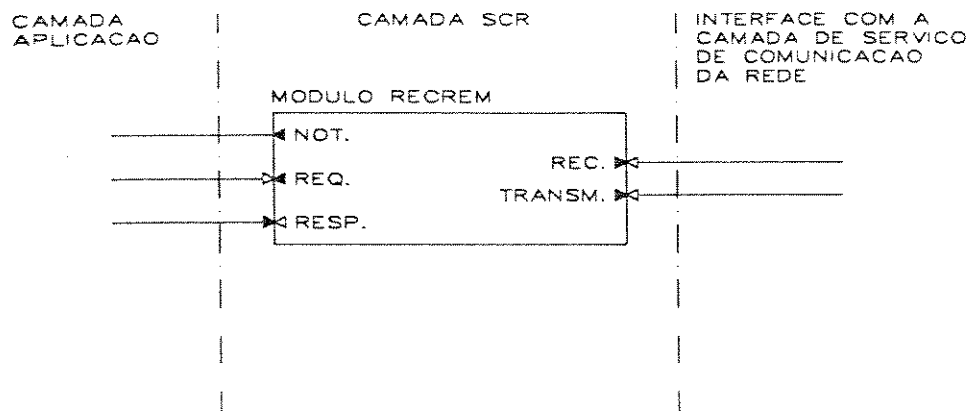

Figura 5.13 : Portas de Comunicação da Interface do módulo RECREM

A sua interface, descrita em LPM, é a seguinte:

```
MODULE RECREM;

    USE undefrs2.inc;

    ENTRYPORT
        RESP : tiposignal REPLY bufdado;

    EXITPORT
        REC      :    aptUPSC REPLY tiposignal;
        TRANSM   :    aptUPSC REPLY tiposignal;
        REQ.     :    bufdado REPLY bufdado;
        NOT.     :    bufdado;
END_MODULE.
```

A sua porta de saída "REC" é uma porta síncrona, utilizada por RECREM para enviar um apontador para seu "buffer" de requisição de mensagens.

A sua porta de saída assíncrona "NOT." é utilizada para fazer uma conexão dinâmica com a porta de entrada do módulo destino da aplicação, no caso de uma comunicação assíncrona.

A sua porta de saída "REQ." é uma porta síncrona, utilizada para fazer uma conexão dinâmica com a porta de entrada do módulo destino da aplicação, no caso de uma comunicação síncrona.

A sua porta de saída "TRANSM" é uma porta síncrona, utilizada para transmitir a resposta da aplicação para o módulo fonte remoto.

A sua porta de entrada " RESP" é utilizada para enviar uma mensagem do tipo "REPLY", para o módulo destino da aplicação.

ESPECIFICAÇÃO INFORMAL:

O Módulo RECREM recebe e trata todas as mensagens recebidas da CSCRL. Podem ser definidas múltiplas instâncias de RECREM, utilizando-se a construção **NETWORK**. Na estação remota, o módulo destino local pode não estar pronto para receber a mensagem que chegou pela CSCRL. Ocorrendo esse caso, o módulo do serviço de comunicação da camada CSCRL seria suspenso, esperando a possibilidade de transferir a mensagem. Para que isso não ocorra, um grupo de módulos (multiplas instâncias do módulo RECREM) são proporcionadas para atuar como substitutos e guardar as mensagens que chegam. O número das instâncias do tipo de módulo RECREM no SCR, também chamadas de "buffers" ativos de mensagens, corresponde à quantidade de mensagens oriundas do meio de comunicação que uma estação pode tratar simultaneamente. Esse número depende da aplicação considerada. O valor padrão considerado pela LCM corresponde a quatro "buffers" ou instâncias de RECREM.

Cada instância livre de um módulo RECREM oferece seus serviços para o módulo de recepção da camada CSCRL enviando, através da sua porta de saída "REC", um apontador para o seu "buffer" de requisição, do tipo **UPSC**, ao módulo de interface com a CSCRL. As instâncias do tipo RECREM livres ficam na fila da porta de entrada do módulo de recepção da camada CSCRL. Quando o módulo de interface com a CSCRL receber uma mensagem com sucesso no "buffer" de requisição de RECREM, ele deverá enviar uma mensagem de sincronismo ("signal"), em resposta, liberando o módulo RECREM correspondente. O módulo de recepção da camada CSCRL envia a mensagem de sincronismo para o "buffer" da primeira instância de RECREM na sua fila. Ao receber o "signal", a instância de RECREM sabe que seu "buffer" contém uma mensagem recém chegada do meio de comunicação e, dependendo do tipo de mensagem, indicada no formato padrão da mensagem chegada, ela toma as atitudes descritas a seguir:

a) Se a mensagem recebida indicar um envio assíncrono (mensagem = "unotify").

O módulo RECREM utiliza o endereço destino, obtido da **UPSC** recebida, para ligar, dinamicamente, sua porta de saída "NOT.", com a porta de entrada do módulo destino da aplicação. Isso é realizado através de serviços privilegiados ao SCR, que acessam funções do STRE tais como: "LUNLIK" e "LLINK". A função "LUNLIK", inicialmente, faz a desconexão da porta "NOT." para ter certeza de que não existe nenhuma comunicação em progresso. A seguir, a função "LLINK" é utilizada para fazer a conexão dinâmica solicitada.

Com a conexão dinâmica estabelecida, o módulo RECREM efetua então uma operação de comunicação normal do STRE, enviando uma mensagem assíncrona ("SEND") à sua porta "NOT.", através da qual a mensagem da aplicação chega ao módulo da aplicação destino.

Como a mensagem é assíncrona, o SEND é executado e o módulo RECREM continua a sua execução, retornando a oferecer seus serviços para a CSCRL.

b) Se a mensagem recebida indicar um envio síncrono (mensagem = "urequest").

O módulo RECREM utiliza o endereço destino, obtido da **UPSC** recebida, para ligar dinamicamente sua porta de saída "REQ." para a porta de entrada do módulo destino da aplicação. Isso é realizado com as funções "LUNLIK" e "LLINK" do STRE. Com a conexão dinâmica estabelecida, o módulo RECREM requisita uma primitiva de comunicação síncrona ("SEND_WAIT"), com o valor de "timeout" igual ao conteúdo do campo valtempo da **UPSC** recebida da CSCRL. Os atrasos de transmissão são pequenos em comparação com valores de "timeout", conseqüentemente o valor de *valtempo* não é ajustado para compensar eventuais atrasos de transmissão. Este "timeout" é utilizado para recuperação de recursos, no caso, recuperação dos "buffers" ativos de comunicação, se a transação falhar.

A mensagem remota, recebida, chega ao módulo destino da aplicação, se não ocorrer nenhuma falha na transmissão.

Como a mensagem enviada à aplicação pelo módulo RECREM foi uma mensagem síncrona, a mensagem de resposta enviada pela aplicação será copiada no "buffer" de requisição de RECREM.

O módulo RECREM, então, troca os endereços de fonte e destino na UPSC.

Usando os serviços de acesso exclusivo, o módulo RECREM preenche os campos de informações da UPSC a ser enviada remotamente. Utiliza a rotina especial do STRE, denominada "OBTAMRL", que obtém a estrutura de dados do NOD, o tamanho real da mensagem enviada pela aplicação e a coloca no campo TAMDADO do quadro. Coloca no campo mensagem, o tipo da mensagem enviada que é "ureply" e copia a mensagem do buffer de requisição para o campo *dadousuario*.

Com a UPSC completa, envia um apontador para esta UPSC à CSCRL, mais especificamente com o módulo de interface de transmissão, através de sua porta síncrona de saída "TRANSM".

O módulo RECREM retorna para o estado de espera por uma nova mensagem, quando a CSCRL responder com uma mensagem ("signal"), indicando que a mensagem de resposta da aplicação, foi transmitida pelo meio de comunicação.

c) Se a mensagem recebida indicar uma resposta (mensagem = "ureply").

Através de serviços de acesso exclusivo, o módulo RECREM solicita a conexão de sua porta de entrada síncrona "RESP.", através da função do STRE denominada "ORGPRTE", com a porta de saída do módulo da aplicação que iniciou a comunicação, de tal modo que tudo se passa como se a sua porta de entrada fosse realmente a porta para a qual a mensagem de requisição fonte foi enviada. Após efetuar a conexão, o módulo RECREM entrega ao módulo da aplicação destinatário a mensagem recebida, executando uma primitiva de resposta ("REPLY"), endereçada à sua porta "RESP.".

5.3.3.3 O Módulo Gerente de Comunicação (GERCOM)

O Módulo **GERCOM** é um módulo priviligeado, (i.e., tem acesso a estruturas de dados do STRE), escrito em LPM, contendo quatro portas de comunicação, conforme ilustra a figura 5.14.

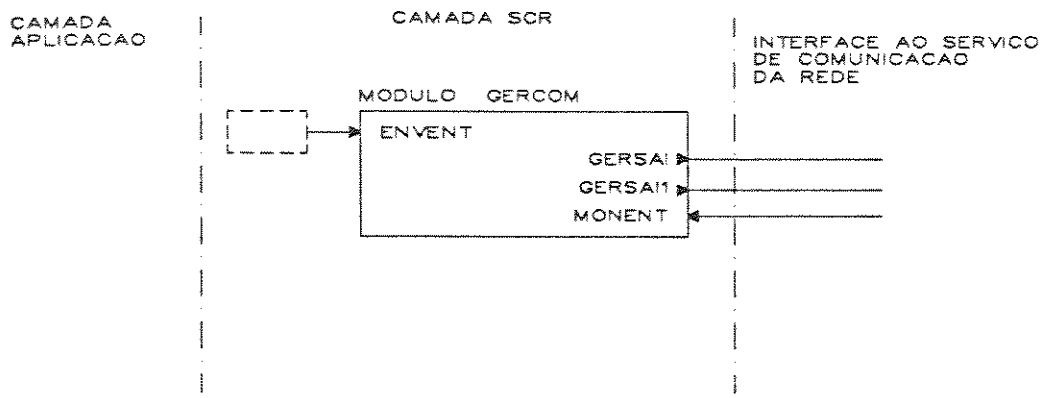

Figura 5.14 : Portas de Comunicação da Interface do módulo GERCOM

A sua interface, descrita em LPM, é a seguinte:

```
MODULE GERCOM;

    USE undefrs2.inc;

    ENTRYPORT
        MONENT : tiposignal;
        ENVENT : tiposignal;

    EXITPORT
        GERSAI  : BYTE;
        GERSAI1 : INTEGER;

END_MODULE.
```

A sua porta de saída "GERSAI" é uma porta assíncrona, utilizada por GERCOM para enviar o endereço local da estação para a CSCRL utilizada.

A sua porta de saída "GERSAI1" é uma porta assíncrona, utilizada por GERCOM para enviar a requisição de consultas às estações remotas, conectadas à estação.

A sua porta de entrada "MONENT" é uma porta assíncrona, utilizada por GERCOM para esperar por uma confirmação da requisição de consulta às estações.

A sua porta de entrada "ENVENT" é uma porta assíncrona, utilizada por GERCOM para esperar por notificações de falhas ocorridas durante a execução.

ESPECIFICAÇÃO INFORMAL :

O módulo GERCOM é responsável pela realização de três funções principais:

- Iniciação do sistema local de comunicação;
- Ativação inicial da aplicação;
- Tratamento de falhas.

A função de iniciação do sistema local de comunicação corresponde, basicamente, à distribuição de parâmetros associados ao sistema local de comunicação como, por exemplo, o endereço local da estação e consultas a estações remotas.

A função de ativação inicial da aplicação visa liberar a aplicação local para execução, após a confirmação positiva das consultas feitas à interface da CSCRL, para certificar-se da disponibilidade das estações necessárias à execução da aplicação distribuída.

A função de tratamento de falhas é requisitada durante a execução da aplicação, quando o módulo GERCOM recebe notificações de falhas de comunicação enviadas pelo módulo Emissor do SCR, executando as ações necessárias.

5.3.4 Interface com o Serviço de Comunicação da Rede

O SCR é independente do meio de comunicação escolhido, necessitando somente de uma interface específica para o software de comunicação suportado pela rede.

A interface consta de dois módulos: Interface de Transmissão e Interface de Recepção (INTRAN e INTREC), escritos em LPM, responsáveis pela troca de informações entre o SCR e a CSCRL utilizada.

A função principal do módulo INTRAN é receber um apontador à mensagem que o SCR deseja enviar, transmitindo-a em seguida, usando as primitivas da CSCRL utilizada.

A função principal do módulo INTREC é receber, em um endereço indicado pelo módulo RECREM do SCR, um UPSC remoto endereçado a estação.

A figura 5.15 ilustra o SCR e sua interface residentes em cada estação da rede.

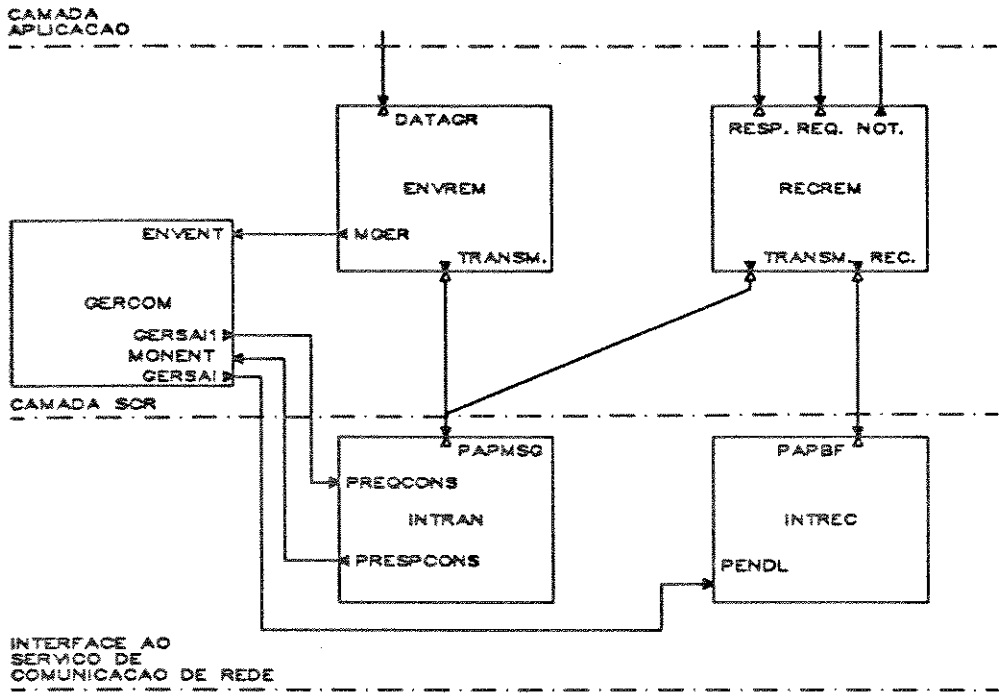

Figura 5.15 : SCR e sua Interface

5.3.5 Configuração do SCR

A figura 5.16 apresenta a LCM a nível STATION, onde são descritas as conexões entre as portas das instâncias dos módulos de comunicação do SCR com as portas dos módulos de interface. A configuração é gerada, automaticamente, pela LCM a partir da especificação da diretiva de rede no nível NETWORK, a ser utilizada pela aplicação.

```
STATION EST;

        INSTANCE
                envr : ENVREM;
                gcom : GERCOM;
                intr : INTREC;
                intt  : INTRAN;
                rec1,rec2,rec3,rec4 : RECREM;

        CREATE
                envr /P=SYSTEMLOWPR,
                gcom /P=SYSTEMLOWPR,
                intr /P=SYSTEMHIGHPR /S=64 /H=2,        intt /P=SYSTEMLOWPR,
                rec1 /P=SYSTEMLOWPR,    rec2 /P=SYSTEMLOWPR,
                rec3 /P=SYSTEMLOWPR,    rec4 /P=SYSTEMLOWPR,

        LINK
                envr.transm         TO   intt.PApMsg;
                envr.mgersai        TO   gcom.envent;
                gcom.gersai         TO   intr.PEndL;
                gcom.gersai1        TO   intt.PReqCons;
                intt.PRespCons      TO   gcom.monent;
                rec1.rec            TO   intr.PApBf;
                rec1.transm         TO   intt.PApMsg;
                rec2.rec            TO   intr.PApBf;
                rec2.transm         TO   intt.PApMsg;
                rec3.rec            TO   intr.PApBf;
                rec3.transm         TO   intt.PApMsg;
                rec4.rec            TO   intr.PApBf;
                rec4.transm         TO   intt.PApMsg;

    END_STATION.
```

Figura 5.16 : Configuração do SCR

5.4 Arquitetura de Implantação e Carga do SDS

A presente versão do **SDS** foi realizada utilizando, para cada estação, um microcomputador compatível com o IBM-PC. A razão da escolha deveu-se à grande popularidade dos compatíveis com o PC de uma maneira geral, bem como à sua penetração nos ambientes industriais. Com a implementação do SDS nesse equipamento, fornece-se uma opção para o desenvolvimento de aplicativos distribuídos de tempo real, voltados ao controle e supervisão de processos.

A carga do SDS, nesta versão, não é feita através da rede local utilizada mas, sim, manualmente; sendo realizada pelo MS-DOS onde o SDS é um programa como outro qualquer. Sendo assim, para ser carregado, é digitado o seu nome no teclado. O MS-DOS permite que se passe uma linha de comando ao

programa sendo carregado. No caso do SDS, a linha de comando esperada deve especificar o nome do arquivo que contém a tabela de configuração da aplicação, específica para cada estação física da rede de comunicação, que se pretende executar.

Para executarmos uma aplicação cuja configuração esteja descrita nas tabelas contidas nos arquivos ESTAÇÃO1.CNF e ESTAÇÃO2.CNF, deve-se digitar no teclado do PC de endereço físico número 1:

C> STRDST ESTAÇÃO1.CNF <CR>

Da mesma maneira, no teclado do PC de endereço físico número 2:

C> STRDST ESTAÇÃO2.CNF <CR>

Em resposta ao comando, de maneira idêntica em cada estação da rede, o MS-DOS carrega o NOD na primeira posição livre de memória, passa-lhe o controle e deixa disponível a linha de comando ESTAÇÃOn.CNF.

A execução do NOD começa na rotina responsável pelas iniciações que antecedem a carga da aplicação. Após esta, a rotina de iniciação solicita ao MS-DOS a carga da tabela de configuração especificada pelo usuário. O MS-DOS carregará a tabela na primeira posição livre de memória após o NOD. A tabela de configuração, embora só contenha dados, é vista pelo MS-DOS como um programa; assim, a liberação explícita de memória é necessária, pois é alocada toda a memória disponível ao programa recém carregado, não sendo possível a solicitação da carga de novos programas, a menos que a memória para esse fim seja liberada. O NOD continua com o processo de iniciação, usando as informações da tabela de configuração, o que é descrito no item iniciação do STRE. Após a utilização da tabela, o NOD libera a memória alocada para a tabela, a qual será alocada para os módulos da aplicação e suas instâncias. O programa de aplicação do usuário é carregado, conforme especificado na tabela de configuração, nas estações físicas, culminando com a transferência de controle do NOD para o módulo de maior prioridade.

A interligação física das estações é realizada através de uma rede local simplificada, que utiliza portas seriais para a comunicação. O software da CSCRL não está diretamente ligado ao objetivo desta dissertação, no entanto, é um trabalho que teve que ser realizado pela necessidade de ter-se uma rede de comunicação local para testar e viabilizar a implementação do SD. No Apêndice D existe uma documentação detalhada com a especificação dos módulos utilizados na rede, que pode servir de base para uma implementação de baixo custo para pessoas em universidades ou com carência de recursos, e que têm as seguintes características:

- Camada da CSCRL : consiste de dois módulos, escritos em LPM, que implementam um protocolo de acesso ao meio do tipo CSMA-CD/PR [49];
- Camada Física : consiste de um par trançado, ao qual as estações são ligadas pelas suas interfaces seriais RS232C.

A justificativa para registrar esse software da CSCRL como parte da dissertação da tese desenvolvida deve-se ao fato de que este teve um certo peso no trabalho de implementação, devido ao esforço e tempo gasto em realizá-lo.

Muito embora esta versão não apresente alto desempenho, o seu uso mostrou-se adequado para a realização de laboratórios de programação e de experimentos distribuídos. A versão atual pode ser vista como uma alternativa para aplicações distribuídas com baixo fluxo de informação entre estações para utilização em ambientes não hostis.

5.5 Exemplo da Dinâmica da Passagem de Informações entre Estações

Este exemplo (figura 5.17) é apresentado para ilustrar a dinâmica envolvida em operações de envio e recepção de mensagens realizadas por um programa de aplicação, entre duas estações remotas, bem como as camadas transparentes ao usuário, por onde essas mensagens fluem até chegar ao seu destino.

Os dois casos em estudo referem-se ao envio de mensagens síncronas e assíncronas entre estações remotas, passando pelas camadas da Aplicação, SCR, CSCRL e meio físico de comunicação.

Os módulos TXCCPE e RXCCPE da camada CSCRL, descritos nos pontos de localização a seguir, correspondem respectivamente aos módulos de interface INTRAN e INTREC, no caso específico da rede serial local simplificada, por nós desenvolvida. As portas dos módulos TXCCPE e RXCCPE que não estão ligadas, na figura 5.17, contém a mesma ligação que aparece na figura 5.15. As ligações não são ilustradas para não sobrecarregar a figura. O mesmo acontece com as portas dos módulos RECREM e ENVREM, ressaltando-se que têm a mesma função em cada estação lógica a que pertencem.

5.5.1 Primeiro Caso

Uma mensagem assíncrona remota é enviada pelo módulo M1, pertencente à estação A, através de sua porta X1 para a porta L1 do módulo M2, pertencente à estação B.

Em tempo de configuração (STATION), a porta X1 interna do módulo M1 da estação A foi associada, pelo usuário, à porta de interface IS1 da estação A. Do mesmo modo, a porta L1 do módulo M2 da estação B foi associada à porta de interface IE2 da estação B. As interconexões entre as estações A e B foram especificadas (NETWORK), ligando-se a porta de interface IS1 com a porta de interface IE2.

Em tempo de carga da aplicação, o STRE cria dinamicamente um BDCR, gerando uma ligação local síncrona da porta X1 com a porta "DATAGR" do módulo emissor de comunicação ENVREM do SCR.

O envio assíncrono ativa o serviço do núcleo relacionado ao envio de mensagens assíncronas. O STRE verificando que se trata de um envio assíncrono remoto, procede como se o módulo M1 tivesse executado uma primitiva de envio síncrono à porta de entrada "DATAGR" do módulo ENVREM do SCR. O módulo M1 fica em espera, apesar do envio assíncrono, bloqueado pelo artifício gerado pelo STRE até a chegada de uma resposta do módulo ENVREM.

Descrevem-se, a seguir, os pontos de localização, referentes à comunicação assíncrona, pertencentes à figura 5.17, para uma melhor compreensão.

PONTO A: O STRE procede como se o módulo da aplicação tivesse enviado uma mensagem síncrona local à porta "DATAGR" do módulo ENVREM do SCR. O módulo ENVREM recebe a mensagem através de um envio de mensagem local. Como o tipo de mensagem recebido refere-se a uma mensagem assíncrona, o módulo ENVREM envia imediatamente uma mensagem de resposta "nula" para a porta "DATAGR", liberando desta maneira o módulo M1 da aplicação. Os campos do cabeçalho da mensagem da aplicação (UPSC) são preenchidos e um apontador para a UPSC formada é enviado para a camada CSCRL através da porta "Transm.". O módulo ENVREM espera por uma confirmação positiva ou negativa da transmissão da mensagem.

PONTO F: O apontador à UPSC, formada na camada SCR, é recebido na porta "PApMsg" do módulo TXCCPE da camada CSCRL. Os campos do cabeçalho do quadro dessa

camada são preenchidos e o quadro formado é transmitido através do meio de comunicação. O módulo TXCCPE libera o módulo ENVREM do SCR com uma confirmação positiva ou negativa da mensagem enviada, permitindo que o módulo ENVREM envie outra mensagem.

PONTO H: Quando as instâncias do módulo RECREM estiverem desocupadas, elas enviam um apontador ao endereço disponível (buffer), onde desejam receber uma UPSC, ao módulo RXCCPE da camada CSCRL. O módulo RXCCPE recebe um quadro de mensagem do meio de comunicação, confirma se o quadro é endereçado a essa estação, e, em caso positivo, e não correspondendo a um quadro duplicado, envia um sinal de sucesso ao módulo RECREM através da sua porta "PApBf".

PONTO E: As mensagens assíncronas, de notificação ("unotify"), recebidas pelo módulo RECREM da camada SCR são enviadas através da conexão local normal entre a porta assíncrona "Not." do módulo RECREM e a porta assíncrona da aplicação. Essa conexão local é realizada dinamicamente pelo SCR, utilizando-se das primitivas especiais do STRE.

5.5.2 Segundo Caso

Uma mensagem síncrona remota é enviada pelo módulo M1, pertencente à estação A, através de sua porta X2 para a porta L2 do módulo M2, pertencente à estação B.

Em tempo de configuração (STATION), a porta X2 interna do módulo M1 da estação A foi associada, pelo usuário, à porta de interface IS2 da estação A. Do mesmo modo, a porta L2 do módulo M2 da estação B foi associada à porta de interface IE1 da estação B. As interconexões entre as estações A e B foram especificadas (NETWORK), ligando-se a porta de interface IS2 com a porta de interface IE1.

Em tempo de carga da aplicação, o STRE cria dinamicamente um BDCR, gerando uma ligação local síncrona da porta X2 com a porta "DATAGR" do módulo emissor de comunicação ENVREM do SCR.

O envio síncrono ativa o serviço do núcleo relacionado ao envio de mensagens síncronas. O STRE verificando que se trata de um envio síncrono remoto, prossegue executando um envio de mensagens síncrona à porta de entrada "DATAGR" do módulo ENVREM do SCR. O módulo M1 fica em espera, bloqueado até a chegada de uma resposta que vem através do módulo RECREM.

Descrevem-se, a seguir, os pontos de localização, referentes à comunicação síncrona, pertencentes à figura 5.17, para uma melhor compreensão.

PONTO B: O STRE envia uma mensagem síncrona local à porta "DATAGR" do módulo ENVREM do SCR. O módulo ENVREM recebe a mensagem através de um envio de mensagem local. Como o tipo de mensagem recebido refere-se a uma mensagem síncrona, o campo valtempo é definido para uso na estação receptora como o valor de "timeout" especificado na primitiva de comunicação original da aplicação. Os campos do cabeçalho (UPSC) da mensagem da aplicação são preenchidos e um apontador para a UPSC formada é enviado para a camada CSCRL através da porta "Transm.". O módulo ENVREM espera a confirmação ou não da transmissão da mensagem pelo CSCRL. O módulo ENVREM não responde à mensagem síncrona. A resposta da comunicação síncrona virá por intermédio do módulo RECREM do SCR.

PONTO F: O apontador à UPSC, formada na camada SCR, é recebido na porta "PApMsg" do módulo TXCCPE da camada CSCRL. Os campos do cabeçalho do quadro dessa camada são preenchidos e o quadro formado é transmitido através do meio de comunicação. O módulo TXCCPE libera o módulo ENVREM do SCR com uma confirmação positiva ou negativa da mensagem enviada, permitindo que o módulo ENVREM envie outra mensagem.

PONTO H: Quando as instâncias do módulo RECREM estiverem desocupadas, elas enviam um apontador ao endereço disponível (buffer), onde desejam receber uma UPSC, ao módulo RXCCPE da camada CSCRL. O módulo RXCCPE recebe um quadro de mensagem do meio de comunicação, confirma se o quadro é endereçado a essa estação, e, em caso positivo, e não correspondendo a um quadro duplicado, envia um sinal de sucesso ao módulo RECREM através da sua porta "PApBf".

PONTO D: As mensagens síncronas, de requisição ("urequest"), recebidas pelo módulo RECREM do SCR são enviadas através da conexão local normal entre a porta síncrona "Req" do módulo RECREM e a porta síncrona da aplicação. Essa conexão local é realizada dinamicamente pelo SCR, utilizando-se das primitivas especiais do STRE. Com a conexão dinâmica estabelecida, o módulo RECREM requisita uma primitiva de comunicação síncrona ("SEND-WAIT"), com o valor de "timeout" igual ao conteúdo do campo valtempo, obtido da UPSC recebida da camada CSCRL. A mensagem retirada da UPSC chega ao módulo destino da aplicação. A mensagem de resposta, da comunicação síncrona, é recebida pelo módulo RECREM. Usando os serviços de acesso às primitivas especiais do núcleo STRE, o módulo RECREM preenche os campos do cabeçalho de informações (UPSC), incluindo a mensagem de resposta e o seu tipo ("ureply"). Um apontador a essa UPSC é enviado para a camada CSCRL, através de sua porta "Transm.".

PONTO G: O módulo RECREM do SCR envia um apontador à UPSC formada, com a mensagem de resposta da aplicação à requisição síncrona enviada pela estação remota, para a porta "PApMsg" do módulo TXCCPE da camada CSCRL para ser transmitida pelo meio de comunicação. O quadro gerado nessa camada é transmitido através do meio de comunicação. O módulo TXCCPE libera o módulo RECREM do SCR com uma confirmação positiva ou negativa da mensagem enviada, permitindo que o módulo RECREM continue com sua execução.

PONTO C: As mensagens de resposta ("ureply") às requisições síncronas, recebidas pelo módulo RECREM do SCR, são enviadas através da conexão local normal entre a porta síncrona "Resp." do módulo RECREM e a porta síncrona da aplicação. Essa conexão é realizada dinamicamente pelo SCR, utilizando-se de primitivas especiais do STRE.

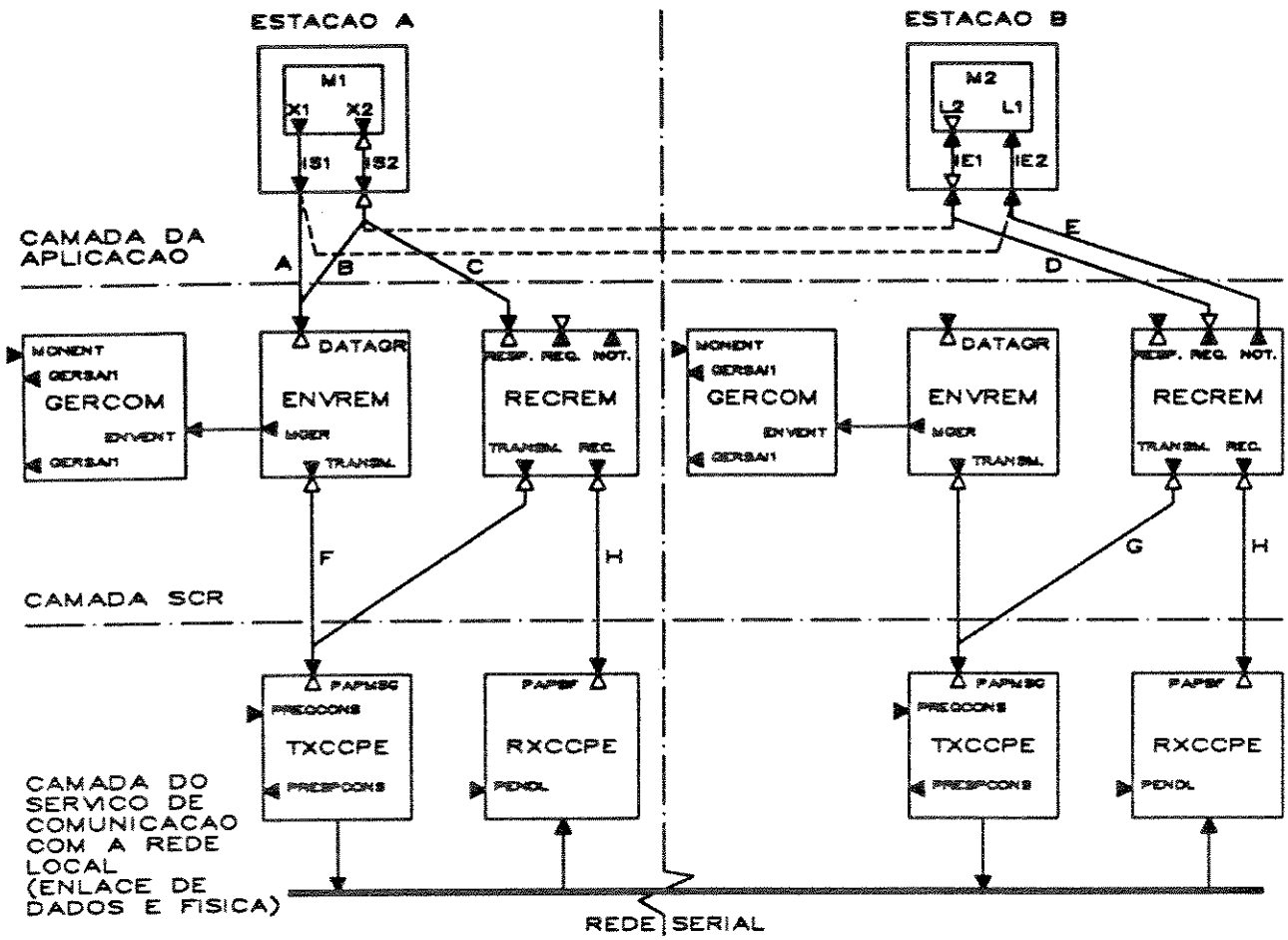

Figura 5.17 : Passagem de Informação entre Camadas Aplicação, SCR e CSCRL

Capítulo 6

Exemplo: Sistema Distribuído de Monitoramento de Pacientes

Este capítulo apresenta a implementação de um sistema de monitoramento de pacientes, de maneira a ilustrar como a programação no Ambiente STER Distribuído pode ser aplicada a situações reais.

Este exemplo, um sistema de monitoramento simplificado retirado da literatura [60], foi estruturado e programado no ambiente STER, de maneira a ressaltar o estilo de desenvolvimento embutido na metodologia e a independência das fases de programação dos módulos em LPM, bem como da configuração da aplicação em LCM, contribuindo para a construção de módulos reusáveis. Os módulos reusáveis são os tipos de módulos e tipos de estações, que podem ser usados, tanto como componentes de sistemas diferentes, quanto componentes de partes diferentes do mesmo sistema.

O objetivo do sistema é monitorar o estado de pacientes em um hospital através da obtenção periódica de fatores a serem controlados. Os fatores dos pacientes são obtidos por intermédio de sensores que medem a sua pressão, temperatura e batimento cardíaco. O sistema lê os fatores periodicamente. As faixas de segurança, para cada fator dos pacientes, são fornecidas no momento da internação do paciente ou durante o período em que está internado, através do teclado conectado ao seu leito. Será enviada uma emissão de alarme para a enfermaria central quando houver desvios das faixas de segurança ou falha dos instrumentos de medição. Há possibilidade de reiniciação das condições de alarme quando a enfermeira atua no paciente, levando-se em conta que a atuação do remédio não é imediata.

6.1 Arquitetura

O sistema distribuído utiliza quatro microcomputadores ligados por uma rede local, três dos quais associados a pacientes internados e outro representando uma estação de monitoramento geral para a enfermaria central (figura 6.1).

Os fatores do paciente, alarmes e indicação de falhas de sensores do paciente são exibidos no monitor de vídeo acoplado ao seu leito periodicamente, recebendo processamento local. Adicionalmente, os alarmes e falhas de sensores de qualquer paciente são transmitidos para a estação enfermaria. Quando solicitado pela estação enfermaria, as informações relativas ao leito especificado são também fornecidas pela estação paciente.

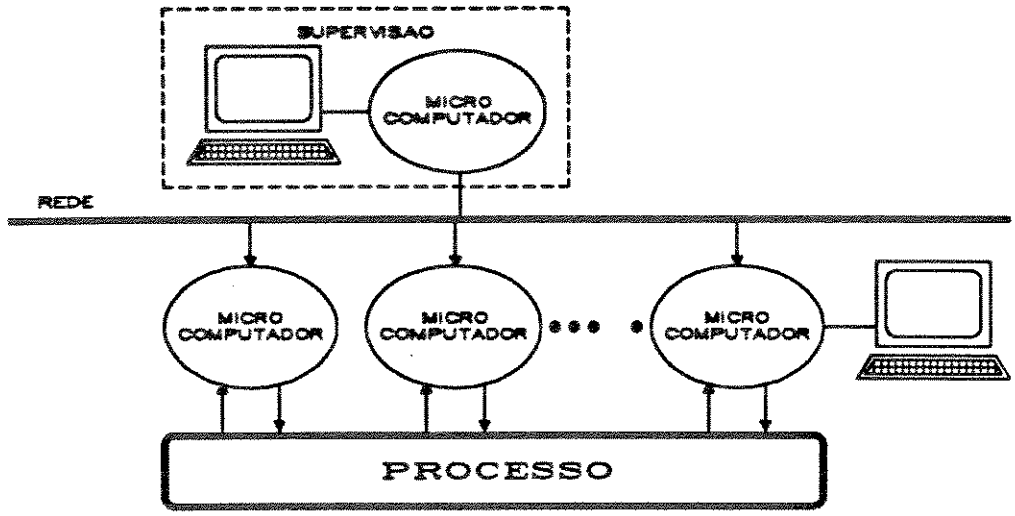

Figura 6.1 : Arquitetura do Sistema de Monitoramento de Pacientes

6.2 Especificação da Configuração Distribuída em STER

A especificação da configuração **NETWORK** (LCMN), figura 6.2, consiste em: declarar as instâncias dos tipos de estações ***PACIENTE*** e ***ENFERMARIA***; criar as instâncias desses tipos de estações fornecendo informações do endereço físico da estação; e estabelecer as conexões entre as estações, através de ligações das portas de interface das estações.

```
NETWORK rede / RS2

INSTANCE
        estacao1 : PACIENTE;
        estacao2 : PACIENTE;
        estacao3 : ENFERMARIA;

CREATE
        estacao1 AT NODE # 1,  {endereço de st. de HW # 1}
        estacao2 AT NODE # 2;  {endereço de st. de HW # 2}
        estacao3 AT NODE # 3;  {endereço de st. de HW # 3}

LINK
        estacao1.Ppacs1     TO    estacao3.Penfe1;
        estacao1.Ppacs2     TO    estacao3.Penfe2;
        estacao2.Ppacs1     TO    estacao3.Penfe3;
        estacao2.Ppacs2     TO    estacao3.Penfe4;
        estacao3.Penfs1     TO    estacao1.Ppace1;
        estacao3.Penfs2     TO    estacao2.Ppace1;

END_NETWORK.
```

Figura 6.2 : Especificação LCMN para Três Estações

O sistema de monitoramento de pacientes caracteriza bem a flexibilidade dos projetos em **STER**. Novas instâncias do tipo ***PACIENTE*** e ***ENFERMARIA*** podem ser facilmente acrescentadas. Uma nova instância do tipo ***PACIENTE*** é acrescentada, conforme ilustrado na figura 6.3.

```
NETWORK rede / RS2

INSTANCE
        estacao1 : PACIENTE;
        estacao2 : PACIENTE;
        estacao3 : ENFERMARIA;
        estacao4 : PACIENTE;

CREATE
        estacao1 AT NODE #1,
        estacao2 AT NODE #2,
        estacao3 AT NODE #3,
        estacao4 AT NODE #4;

LINK
        estacao3.penfs1     TO    estacao1.ppace1;
        estacao3.penfs2     TO    estacao2.ppace1;
        estacao3.penfs3     TO    estacao4.ppace1;
        estacao2.ppacs1     TO    estacao3.penfe3;
        estacao2.ppacs2     TO    estacao3.penfe4;
        estacao1.ppacs1     TO    estacao3.penfe1;
        estacao1.ppacs2     TO    estacao3.penfe2;
        estacao4.ppacs1     TO    estacao3.penfe5;
        estacao4.ppacs2     TO    estacao3.penfe6;

END_NETWORK.
```

Figura 6.3 : Especificação LCMN para Quatro Estações

O sistema foi estruturado, segundo mostra a figura 6.4, simplificada, para duas estações tipo PACIENTE e uma estação tipo ENFERMARIA.

O exemplo é formado por dois tipos de estações: PACIENTE e ENFERMARIA. As estações serão descritas nos próximos itens onde depois de decompostas em sub-funções e identificados os fluxos de informações entre os módulos, programam-se os tipos de módulos, apresentados a seguir, componentes de cada estação.

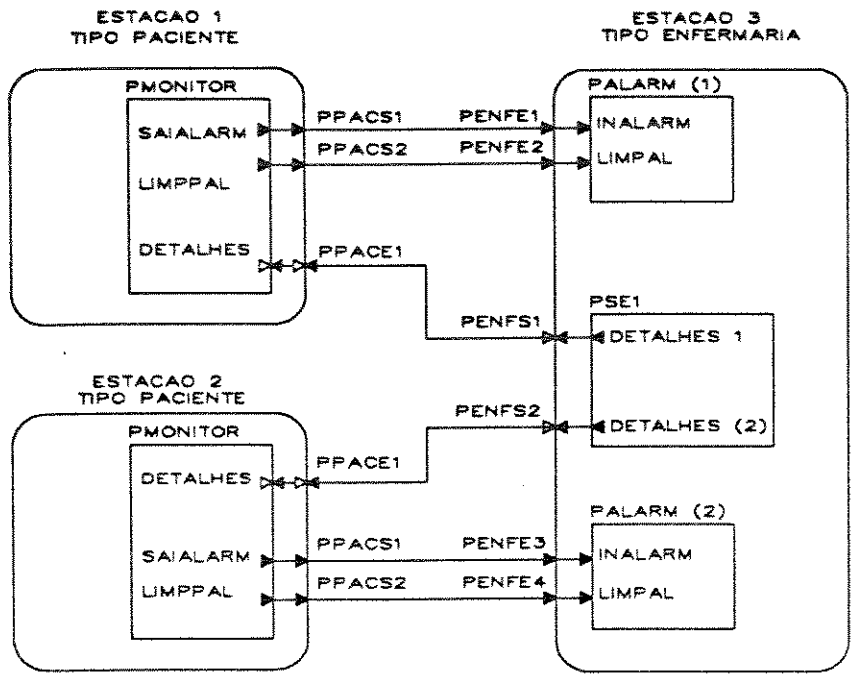

Figura 6.4 : Configuração Distribuída do Sistema

6.3 Estação PACIENTE

A estação Paciente é uma entidade que representa os pacientes do hospital. Cada paciente tem uma estação com processador e um monitor de vídeo com teclado acoplado a seu leito. Cada leito do paciente está ligado a dispositivos que medem, ininterruptamente, os fatores que devem ser monitorados e que, periodicamente, são enviados ao sistema de monitoramento.

Para esse sistema, cada fator do paciente será simulado por um tipo de módulo componente da estação PACIENTE, que gera valores correspondentes às leituras do dispositivo controlado.

As leituras atualizadas periodicamente dos fatores de cada paciente, bem como as suas informações cadastrais são exibidas no seu respectivo monitor de vídeo.

Podem-se alterar os dados cadastrados dos pacientes quando há alteração de um paciente em um determinado leito ou quando há alterações nas características patológicas de um paciente.

Para cada paciente são especificados os limites de segurança de cada fator. Se um fator ultrapassa os limites de segurança ou se um dos instrumentos de medição falhar, a estação ENFERMARIA é notificada.

Quando ocorre um alarme na estação ENFERMARIA, esta notifica uma enfermeira de plantão para atuar no paciente em condições críticas. Após tomar as providências necessárias, a enfermeira envia um sinal ao sistema para confirmar sua atuação no paciente, através do teclado acoplado ao leito do paciente.

A estação PACIENTE é formada por oito tipos de módulos. A função principal de cada tipo de módulo, a sua interface e configuração a nível STATION são descritas a seguir.

6.3.1 Módulo PSIMUL

O módulo PSIMUL simula os fatores (entradas analógicas) dos pacientes a serem controlados. Periodicamente (a periodicidade é determinada por taxa) envia um conjunto de leituras simuladas dos sensores para a sua porta "saisensor". Quando um determinado sensor estiver desligado, o valor zero é gerado. Quando um sensor estiver no estado congelado, o valor anteriormente gerado é mantido até que o sensor especificado seja ativado, em sua porta "s_com", voltando a gerar os dados do sensor normalmente.

A sua interface, descrita em LPM, é a seguinte:

```
MODULE psimul (taxa : INTEGER);

    USE undefrs2.inc;
    ENTRYPORT
        s_com[0..2] : t_s_com;
    EXITPORT
        saisensor : tipoleitura;

END_MODULE.
```

6.3.2 Módulo PDISP

O módulo PDISP requisita informações do paciente através da sua porta "detalhes", e atualiza-as periodicamente (periodicidade determinada por taxa) no formato padrão de exibição, para serem enviadas para exibição através de sua porta "saida".

A sua interface, escrita em LPM, é a seguinte:

```
MODULE pdisp (taxa : INTEGER);

    USE undefrs2.inc;
    EXITPORT
        detalhes : tiposignal REPLY tipoleito;
        saida : tipio REPLY tiposignal;

END_MODULE.
```

6.3.3 Módulo PJANELA

O módulo PJANELA proporciona a capacidade de particionar a tela em um conjunto de janelas distintas, identificadas pelo parâmetro de entrada "num". Em sua porta "janela" recebe cadeias de caracteres a serem exibidas na área correspondente à sua janela (definida durante a configuração). Essa cadeia de caracteres pode ser de dois tipos: dados a serem exibidos e caracteres de controle (limpa tela,

pula de linha,etc.). Envia a posição do cursor através de sua porta "poslcol". Esse módulo utiliza, da biblioteca do STER, um pacote gráfico formado por rotinas escritas em linguagem PASCAL e ASSEMBLER.

A sua interface, escrita em LPM, é a seguinte :

```
MODULE pjanela (num,cl1,cl2,l1,l2,atr : INTEGER);

    USE undefrs2.inc;
    ENTRYPORT
        janela : tlstring REPLY tiposignal;
    EXITPORT
        poslcol : tlincol;

END_MODULE.
```

6.3.4 Módulo PMONITOR

O módulo PMONITOR monitora os valores de sensor gerando alarmes quando o valor do sensor estiver fora do limite especificado para um paciente ou se um sensor falhar.

Os dados recebidos na porta "entsensor" são comparados com os limites de segurança, gerando alarmes para a estação ENFERMARIA através do envio de um alarme na porta "saialarm" quando estiver fora dos limites e, também, uma mensagem para congelar o valor associado ao respectivo sensor através da porta "s_alarm".

O "status" dos fatores que se encontram fora dos limites de segurança são guardados para quando da ocorrência de notificação de atuação da enfermeira no paciente através da porta "atuapac". A atuação ocasiona o envio de uma mensagem para desligar o sensor pelo tempo determinado em tempo de configuração. Um sinal é enviado para a porta "limpal" para reiniciar a janela associada ao paciente na estação ENFERMARIA.

As mudanças relativas aos dados de um paciente são recebidas através da porta "mudapac". Todas as informações do paciente, armazenadas pelo módulo, tornam-se disponíveis via requisições através da porta "detalhes".

A sua interface, escrita em LPM, é a seguinte :

```
MODULE pmonitor;
    USE undefrs2.inc;
    ENTRYPORT
        entsensor : tipoleitura;
        mudapac : tipopaciente REPLY tiposignal;
        detalhes : tiposignal REPLY tipoleito;
        atuapac : tiposignal;
    EXITPORT
        saialarm : tipoalarmes;
        s_alarm[0..2] : t_s_com;
        limpal : tiposignal;

END_MODULE.
```

6.3.5 Módulo PCOM

O módulo PCOM gerencia os aspectos de interface Homem/Máquina com os "menus" enviados através da porta "saistr", bem como gerencia as informações recebidas através de sua porta "enstr". Os novos dados do paciente são enviados através de sua porta "novopac". Recebe, através de sua porta "atuapac", o sinal da enfermeira que atuou sobre o paciente, administrando o remédio necessário.

A sua interface, escrita em LPM, é a seguinte :

```
MODULE pcom;

    USE undefrs2.inc;
    ENTRYPORT
        enstr : tipopaciente REPLY tiposignal;
    EXITPORT
        saistr : tipio REPLY tiposignal;
        novopac : tipopaciente REPLY tiposignal;
        atuapac : tiposignal;

END_MODULE.
```

6.3.6 Módulo PCONSOLE

O módulo PCONSOLE permite que novos dados do paciente, tais como: limites de segurança e nome do paciente, sejam fornecidos através do teclado associado a estação PACIENTE; usa, para isto, os serviços disponíveis pelo MS-DOS.

Esse módulo recebe também o sinal de atuação da enfermeira no paciente.

A sua interface, escrita em LPM, é a seguinte :

```
MODULE pconsole;

    USE undefrs2.inc;
    ENTRYPORT
        saistr : tipio REPLY tiposignal;
    EXITPORT
        enstr : tipopaciente REPLY tiposignal;
        saijan : tlstring REPLY tiposignal;
        poslcol : tlincol;

END_MODULE.
```

6.3.7 Módulo ATIVA_SENSOR

O módulo ATIVA_SENSOR espera a mensagem de comando em sua porta "s_alarm", que vai determinar qual a ação a ser enviada ao módulo psimul, pela porta "s_com". Se o comando recebido indicar para desligar o sensor, este módulo fica em espera por tempo (DELAY), por um período determinado por pausa, em tempo de configuração, onde determina-se o tempo em que o sensor permanecerá desligado após a atuação da enfermeira no paciente em condição crítica.

A sua interface, escrita em LPM, é a seguinte :

```
MODULE Ativa_Sensor (pausa : INTEGER);

    USE undefrs2.inc;
    ENTRYPORT
        s_alarm : t_s_com;
    EXITPORT
        s_com : t_s_com;

END_MODULE.
```

6.3.8 Módulo GERSAIDA

O módulo GERSAÍDA gerencia as janelas de exibição do vídeo enviando as mensagens específicas a cada janela.

A sua interface, escrita em LPM, é a seguinte:

```
MODULE gersaida;

    USE undefrs2.inc;
    ENTRYPORT
        entstr : tipio REPLY tiposignal;
    EXITPORT
        saistr[1..3] : tistring REPLY tiposignal;

END_MODULE.
```

6.3.9 Configuração a Nível STATION da Estação PACIENTE

A especificação da configuração (LCMS) do tipo de estação PACIENTE é mostrada na figura 6.5.

Um exemplo do formato de exibição para a estação PACIENTE é ilustrado na figura 6.6.

```
STATION PACIENTE;

    EXITPORT
        ppacs1 : alarmstype;
        ppacs2 : tiposignal;

    ENTRYPORT
        ppace1 : tiposignal REPLY bedtype;

    INSTANCE
        i0 : psimul;
        a1 : pdisp;
        b1,b4,b5,b6 : pjanela;
        d1 : pcom;
        e1 : pmonitor;
        f1 : gersaida;
        h1 : pconsole;
        ats1,
        ats2,
        ats3 : ativa_sen;

    CREATE
        i0(500),      a1(600),
        b1(1,15,54,2,8,63) /S=64 /H=2, b4(4,10,75,10,14,63) /S=64 /H=2,
        b5(5,10,75,17,24,63) /S=64 /H=2,      b6(6,55,70,2,8,191) /S=64 /H=2,
        d1,    e1,
        f1,    h1 /P=LOWPR /S=64 /H=2,
        ats1(100),   ats2(150),   ats3(170);

    LINK
        a1.detalhes     TO   e1.detalhes;     a1.saida        TO   f1.entstr;
        d1.saistr       TO   h1.saistr;       d1.novopac      TO   e1.mudapac;
        d1.atuapac      TO   e1.atuapac;      f1.saistr[1]    TO   b1.janela;
        f1.saistr[2]    TO   b4.janela;       f1.saistr[3]    TO   b6.janela;
        h1.enstr        TO   d1.enstr;        h1.saijan       TO   f1.entstr;
        h1.poslcol      TO   b6.poslcol;      i0.saisensor    TO   e1.entsensor;
        ats1.s_com[1]   TO   i0.scom[0];      ats2.s_com[2]   TO   i0.scom[1];
        ats3.s_com[3]   TO   i0.scom[2];      e1.s_alam0      TO   ats1.s_alarm;
        e1.s_alam1      TO   ats1.s_alarm;    e1.s_alam2      TO   ats1.s_alarm;

    ASSOCIATE
        e1.detalhes     WITH   ppace1;
        e1.saialarm     WITH   ppacs1;
        e1.limpal       WITH   ppacs2;

END_STATION.
```

Figura 6.5 : Especificação LCMS para Estação PACIENTE

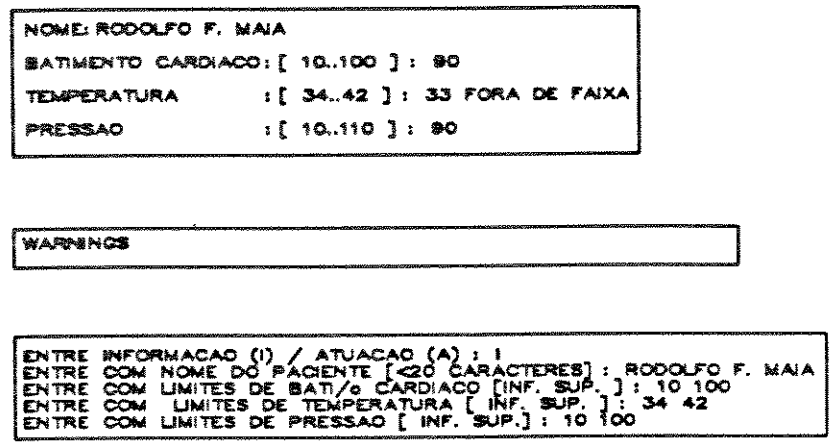

Figura 6.6 : Formato de Exibição para Estação PACIENTE

6.4 Estação ENFERMARIA

A estação ENFERMARIA é uma entidade que representa a enfermaria central, composta de um processador e um monitor de vídeo com teclado. É responsável pela exibição dos alarmes gerados pelas estações dos pacientes, quando ocorrem desvios das faixas de segurança dos fatores monitorados para os pacientes ou quando ocorre a falha de um dos instrumentos. Por solicitação da estação ENFERMARIA são exibidas as informações de um paciente para um determinado leito, selecionado através do teclado. Esta estação exibe no monitor de vídeo exatamente as mesmas informações que são exibidas no monitor de vídeo da estação que monitora o paciente.

A estação ENFERMARIA é formada por sete tipos de módulos. A função principal de cada tipo de módulo, a sua interface e configuração a nível STATION são descritas a seguir.

6.4.1 Módulo PSEL

O módulo PSEL requisita informações do paciente, através de uma de suas portas "detalhes n" (uma para cada estação PACIENTE pertencente a rede) para o módulo PMONIT da estação PACIENTE, dependendo do valor especificado pela enfermeira e recebida através da porta "selpac". As informações tornam-se disponíveis para serem solicitadas, na porta "infpac", pelo módulo encarregado de exibi-las na tela.

O módulo espera uma resposta a sua requisição por informações do paciente, por um período de tempo, exibindo a mensagem indicativa de erro na janela de "warnings", caso a resposta não chegue.

A sua interface, escrita em LPM, é a seguinte :

```
MODULE psel (periodo : INTEGER);

    USE undefrs2.inc;
    ENTRYPORT
        selpac : INTEGER;
        infpac : tiposignal REPLY tipoleito;
    EXITPORT
        detalhes[1..2] : tiposignal REPLY tipoleito;
        p_erro : tipio REPLY tiposignal;

END_MODULE.
```

6.4.2 Módulo PENFERMARIA

O módulo PENFERMARIA gerencia os aspectos de interface Homem/Máquina, com "menus" enviados através da porta "saistr" e o número do leito requisitado através de sua porta "enstr".

Esse módulo executa a função de interpretador de comando, permitindo que uma enfermeira selecione um leito para obter informações para a supervisão dos pacientes.

O número do leito selecionado é enviado para o módulo psel através da porta "selpac".

A sua interface, escrita em LPM, é a seguinte:

```
MODULE penfermaria;

    USE undefrs2.inc;
    ENTRYPORT
        enstr : INTEGER REPLY tiposignal;
    EXITPORT
        saistr : tipio REPLY tiposignal;
        selpac : INTEGER;

END_MODULE
```

6.4.3 Módulo PALARM

O módulo PALARM recebe as condições de alarme do paciente (numleito) através da porta "inalarm" e exibe na janela correspondente ao paciente na estação ENFERMARIA através da porta "saistring".

Esse módulo reinicializa a janela correspondente ao receber um sinal na porta "limpal".

A sua interface, escrita em LPM, é a seguinte :

```
MODULE palarm (numleito);

    USE undefrs2.inc;
    ENTRYPORT
        inalarm : tipoalarmes;
        limpal : tiposignal;
    EXITPORT
        saistring : tipio REPLY tiposignal;

END_MODULE.
```

6.4.4 Módulo CONSOLE

O módulo CONSOLE recebe o número do leito pelo teclado, digitado pela enfermeira que deseja maiores informações a respeito do paciente que ocupa o leito especificado.

A sua interface, escrita em LPM, é a seguinte :

```
MODULE Console ;

    USE undefrs2.inc;
    ENTRYPORT
        saistr : tlincol;
    EXITPORT
        enstr : INTEGER REPLY tiposignal;
        poslcol : tlincol;
        saijan : tipio REPLY tiposignal;

END_MODULE.
```

6.4.5 Módulos PDISP, JANELA e GERSAIDA

Os módulos PDISP, JANELA e GERSAIDA são os mesmos utilizados pelo sistema na estação PACIENTE, sendo, portanto, reutilizados pela estação ENFERMARIA. Estes módulos, podendo ser usados em vários lugares do sistema, reduzem o esforço de programação.

6.4.6 Configuração a Nível STATION da Estação ENFERMARIA

A especificação da configuração (LCMS) do tipo de estação ENFERMARIA é mostrada na figura 6.7.

```
STATION ENFER;
    EXITPORT
        Penfs1 : tiposignal REPLY tipoleito;
        Penfs2 : tiposignal REPLY tipoleito;

    ENTRYPORT
        Penfe1, Penfe3 : tipoalarmes;
        Penfe2, Penfe4 : tiposignal;

    INSTANCE
        a1 : pdisp;
        b1,b2,b3,b4,b5,b6 : pjanela;
        c1 : psel;
        d1 : penfermaria;
        f1 : gersaida;
        g1,g2 : palarm;
        h1 : pconsole;

    CREATE
        a1(600),
        b1(1,15,54,1,6,63) /S=64 /H=2,          b2(2,17,37,8,12,191) /S=64 /H=2,
        b3(3,48,68,8,12,191) /S=64 /H=2,        b4(4,10,75,14,16,242) /S=64 /H=2,
        b5(5,10,75,18,24,63) /S=64 /H=2,        b6(6,55,70,1,6,63) /S=64 /H=2,
        c1(600),   d1,   f1,
        g1(1),     g2(2),     h1 /P=LOWPR /S=64 /H=2;

    LINK
        a1.detalhes     TO   c1.infpac;        a1.saida        TO   f1.entstr;
        d1.selpac       TO   c1.selpac;        c1.p_erro       TO   f1.entstr;
        g1.saistring    TO   f1.entstr;        g2.saistring    TO   f1.entstr;
        d1.saistr       TO   h1.saistr;        f1.janela1      TO   b1.janela;
        f1.janela2      TO   b2.janela;        f1.janela3      TO   b3.janela;
        f1.janela4      TO   b4.janela;        f1.janela5      TO   b5.janela;
        f1.janela6      TO   b6.janela;        h1.enstr        TO   d1.enstr;
        h1.poslcol      TO   b5.poslcol;       h1.saijan       TO   f1.entstr;

    ASSOCIATE
        c1.detalhes[1]  WITH   Penfs1;
        c1.detalhes[2]  WITH   Penfs2;
        g1.inalarm      WITH   Penfe1;
        g1.limpal       WITH   Penfe2;
        g2.inalarm      WITH   Penfe3;
        g2.limpal       WITH   Penfe4;

END_STATION.
```

Figura 6.7 : Especificação LCMS para a Estação ENFERMARIA

Um exemplo do formato de exibição para a estação ENFERMARIA é ilustrado na figura 6.8.

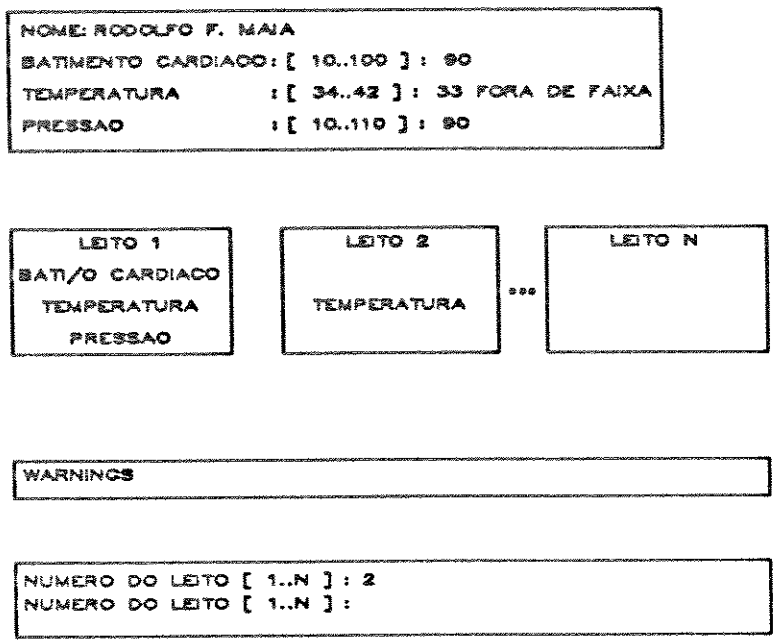

Figura 6.8 : Formato de Exibição para a Estação ENFERMARIA

6.5 Tipos de Mensagens do Sistema de Monitoramento de Pacientes

Os tipos de mensagens utilizados pelo sistema aplicativo de monitoramento de pacientes, pertencentes ao arquivo "undefrs2.def", são apresentados na figura 6.9.

6.6 Diagrama de Configuração do Sistema de Monitoramento de Pacientes

O diagrama completo da configuração do sistema de monitoramento de pacientes, ilustrando uma versão simplificada com duas estações do tipo PACIENTE e uma estação do tipo ENFERMARIA é apresentado na figura 6.10

O sistema de monitoramento de pacientes ilustra como sistemas complexos podem ser construídos de um conjunto de módulos simples. O texto de programa para cada tipo de módulo contém uma ou, no máximo, duas páginas. A independência da configuração dos módulos em STER permite o seu uso em mais do que uma parte do sistema.

```
TYPE

    oknoktipo = (ok,notok,desligado);
    tiposensor = (pressão,temperatura,batimentocard);
    tipoleit =          RECORD
                            xestado : oknoktipo;
                            xvalor  : INTEGER;
                        END;
    tipoleitura = ARRAY[tiposensor] OF tipoleit;
    tipolimite =        RECORD
                            supr,infr : INTEGER;
                        END;
    tipopaciente =      RECORD
                            nome : lstring(20);
                            lims : ARRAY[tiposensor] OF tipolimite;
                        END;
    tipoalarm = (foralimite,falhasensor,semalarme);
    tipoalarmes = ARRAY[tiposensor] OF tipoalarm;
    tipoleito =         RECORD
                            paciente : tipopaciente;
                            leituras : tipoleitura;
                            alarmes  : tipoalarmes;
                        END;
    t_s_com = (liga,desliga,congela);
    tipio =             RECORD
                            numio : INTEGER;
                            msg   : lstring(60);
                        END;
    tlincol =           RECORD
                            l1, col1 :INTEGER;
                        END;
    tipofatores =       RECORD
                            leituras : tipoleitura;
                            alarmes  : tipoalrmes;
                        END;
```

Figura 6.9 : Tipos de Mensagens do Sistema

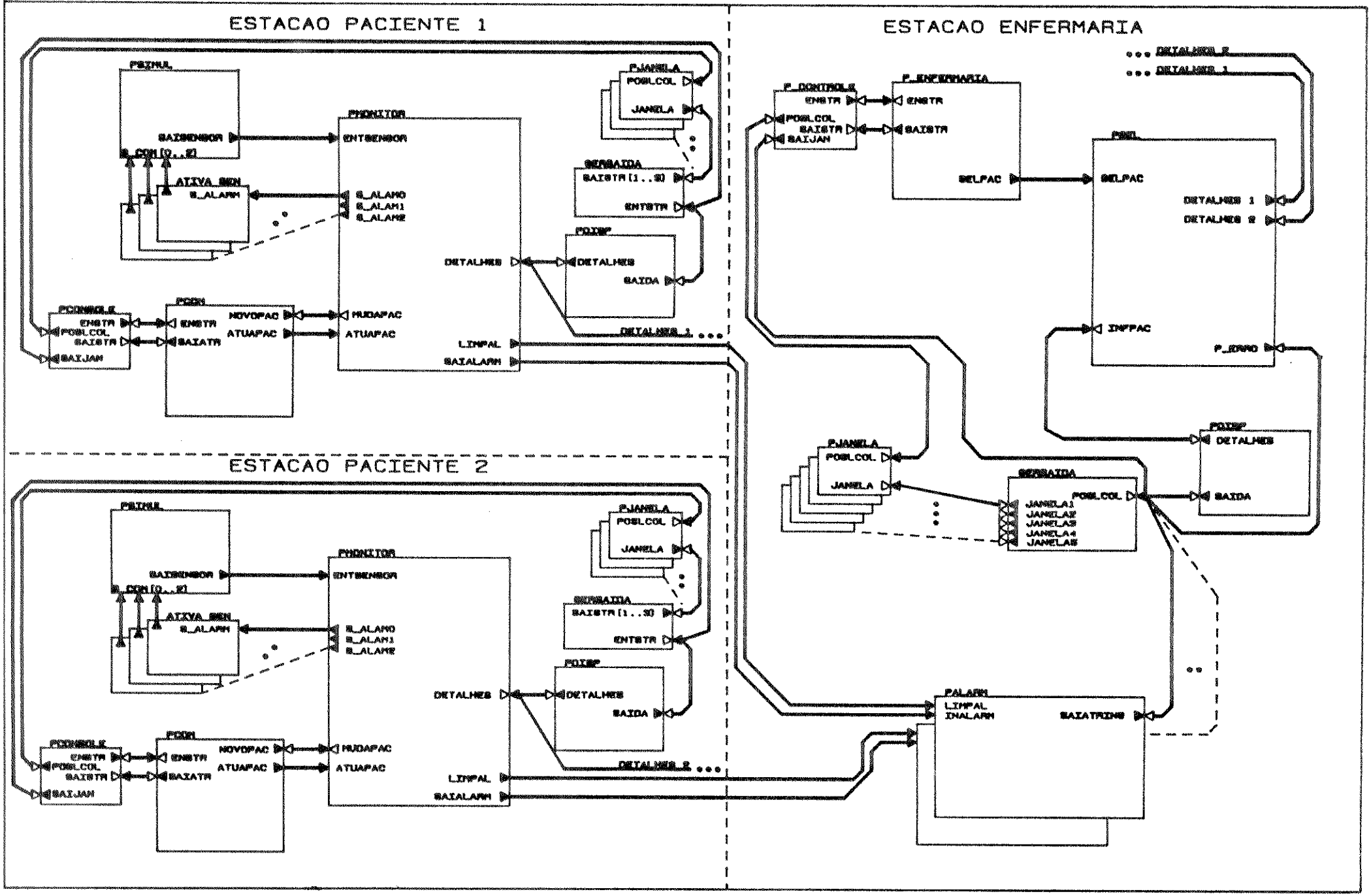

Figura 6.10 : Diagrama de Configuração do Sistema de Monitoramento de Pacientes

Capítulo 7

Conclusão

Este trabalho apresentou a implementação da distribuição de um ambiente para o desenvolvimento de software de tempo real - STER.

O desenvolvimento de sistemas distribuídos de tempo real pode ser grandemente simplificado se dispusermos de ambientes orientados a linguagem de alto nível, constituídos de ferramentas que possibilitem uma programação de software altamente modular, reconfigurável e independente da arquitetura de hardware onde será executado.

A versão atualmente implementada do STER suporta a configuração estática das aplicações distribuídas. A utilização dessa versão tem demonstrado a sua importância no desenvolvimento de aplicações de tempo real, centralizadas e distribuídas, tendo sido constatada, através dos laboratórios, para alunos de graduação e de pós-graduação, e de experimentos realizados, a sua simplicidade de utilização, inclusive para usuários não familiarizados com a programação concorrente.

A independência das etapas de programação dos módulos e de configuração da aplicação distribuída tem permitido o desenvolvimento de sistemas distribuídos de fácil projeto, implementação, manutenção e expansão. Para isso, são fornecidas duas linguagens: a Linguagem de Programação de Módulos (LPM) e a Linguagem de Configuração de Módulos (LCM). Essa característica possibilita a incorporação de mecanismos de reconfiguração dinâmica e, conseqüentemente, de tolerância a falhas.

No ambiente STER, as aplicações distribuídas e o próprio sistema de suporte do ambiente e das ferramentas são construídos usando-se as mesmas ferramentas e técnicas. Com a exceção do núcleo STRE, todo o software de comunicação foi implementado através das linguagens LPM e LCM. Por exemplo, novos "drives" de comunicação específicos para redes locais diferentes e utilitários podem ser escritos usando-se as linguagens oferecidas. Essa uniformidade permite aos usuários estender as facilidades oferecidas de maneira a satisfazer os seus requisitos (embora esse tipo de alteração não seja feita por programadores comuns e sim por especialistas em implementação de sistemas distribuídos experimentais).

A presente versão do Sistema Distribuído STER (SDS) foi realizada utilizando, para cada estação, um microcomputador compatível com o IBM-PC. A razão dessa escolha deveu-se à grande popularidade dos compatíveis com o PC de uma maneira geral, bem como à sua penetração nos ambientes industriais. Com a implementação do SDS nesse equipamento, fornece-se uma opção para o desenvolvimento de aplicações distribuídas de tempo real voltadas ao controle e supervisão de processos. A interligação das estações físicas foi realizada através de uma rede local simplificada, desenvolvida especificamente para testes dessa versão.

Uma evolução do trabalho, em uma próxima versão, é o desenvolvimento de um "driver" de comunicação para uma rede local disponível comercialmente. Conseqüentemente, a carga da aplicação será feita através dessa rede.

A nível de continuidade do trabalho apresentado, prevê-se uma fase de testes no CTI e na UNICAMP.

Paralelamente a essa versão distribuída do ambiente STER, estão sendo realizados estudos, no CTI e na UNICAMP, visando a evolução do ambiente STER, segundo aspectos que têm merecido a atenção dos grupos de pesquisa:

- A Reconfiguração Dinâmica das Aplicações;
- A inclusão da variável tempo no processo de desenvolvimento das aplicações ("Hard Real Time");
- O transporte do ambiente STER para ambientes UNIX (estações SUN);
- O suporte para arquiteturas heterogêneas;
- Prototipagem de Especificações escritas em LOTOS;
- Tolerância a Falhas.

Apêndice A

Descrição BNF das Linguagens LPM e LCM

Neste apêndice, são apresentadas as descrições sintáticas das linguagens LPM e LCM. Para representação da sintaxe, utiliza-se uma notação BNF estendida, onde:

- [<construção>] : representa uma construção opcional;
- [<construção>]* : representa zero ou mais repetições da construção

Na descrição correspondente à LPM, apenas serão apresentadas as construções das extensões à linguagem PASCAL, as quais constituem a secção 5.1 da dissertação de mestrado desenvolvida no CTI/UNICAMP [41]. As contruções não definidas neste apêndice podem ser obtidas na referência [29]. Na descrição correspondente à LCM serão apresentadas as extensões realizadas na versão distribuída.

A.1 LINGUAGEM DE PROGRAMAÇÃO DE MÓDULOS

A.1.1 Definição De Um Módulo

```
<definição de um módulo> ::=
        MODULE <identificador> [(<parâmetros formais>)];
                [<importação definições>]
                [<declaração de portas>]
                [<declaração de mensagens>]
                <bloco LPM>.

<parâmetros formais> ::=
        <grupo de parâmetros> [; <grupo de parâmetros>]

<grupo de parâmetros> ::=
        <lista de identificadores> : <tipo básico PASCAL>
```

```
bloco LPM> ::=
        [<declaração de rótulos>]
        [<declaração de constantes>]
        [<declaração de tipos>]
        [<declaração de variáveis>]
        [<declaração de funções e procedimentos>]
        BEGIN_MODULE <comandos>
END_MODULE
```

A.1.2 Importação De Definições

```
<importação definições> ::=
        USE <nome arquivo definições>;
```

A.1.3 Declaração De Portas

```
<declaração de portas> ::=
        ENTRYPORT <decl. porta entrada>
                [; <decl. porta entrada>]*; |
        EXITPORT  <decl. porta saída> [; <decl. porta saída>]* ;
```

A.1.4 Declaração Porta Entrada

```
<decl. porta entrada> ::=
        <declarador> [, <declarador>] : <tipo mensagem>
                [<parte reply>  |  <parte queue>]

<declarador> ::=
        <identificador> [ "[" <família> "]" ]

<família> ::=
        <inteiro decimal> .. <inteiro decimal>

<tipo mensagem> ::=
        <tipo básico PASCAL>  |  <identificador tipo mensagem>

<parte reply> ::=
        REPLY <tipo mensagem>

<parte queue> ::=
        QUEUE <inteiro decimal>
```

A.1.5 Declaração Porta Saída

```
<decl. porta saída> ::=
        <declarador> [, <declarador>] : <tipo mensagem>
                [<parte reply>]
```

A.1.6 Declaração De Mensagens

```
<declaração de mensagens> ::=
        MESSAGE <declarador de mensagens>
                [; <declarador de mensagens>];

<declarador de mensagens> ::=
        <identificador> [, <identificador>] : <tipo mensagem>
```

A.1.7 Comandos

```
<comandos> ::=
        <comando PASCAL> | <comando LPM>
<comando LPM> ::=
        <comando LOOP> | <comando SEND> |
        <comando RECEIVE> | <comando REPLY>
        <comando SELECT> | <comando FORWARD>
        <comando EXIT>
```

A.1.7.1 Comando LOOP

```
<comando LOOP> ::=
        LOOP <comandos> END_LOOP
<comando EXIT> ::= EXIT
```

A.1.7.2 Comando SEND

```
<comando SEND> ::=
        SEND <ident. mensagem> TO <ident. porta saída>
                [<cláusula resposta>]

<cláusula resposta> ::=
        WAIT <ident. mensagem> [<opções de resposta> END_SEND]

<opções de resposta> ::=
        => <comandos> [<tratamento falha>] | <tratamento falha>

<tratamento falha> ::=
        FAIL [<tempo>] => <comandos>

<tempo> ::= <inteiro decimal>
```

A.1.7.3 Comando RECEIVE

```
<comando RECEIVE> ::=
        RECEIVE <ident. mensagem> FROM <ident. porta entrada>
                [REPLY <ident. mensagem>]
```

A.1.7.4 Comando REPLY

```
<comando REPLY> ::=
        REPLY <ident. mensagem> TO <ident. porta entrada>
```

A.1.7.5 Comando SELECT

```
<comando SELECT> ::=
        <tipo de seleção> <parte select>
                [OR_SELECT <parte select>]*
                [else_select <comandos>]
        END_SELECT

<tipo de seleção> ::=
        PSELECT | RSELECT

<parte select> ::=
        [<repetição>] [<guarda>] <cláusula de seleção>
                [⇒ <comandos>]

<repetição> ::=
        FOR <variável de controle> := <valor inicial>
                (TO | DOWNTO) <valor final> DO

<guarda> ::=
        WHEN <expressão booleana>

<cláusula de seleção> ::=
        <comando RECEIVE> | <cláusula de tempo>

<cláusula de tempo> ::=
        TIMEOUT <unidades de tempo>

<unidades de tempo> ::=
        <expressão inteira>
```

A.1.7.6 Comando FORWARD

```
<comando FORWARD> ::=
        FORWARD <ident. porta entrada> TO <ident. porta saída>
```

A.2 LINGUAGEM DE CONFIGURAÇÃO DE MÓDULOS

A.2.1 Programa de Configuração Nível Construção STATION

Este item apresenta a estrutura de uma especificação a nível da construção STATION para a LCM da versão distribuída.

ESPECIFICAÇÃO STATION

<especificação STATION> ::=
STATION <identificador>;
[<definição da interface da estação>]
<declaração de instâncias de módulos>
<criação de instâncias de módulos>
[<conexão de instâncias de módulos>]
[<associação de portas a interface da estação>]
END_STATION.

A.2.1.1 Definição da Interface da Estação

<definição da interface da estação> ::=
<declaração de portas>

<declaração de portas> ::=
ENTRYPORT <decl.porta_interf._estação>
[; <decl.porta_interf._estação>]*; |
EXITPORT <decl.porta_interf._estação>
[; <decl.porta_interf._estação>]*;

<decl.porta_interf._estação> ::=
<declarador> [,<declarador>] : <tipo mensagem>
[<parte reply>]

<declarador> ::=
<identificador> ["[" <família> "]"]

<família> ::=
<inteiro decimal> .. <inteiro decimal>

<tipo mensagem> ::=
<tipo básico PASCAL> | <identificador tipo mensagem>

<parte reply> ::=
REPLY <tipo mensagem>

A.2.1.2 Declaração de Instâncias de Módulos

<declaração de instâncias de módulos> ::=
INSTANCE <declarador instâncias>
[;<declarador instâncias>]*;
<declarador instâncias> ::=
<identificador> [, <identificador>]* : <tipo do módulo>

A.2.1.3 Criação de Instâncias de Módulos

<criação de instâncias de módulos> ::=
CREATE <instância_módulo> [, <instância_módulo>]*;

```
<instância_módulo> ::=
        <identificador instância módulo>
                [(<parâmetros reais módulo>)]

<parâmetros reais módulo> ::=
        <constante inteira> | <constante real>
        <constante lógica> | <carater>
```

A.2.1.4 Conexão de Instâncias de Módulos

```
<conexão de instâncias de módulos> ::=
        LINK <porta saída> TO <porta entrada>
                [, <porta entrada>]*;
```

A.2.1.5 Porta Saída e Porta Entrada

```
<porta saída> ::=
        <identificador instância módulo> . <ident. porta módulo>
                [ "[" <índice> "]" ]

<porta entrada> ::=
        <identificador instância módulo> . <ident. porta módulo>
                [ "[" <índice> "]" ]
```

A.2.1.6 Associação de Portas a Interface da Estação

```
<associação portas a interface estação> ::=
        ASSOCIATE [<porta_módulo> WITH <id_porta_int_estação>]*;

<porta_módulo> ::=
        <identificador instância módulo> . <ident. porta módulo>
                [ "[" <índice> "]" ]

<id_porta_int_estação> ::=
        <identificador>  "[" <índice> "]"
                [ "[" <índice> "]" ]*
```

A.2.2 Programa de Configuração Nível Construção NETWORK

Este item apresenta a estrutura de uma especificação a nível da construção NETWORK para a LCM da versão distribuída.

ESPECIFICAÇÃO NETWORK

```
<especificação NETWORK> ::=
        NETWORK <identificador> [<diretiva de rede>];
                <declaração de instâncias de estações>
                <criação de instâncias de estações>
                [<conexão de instâncias de estações>]
        END_NETWORK
```

A.2.2.1 Diretiva de Rede

<diretiva de rede> ::= { } |
    <especificação sistema comunicação>

<especificação sistema comunicação> ::= "/" RSn

A.2.2.2 Declaração de Instâncias de Estações

<declaração de instâncias de estações> ::=
    INSTANCE <declarador instâncias>
        [;<declarador instâncias>]*;

<declarador instâncias> ::=
    <identificador> [, <identificador>]* : <tipo de estação>

A.2.2.3 Criação de Instâncias de Estações

<criação de instâncias de estações> ::=
    CREATE
        [<identificador instância estação> <diretiva buffers>
            AT NODE <endereço estação>;]*

<identificador instância estação> ::=
    <identificador>

<diretiva buffers> ::= "|" B "=" <inteiro decimal>

<endereço estação> ::= "#" <inteiro decimal>

A.2.2.4 Conexão de Instâncias de Estações

<conexão de instâncias de estações> ::=
    LINK
        [<porta saída estação> TO <porta entrada estação>
            [, <porta entrada estação>]*;

<porta saída estação> ::=
    <identificador instância estação> .
    <ident. porta saída estação>

<porta entrada estação> ::=
    <identificador instância estação> .
    <ident. porta entrada estação>

<ident. porta entrada estação> ::= <identificador>

<ident. porta saída estação> ::= <identificador>

Apêndice B

Serviços Oferecidos Pelo Suporte de Tempo Real (STRE)

Este apêndice apresenta uma descrição da interface dos serviços do núcleo tempo real, a qual constitui a secção 4.1 da dissertação de mestrado desenvolvida no CTI/UNICAMP [1].

B.1 TROCA DE MENSAGENS

Os serviços de troca de mensagens suportam a execução das primitivas e funções relacionadas ao envio e recepção de mensagens através das quais os módulos se comunicam e sincronizam.

B.1.1 Envio Assíncrono de uma Mensagem

O comando de envio assíncrono de uma mensagem

SEND <mensagem> to <porta_de_saída>

é suportado pelo serviço *env_a* cuja interface é dada por:

```
PROCEDURE env_a (prt_saída, tam_men: INTEGER; end_mem: ADDRESS);
```

onde *prt_saída* é o número correspondente à <porta_de_saída> através da qual a <mensagem> deve ser enviada; *tam_men* é o tamanho da <mensagem> a enviar e *end_mem* é a sua posição no espaço de endereçamento do módulo emissor.

B.1.2 Envio Síncrono de uma Mensagem

O comando de envio síncrono de uma mensagem

SEND <mensagem> TO <porta_de_saída> WAIT <resposta>

é suportado pelo serviço *env_s* cuja interface é dada por:

```
PROCEDURE env_s (prt_saída, tam_men: INTEGER; end_mem, end_resp: ADDRESS);
```

onde os parâmetros *prt_saída, tam_mem* são idênticos, em significado, aos seus correspondentes no serviço *env_a*. O parâmetro *end_resp* especifica o endereço onde deverá ser copiada a mensagem de <resposta>.

B.1.3 Envio Síncrono de uma Mensagem com Cláusula para Tratamento de Falha

O comando de envio síncrono de uma mensagem, com cláusula para tratamento de falha,

```
SEND <mensagem> TO <porta_de_saída>
        WAIT <resposta> => S1
        FAIL <tempo de espera> => S2
END
```

é suportado pelo serviço *env_s_temp* cuja interface é dada por:

```
FUNCTION  env_s_temp (tempo: LONGINTEGER; prt_saída; tam_mem: INTEGER;
                      end_mem, end_resp: ADDRESS): BOOLEAN;
```

onde os parâmetros *prt_saída*, *tam_mem*, *end_mem* e *end_resp* são idênticos em significado aos seus correspondentes no serviço env_s. O parâmetro tempo especifica o período de tempo durante o qual deve ser aguardada uma mensagem de <resposta>, após o que deve-se executar o comando S2. Caso a cláusula FAIL do comando SEND não inclua um tempo de espera, indicando um tempo de espera ilimitado, o parâmetro *tempo* deve ser um número negativo.

A função *env_s_temp*, que implementa o envio síncrono temporizado, retomará o valor *TRUE* caso receba uma mensagem dentro do período especificado e o valor *FALSE*, em caso contrário. Em função do valor retomado por *env_s_temp* será executado o comando *S1* ou *S2*, assim uma possível tradução para o comando de envio síncrono de mensagem é:

```
IF env_s_temp (...)
        THEN S1
        ELSE S2
```

B.1.4 Recepção Bloqueante de uma Mensagem

O comando de recepção bloqueante de uma mensagem

```
RECEIVE <mensagem> FROM <porta de entrada>
```

é suportado pelo serviço rec, cuja interface é dada por:

```
PROCEDURE rec (end_mem: ADDRESS; prt_entrada: INTEGER);
```

onde o parâmetro *end_mem* especifica a posição, do espaço de endereçamento do módulo requisitante, onde a mensagem deve ser copiada e *prt_entrada* dá o número da porta de entrada através da qual a mensagem deve ser recebida.

B.1.5 Envio de uma Mensagem de Resposta

O comando não bloqueante de envio de uma mensagem de resposta em uma comunicação síncrona

REPLY <mensagem> TO <porta de entrada>

é suportado pelo serviço *resp*, cuja interface é dada por:

PROCEDURE resp (prt_entrada, tam_mem: INTEGER; end_mem: ADDRESS);

onde *prt_entrada* é o número da porta de entrada à qual a mensagem de resposta deve ser enviada; *tam_mem* é o tamanho, em bytes, dessa mensagem e *end_mem* é o seu endereço.

B.1.6 Desvio de uma Comunicação Síncrona

O comando de desvio de uma comunicação síncrona

FORWARD <porta de entrada> TO <porta de saída>

é suportado pelo serviço *desvia*, cuja interface é dada por:

PROCEDURE desvia (prt_saída, prt_entrada: INTEGER);

onde os parâmetros *prt_saída* e *prt_entrada* correspondem, respectivamente, aos números das portas <porta de saída>, para onde a mensagem será desviada, e <porta de entrada>, de onde a mensagem será desviada.

B.1.7 Recepção Seletiva de uma Mensagem

O suporte dos comandos PSELECT e RSELECT requer o uso dos serviços *ini_rec_sel*, *ini_temp_sel* e *sel*. A execução de um comando SELECT compreende duas fases, uma de iniciação e outra de seleção. Na primeira fase, são iniciadas as estruturas de dados do STR associadas a cada alternativa do comando SELECT. Na segunda fase, escolhe-se a seqüência de comandos associada a uma das alternativas para execução. Os serviços *ini_rec_sel* e *ini_temp_sel* são empregados na fase de iniciação e são comuns aos comandos PSELECT e RSELECT. O serviço *SEL* é usado durante a fase de seleção dos comandos PSELECT e RSELECT.

Para cada valor da variável de controle do FOR, de uma alternativa de um comando SELECT, que contém um RECEIVE, como por exemplo:

```
PSELECT
   .
   .
   .
      OR FOR <variável de controle> :=...TO...
            RECEIVE <mensagem> FROM <porta de entrada>
            ⇒ S1
END
```

é gerada uma chamada ao *serviço ini_rec_sel, cuja interface é dada por:*

```
PROCEDURE  ini_rec_sel (índice_for, cláusula, prt_entrada: INTEGER;
                        end_mem:  ADDRESS);
```

onde *índice_for* é o valor da <variável de controle> do *FOR* para a qual foi gerada esta requisição; *cláusula* é um inteiro que identifica a recepção sendo inicializada; *prt_entrada* é o número da <porta de entrada> e *end_mem* é o endereço onde a <mensagem> deverá ser recebida.

Para cada valor da variável de controle do FOR de uma alternativa de um comando SELECT que contiver um TIMEOUT como por exemplo:

```
PSELECT
   .
   .
   .
      OR FOR <variável de controle> :=...TO...
            TIMEOUT <tempo de espera>
            ⇒ S2
END
```

é gerada uma chamada ao serviço *ini_temp_sel*, cuja interface é dada por:

```
PROCEDURE ini_temp_sel (índice_for, cláusula: INTEGER; tempo: LONGINTEGER);
```

onde os parâmetros *índice_for* e *cláusula* têm significado idêntico aos seus correspondentes no serviço *ini_rec_sel*; tempo específico ou <tempo de espera> especificado pela cláusula TIMEOUT.

Caso uma alternativa de um comando SELECT não inclua uma repetição (FOR...) será feita uma única chamada ao serviço de iniciação correspondente à cláusula contida na alternativa RECEIVE ou TIMEOUT, devendo ser passado um valor qualquer em correspondência ao parâmetro *índice_for*.

A fase de seleção de um comando SELECT corresponde à requisição do serviço sel cuja interface é dada por:

```
FUNCTION sel (tipo: CHAR; cláusula_else: INTEGER; VAR índice_for,
              prt_entrada: INTEGER): INTEGER;
```

onde *tipo* especifica o tipo de seleção desejada, podendo assumir os valores P ou R que indicam escolhas priorizadas ou randômicas, respectivamente, *cláusula_else* é um valor inteiro que identifica a cláusula ELSE do comando SELECT. Caso não haja uma cláusula ELSE, o parâmetro *cláusula_else* conterá o valor zero. A função *sel* escolhe uma cláusula do comando SELECT e retoma o valor inteiro que lhe foi associado durante a fase de inicialização. Adicionalmente, em *índice_for* e *prt_entrada* são retomados, respectivamente, os valores correspondentes à variável de controle do FOR da cláusula escolhida e ao número da porta de entrada onde foi recebida a mensagem (quando for o caso).

A diferença entre os valores *sel*('P',...) e *sel*('R',...) reside no processo de seleção da cláusula a executar. O serviço *sel*('P',...) considera como mais prioritária a primeira cláusula a ser inicializada (via *ini_rec_sel*) e como menos prioriária a última. No caso de um serviço *sel*('R',...) todas as cláusulas RECEIVE têm igual prioridade e a escolha é feita aleatoriamente.

B.1.8 Cancelamento de uma Comunicação Síncrona

O procedimento pré-declarado ABORT, que permite cancelar uma comunicação síncrona e suja interface é dada por:

```
PROCEDURE ABORT (<porta de entrada>, <motivo da falha>: INTEGER);
```

é suportado pelo serviço *resp_falha*, cuja interface é dada por:

```
PROCEDURE resp_falha (motivo, prt_entrada: INTEGER);
```

onde os parâmetros *motivo* e *prt_entrada* correspondem, respectivamente, ao <motivo da falha> e ao número da <porta de entrada>. O valor inteiro *motivo* deve ser maior que a constante pré-declarada *erro_max*.

B.1.9 Determinação da Razão de uma Falha

A função pré-declarada REASON que permite saber o motivo pelo qual uma troca síncrona de mensagens falhou (foi cancelada) e cuja interface é dada por:

```
PROCEDURE REASON: INTEGER;
```

é suportada pelo serviço *motivo_falha* cuja interface é dada por:

```
FUNCTION motivo_falha: INTEGER;
```

O valor retornado por *motivo_falha*, corresponderá às constantes pré-declaradas *etimeout* ou *elink* ou a um valor indicado pela execução do procedimento ABORT no módulo que enviou a resposta à comunicação síncrona. O código *etimeout* indica que a cláusula FAIL, foi executada porque o tempo de espera por uma resposta expirou. O código *elink* indica que se tentou estabelecer uma comunicação através de uma porta de saída não conectada.

B.1.10 Teste de Conexão de uma Porta de Saída

A função pré-declarada LINKED que permite saber se uma dada porta de saída está conectada a alguma porta de entrada e cuja interface é dada por:

```
FUNCTION LINKED ( < porta de saída>): BOOLEAN;
```

é suportada pelo serviço *conectada*, cuja interface é dada por:

```
FUNCTION conectada (prt_saída: INTEGER): BOOLEAN;
```

onde *prt_saída* é o número da <porta de saída> a ser testada.

B.1.11 Determinação da Quantidade de Mensagens Enfileiradas numa Porta de Entrada

A função pré-declarada QLEN, que permite saber quantas mensagens enfileiradas numa porta de entrada (bufereada ou não) e cuja interface é dada por:

```
FUNCTION QLEN (<porta de entrada>);
```

é suportada pelo serviço *mem_prt_ent*, cuja interface é dada por:

```
FUNCTION mem_prt_ent (prt_entrada: INTEGER): INTEGER;
```

onde *prt_entrada* é o número da <porta de entrada>.

B.2 MUDANÇA DA PRIORIDADE DE UM MÓDULO

O procedimento pré-declarado SETPRIORITY, que permite alterar a prioridade de um módulo e cuja interface é dada por:

```
PROCEDURE SETPRIORITY (<nível de prioridade>: PRIORITY);
```

é suportado pelo serviço *muda_prio*, cuja interface é dada por:

```
PROCEDURE muda_prio (prio: PRIORITY);
```

onde *prio* indica o novo nível de prioridade do módulo.

B.3 SERVIÇOS DE TEMPORIZAÇÃO

São descritos os serviços que suportam os procedimentos e funções pré-declaradas da LPM, TIME e DELAY .

B.3.1 Leitura do Relógio do STR

A função pré-declarada TIME, que permite saber o valor do relógio do STR em unidades de tempo e cuja interface é dada por:

```
FUNCTION TIME: LONGINTEGER;
```

é suportada pelo serviço *relógio*, cuja interface é dada por:

```
FUNCTION relógio: LONGINTEGER;
```

B.3.2 Retardamento de um Módulo

O procedimento pré-declarado DELAY, que suspende a execução do módulo requisitante por um período de tempo e cuja interface é dada por:

```
PROCEDURE DELAY (<período de tempo>: LONGINTEGER);
```

é suportado pelo serviço *retarda*, cuja interface é dada por:

```
PROCEDURE retarda (tempo: LONGINTEGER);
```

onde *tempo* especifica o número de "ticks" correspondente ao <período de tempo> durante o qual o módulo requisitante deve ser retardado. A demanda de retarda com o parâmetro *tempo* igual a zero provoca apenas um reescalonamento.

B.4 INTERRUPÇÕES

São descritos os serviços que suportam a função INTALLOC e o procedimento WAITIO, ambos pré-declarados na LPM.

B.4.1 Mapeamento de um Elemento do Vetor Físico de Interrupções

A função pré-declarada INTALLOC, através da qual o núcleo mapeia um elemento do vetor físico de interrupções num elemento do vetor lógico de interrupções, cuja interface é dada por:

FUNCTION INTALLOC (<endereço do vetor físico> ADDRESS): INTEGER;

é suportada pelo serviço *map_int*, cuja interface é dada por:

FUNCTION map_int (<end_int>: ADDRESS): INTEGER;

onde *end_int* é o endereço de memória para onde será passado o controle quando a interrupção for atendida pelo processador. A função que implementa o serviço *map_int*, retornará um número inteiro correspondente a um elemento do vetor lógico de interrupções mantido pelo STR. O módulo tratador de interrupções que requisitou o serviço deverá, sempre que quiser, esperar pela ocorrência da interrupção envolvida no mapeamento, referenciar o elemento do valor lógico retornado pelo serviço *map_int*.

B.4.2 Espera pela Ocorrência de uma Interrupção

O procedimento pré-declarado WAITIO, através do qual um módulo tratador de interrupções aguarda a sua ocorrência, cuja interface é definida por:

PROCEDURE WAITIO (<vetor lógico>: INTEGER);

é suportado pelo serviço *espera_int*, cuja interface é dada por:

PROCEDURE espera_int (int_log: INTEGER);

onde *int_log* é o número do elemento do vetor lógico de interrupções associado à interrupção que o módulo requisitante quer aguardar.

B.4.3 Verifica a Ocorrência de uma Interrupção

A função pré-declarada OCORREINT, através da qual verifica-se a ocorrência de uma interrupção lógica, cuja interface é dada por:

FUNCTION OCORREINT (<vetor lógico>:INTEGER): BOOLEAN;

é suportada pelo serviço OCORREU-INT , cuja interface é dada por:

FUNCTION ocorreu-int (int-log: INTEGER): BOOLEAN;

onde *int-log* é o número do elemento do vetor lógico de interrupções associado á interrupção que o módulo requisitante quer verificar.

B.4.4 Reinicia o Indicador de Ocorrência de uma Interrupção

O procedimento pré-declarado LIMPAINT reinicia o indicador de ocorrência de uma interrupção lógica, cuja interface é dada por:

PROCEDURE LIMPAINT (<vetor lógico>: INTEGER);

é suportado pelo serviço *limpa-int*, cuja interface é dada por:

```
PROCEDURE limpa-int (int-log: INTEGER);
```

onde *int-log* é o número do elemento do vetor lógico de interrupção associado à interrupção que o módulo requisitante quer reiniciar.

B.4.5 Libera uma Interrupção Lógica

O procedimento pré-declarado DEALOCINT libera uma interrupção lógica especificada, cuja interface é dada por:

```
PROCEDURE DEALOCINT (<vetor lógico>: INTEGER);
```

é suportado pelo serviço *dealoca-int*, cuja interface é dada por:

```
PROCEDURE dealoca-int (int-log: INTEGER);
```

onde *int-log* é o número do elemento do vetor lógico de interrupção associado à interrupção que o módulo requisitante quer liberar.

Apêndice C

Especificação Formal dos Módulos do SCR e do CSCRL

Este apêndice apresenta a especificação formal de módulos pertencentes às camadas SCR e CSCRL, através do modelo de Máquinas de Estados Finita (MEF) [50].

A MEF está sempre em um determinado estado em qualquer instante de tempo.

O conjunto finito de estados possíveis para o SCR são descritos a seguir :

- esp_buf_usuario : aguardando buffer de mensagem do usuário, enviada através da camada STRE.
- esp_cnf_tx : aguardando reconhecimento de transmissão (ACK).
- idle : estado inativo.
- esp_cnf_consulta : aguardando confirmação de consulta.
- esp_notif_falha : aguardando notificação de falha.
- esp_rsp : aguardando resposta ("SIGNAL").
- esp_wait_rsp : aguardando em WAIT por resposta.

O conjunto finito de estados possíveis para a CSCRL são descritos a seguir:

- idle: : estado inativo.
- esp_end_local : aguardando endereço local da estação.
- esp_req_serviço : aguardando requisição de transmissão de mensagens ou o pedido à consultas.
- esp_meio_livre : aguardando o meio de comunicação ficar livre para a transmissão.
- esp_cnf_conex : aguardando confirmação do pedido de conexão.
- esp_cnf_quadro : aguardando confirmação do quadro transmitido.

- testa_cntTentTx : testa a ocorrência máxima de tentativas de transmissão.
- testa_cnt_col : testa a ocorrência máxima de colisões.
- testa_cntTentconex : testa a ocorrência máxima de tentativas de conexão.
- esp_apt_UPSC : aguardando apontador ao formato padrão UPSC enviado pelo módulo RECREM.
- esp_ped_conexao e End_Est : aguardando pedido de conexão e endereço de estação.
- quadro_ok : testa se quadro recebido está correto.

Como uma MEF, o protocolo (da Camada SCR) permanece em um certo estado até que chegue alguma "entrada" (por exemplo : comandos da Camada STRE, chegada de mensagens da CSCRL através das interfaces, expiração de temporizações internas, etc.).

O protocolo responde à entrada fazendo algum processamento (modelado por uma transição da MEF) e em seguida muda para um outro estado, produzindo uma "saída", na forma de uma requisição de serviço para a CSCRL, envio de uma mensagem para a Camada do STRE, etc.

A estrutura de uma função será a seguinte :

```
nome da função
        ação 1
        ação 2
          .
          .
          .
ação n
atualiza próximo estado
```

As entradas possíveis e as funções necessárias à MEF serão ilustradas junto com cada módulo do SCR, descritos nos próximos itens.

C.1 Especificação Formal de Módulos da Camada SCR

As entidades pertencentes a cada estação, a nível da Camada SCR são simétricas (cada uma tem comportamento idêntico a outra), assim é especificada uma única máquina ressaltando-se que existem máquinas idênticas em cada estação adjacente.

C.1.1 Módulo ENVREM

C.1.1.1 Especificação Formal

O diagrama de estados para o módulo "ENVREM", as entradas e funções são ilustrados na figura C.1.

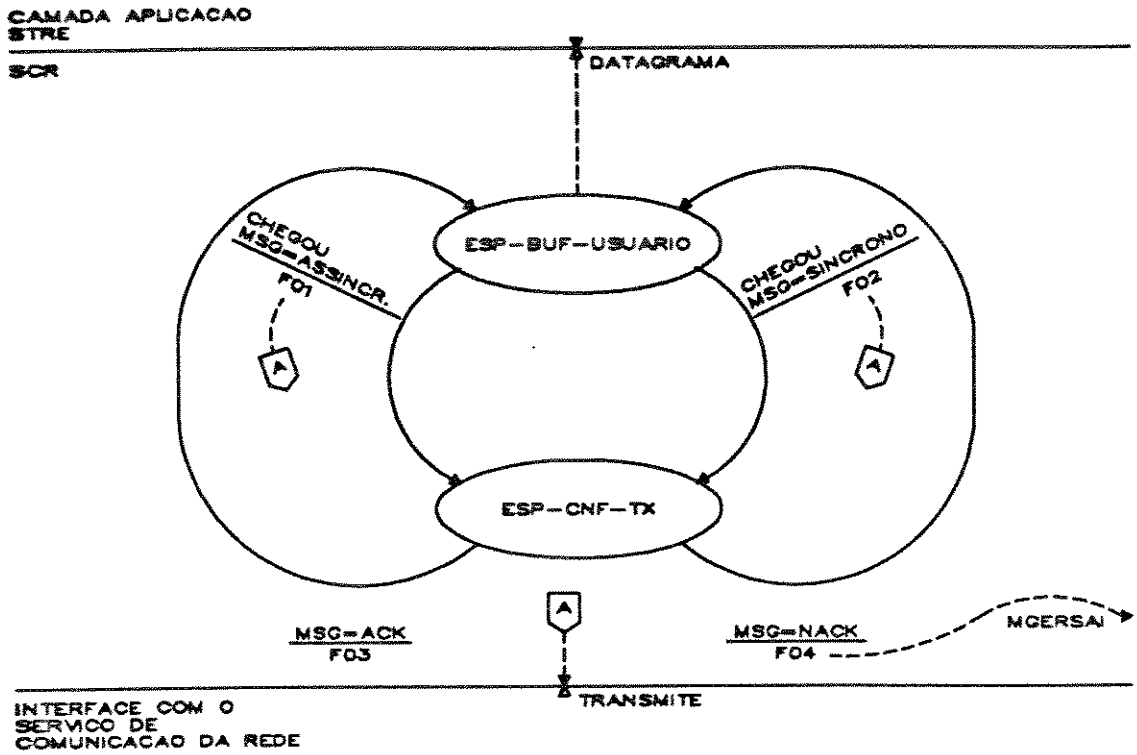

Figura C.1 : Diagrama de Estados para o Serviço Emissor (ENVREM)

C.1.1.2 Funções

A seguir, são descritas as funções que compõem o módulo ENVREM:

a) Função 01 (F01) :
   - Obtém informações para cabeçalho;
   - Libera aplicação (módulo fonte);
   - Envia UPSC formada para interface com rede;
   - Próximo estado = esp_cnf_tx.

b) Função 02 (F02) :
   - Obtém informações para cabeçalho;
   - Coloca "timeout" no UPSC;
   - Envia UPSC formada para interface com rede;
   - Próximo estado = esp_cnf_tx.

c) Função 04 (F04) :
   - Notifica falha para o módulo Gerente de Comunicação (GERCOM);
   - Próximo estado = esp_buf_usuario.

C.1.2 Módulo RECREM

a) Especificação Formal:

O diagrama de estados para o módulo "RECREM", as entradas e funções são ilustrados na figura C.2.

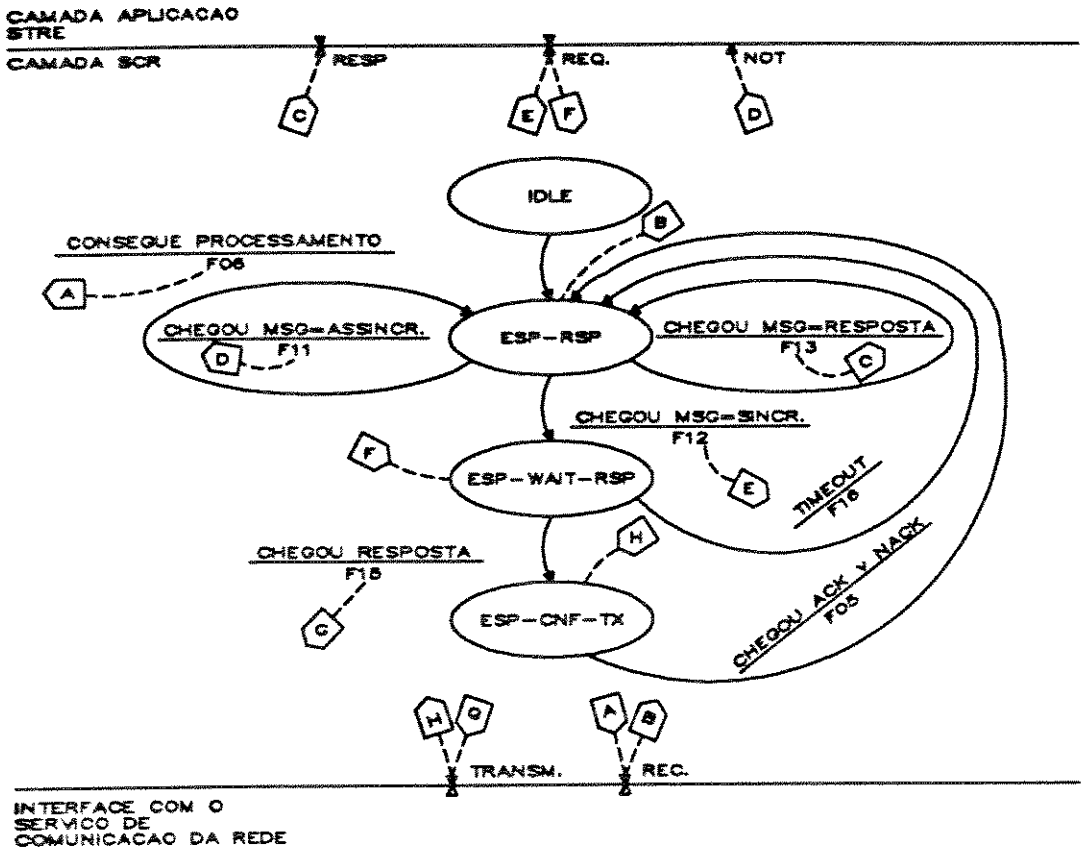

Figura C.2 : Diagrama de Estados para o Serviço RECEPTOR (RECREM)

C.1.2.1 Funções

A seguir, são descritas as funções que compõem o módulo RECREM:

a) Função 06 (F06):

- Envia apontador para buffer para o módulo de interface INTREC;
- Próximo estado = esp_rsp.

b) Função 11 (F11):

- Ligação dinâmica de sua porta "LIGASSINC" para a porta da aplicação;
- Envio assíncrono da mensagem para a aplicação;
- Próximo estado = esp_rsp.

c) Função 12 (F12):

- Ligação dinâmica de sua porta "LIGSINDIN" para a porta da aplicação;
- Envio síncrono da mensagem para a aplicação;
- Próximo estado = esp_wait_rsp.

d) Função 13 (F13):

- Ligação dinâmica de sua porta "RESPOSTA" para a porta da aplicação;
- Envia mensagem de resposta da aplicação executando uma primitiva de resposta ("REPLY");
- Próximo estado = esp_rsp.

e) Função 15 (F15):

- Obtém informações para cabeçalho da UPSC.;
- Envia apontador da UPSC para o módulo de interface com o Serviço de Comunicação da Rede.;
- Próximo estado = esp_cnf_tx.

f) Função 16 (F16):

- Obtém tempo atual (função Time);
- Exibe mensagem no vídeo;
- Próximo estado = esp-rsp.

C.1.3 Módulo GERCOM

C.1.3.1 Especificação Formal

O diagrama de estados para o módulo "GERCOM", as entradas e funções são descritas na figura C.3.

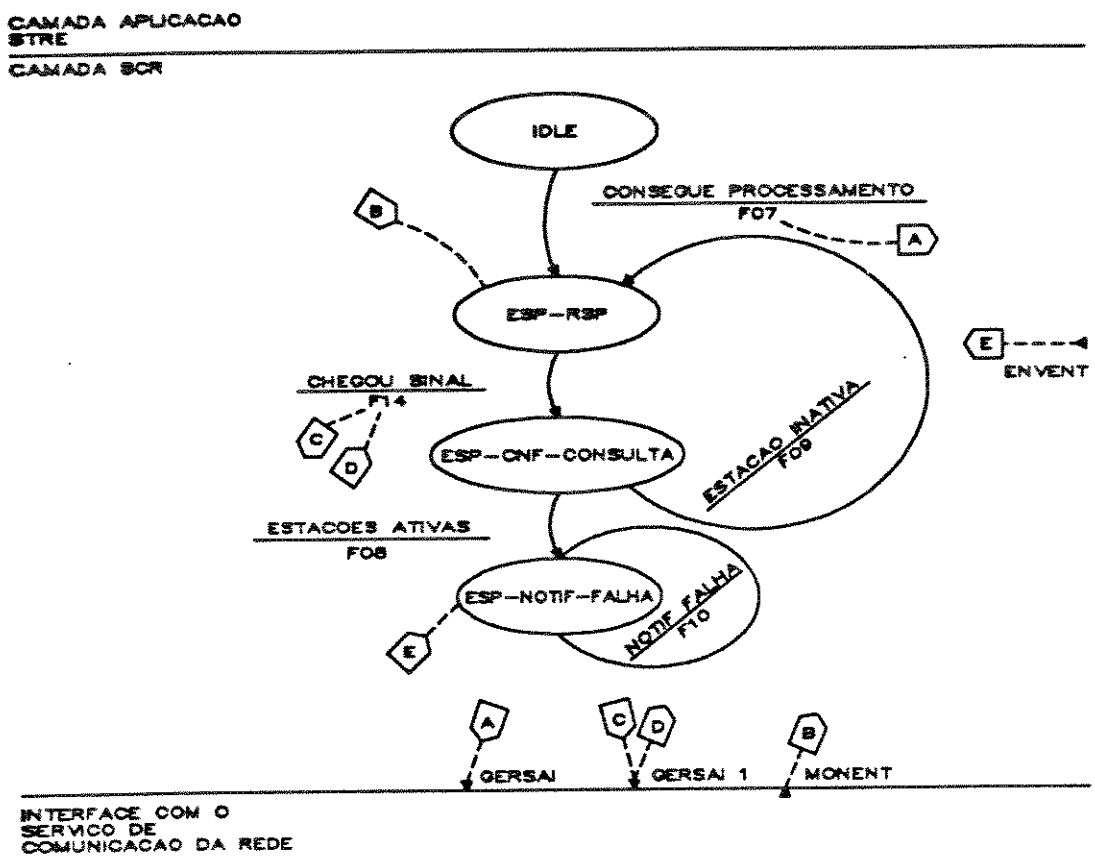

Figura C.3 : Diagrama de Estados para o Serviço GERCOM

C.1.3.2 Funções

A seguir, são descritas as funções que compõem o módulo GERCOM:

a) Função 07 (F07):

- Acessa a estrutura de dados do STRE obtendo informações de endereço local, número de estações ligadas remotamente a estação e identificador de cada estação remota;
- Envio assíncrono contendo o endereço local da estação;
- Próximo estado = esp-rsp.

b) Função 08 (F08):

- Ativação inicial da aplicação;
- Próximo estado = esp-notif-falha.

c) Função 09 (F09):

- Exibe mensagem no vídeo notificando aplicação;
- Próximo estado = esp-rsp.

d) Função 10 (F10):

- levantamento de falhas com posterior desconexão da porta ligada remotamente da aplicação.
- próximo estado =esp-notif-falha.

e) Função 14 (F14):

- iniciação do sistema local de comunicação com consultas à estações remotas.
- próximo estado = esp-cnf-consulta.

C.2 Especificação Formal de Módulos da Camada CSCRL

As entidades pertencentes a cada estação, a nível da Camada CSCRL são simétricas (cada uma tem comportamento idêntico à outra), assim é especificada uma única máquina ressaltando-se que existem máquinas idênticas em cada estação adjacente.

C.2.1 Módulo TXCCPE

C.2.1.1 Especificação Formal

O diagrama de estados para o módulo "TXCCPE", as entradas e funções são descritas na figura C.4 .

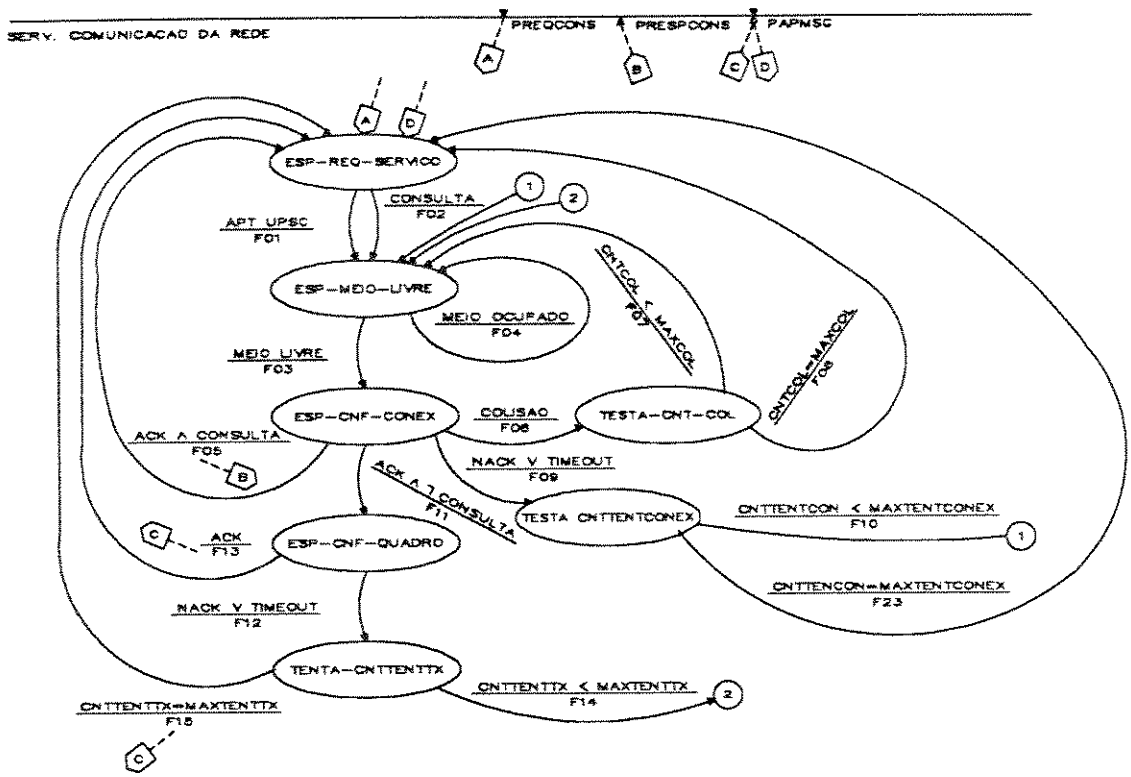

Figura C.4 : Diagrama de Estados para o Serviço Transmissor (TXCCPE)

C.2.1.2 Funções

A seguir, são descritas as funções que compõem o módulo TXCCPE:

a) Função 01 (F01):
   - Iniciação de variáveis de consulta e endereço de estação remota, denotando um pedido de transmissão de UPSC;
   - Próximo estado := Esp_Meio_Livre.

b) Função 02 (F02): . iniciação de variáveis de consulta e endereço de estação remota, denotando um pedido de consulta a estações ativas;
   - Próximo estado := Esp_Meio_Livre.

c) Função 03 (F03) :
   - Aguarda TBE;
   - Desabilita interrupções;
   - Envia pedido de conexão;
   - Próximo estado := Esp_Cnf_Conex;

d) Função 04 (F04):
   - Retardo através de comando DELAY da LPM;
   - Próximo estado := Esp_Cnf_Conex.

e) Função 05 (F05):
   - Habilita interrupção;
   - Envia resposta de sucesso a pedido de consulta a estações remotas ativas;
   - Próximo estado := Esp_Req_Serviço.

f) Função 06 (F06);
   - Especifica que houve uma colisão na variável apropriada;
   - Habilita interrupções;
   - Incrementa contador de colisões;
   - Próximo estado := Testa_Cnt_Col.

g) Função 07 (F07):
   - Retardo aleatório determinado por Backoff;
   - Próximo estado := Esp_Meio_Livre.

h) Função 08 (F08):

- envio de mensagem de resposta de fracasso devido a atingir o número máximo de colisões; próximo estado := Esp_Req_Serviço.

i) Função 09 (F09):

- habilita interrupções;
- incrementa contador de conexões;
- próximo estado := Testa CntTentConex.

k) Função 11 (F11) :

- envia quadro com tamanho de dados, dados e BCC1;
- envia identificadores origem e de quadro e BCC2;
- próximo estado := Esp_Cnf_Quadro.

l) Função 12 (F12):

- habilita interrupções de recepção;
- incrementa contador de tentativas de transmissão;
- próximo estado := Testa_CntTenTx.

m) Função 13 (F13):

- habilita interrupções;
- incrementa identificador de quadro enviado;
- envia resposta de sucesso ao módulo ENVREM do SCR;
- próximo estado := Esp_Req_Serviço.

o) Função 15 (F15):

- envio de mensagem de fracasso devido a atingir o número máximo de tentativas de transmissão ao módulo ENVREM do SCR;
- próximo estado := Esp_Req_Serviço.

p) Função 16 (F16):

- envio de uma mensagem de fracasso devido a atingir o número máximo de tentativas de conexão;
- próximo estado := Esp_Req_Serviço.

C.2.2 Módulo RXCCPE

C.2.2.1 Especificação Formal

O diagrama de estados para o módulo "RXCCPE", as entradas e funções são descritas na figura C.5.

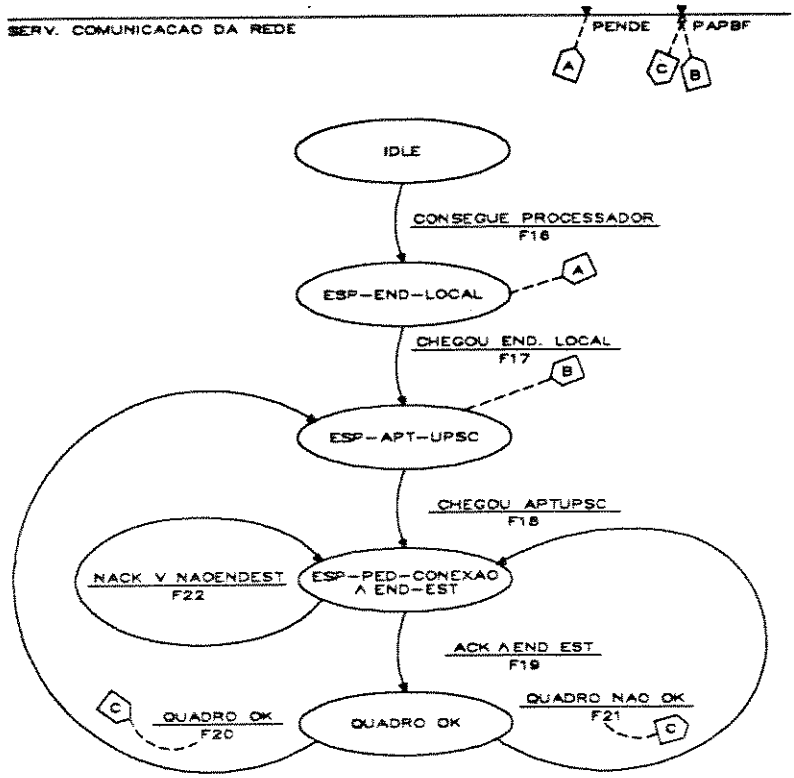

Figura C.5 : Diagrama de Estados para o Serviço de Recepção (RXCCPE)

C.2.2.2 Funções

A seguir, são descritas as funções que compõem o módulo RXCCPE:

a) Função 16 (F16):
   - iniciação da serial;
   - iniciação de variáveis;
   - próximo estado := Esp_End_Local.

b) Função 17 (F17):
   - recebe endereço local desta estação;
   - próximo estado := Esp_Apt_UPSC.

c) Função 18 (F18):

- recebe apontador a buffer disponível;
- próximo estado := Esp_Ped_Conexao AND End_Est.

d) Função 19 (F19):

- recebe quadro da mensagem;
- recebe identificadores de origem;
- recebe identificadores de quadro de mensagem;
- habilita interrupções;
- próximo estado := Quadro_OK.

e) Função 20 (F20):

- envia mensagem de sucesso para o módulo RECREM da camada SCR;
- próximo estado := Esp_Apt_UPSC.

g) Função 22 (F22):

- espera tempo relativo ao tamanho do quadro transmitido à outra estação;
- próximo estado := Esp_Ped_Conexão AND End_Est.

Apêndice D

Camada do Serviço de Comunicação da Rede Local (CSCRL)

Este apêndice apresenta uma descrição de uma rede serial simplificada para testes da versão distribuída do ambiente STER.

D.1 SERVIÇO DE COMUNICAÇÃO DA REDE

A necessidade de uma rede local para testes do ambiente STER distribuído e a não disponibilidade de uma rede no CTI/UNICAMP conduziu ao desenvolvimento de uma rede local simplificada para aplicações distribuídas, com baixo fluxo de informação entre estações e denominada Serviço de Comunicação da Rede Local.

A camada do Serviço de Comunicação da Rede Local (CSCRL) corresponde ao primeiro nível de software de uma estação integrada à rede no ambiente STER e é responsável pela comunicação de dados de forma transparente sobre a rede.

O desenvolvimento do CSCRL tem dois aspectos principais.

O primeiro aspecto corresponde a uma de suas principais funções que é a de oferecer à camada do SCR, sua camada Superior, uma interface lógica para transmissão e recepção de mensagens, independente das características do hardware que implementa a camada física. Consiste de módulos, escritos em LPM, responsáveis pela transmissão e recepção de quadros de mensagens pelo meio físico de comunicação. Para essa camada utiliza-se a nomenclatura "quadro" de mensagens.

O segundo aspecto corresponde à implantação da rede em um hardware específico e consiste de uma interface de hardware escrita em linguagem Assembly.

A Topologia em Barra foi escolhida para implementação, pela facilidade de implementação, simplicidade e baixo custo. O meio físico de transmissão é composto por um único segmento de transmissão multiponto, compartilhado pelas diversas estações interconectadas.

A figura D.1, mostra o fluxo de dados da comunicação entre estações onde estação origem e estação destino se referem, respectivamente, à estação que está enviando os dados e à estação à qual os dados são dirigidos.

O fluxo de dados ocorre quando a camada superior envia um apontador a um buffer, contendo a UPSC formada, a ser enviada via o meio de comunicação. Esse buffer é tratado como um quadro pela camada CSCRL e a este são acrescidas algumas informações, formando o novo cabeçalho correspondente à camada CSCRL.

A camada CSCRL remota, após receber o quadro contendo a mensagem, deverá retirar o cabeçalho correspondente à sua camada e enviá-lo, contendo a UPSC formada na estação origem, à sua camada superior a qual, após retirar o seu cabeçalho, se responsabilizará por entregá-la à porta do módulo destino da camada da aplicação.

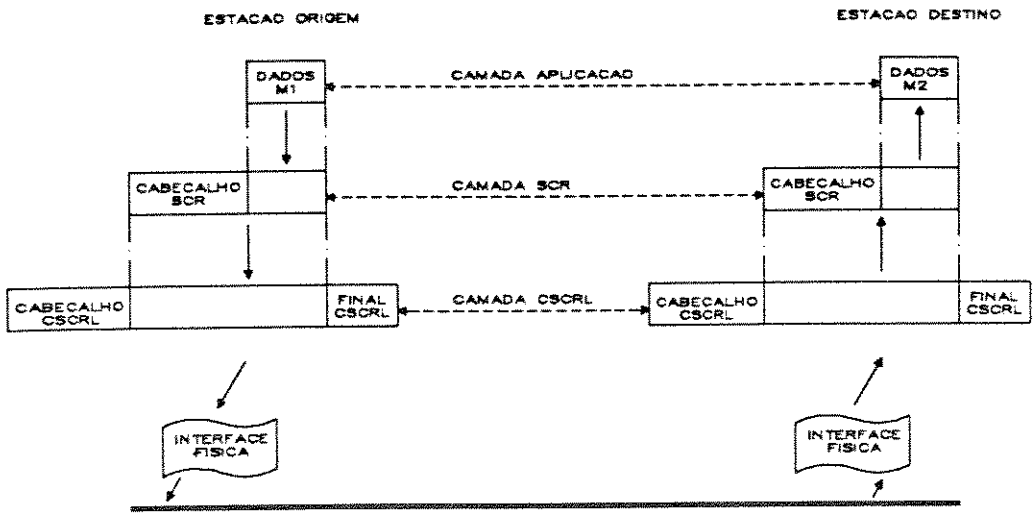

Figura D.1 : Fluxo de Informação entre Estações

D.2 Hardware de Comunicação

O sistema, no que diz respeito ao hardware de comunicação, consiste de uma interface de comunicação serial, padrão RS232-C, um adaptador para rede local em barra e um par de fios trançados formando a rede de comunicação, como mostra a figura D.2.

Os nós correspondentes às estações físicas devem ser conectados à rede de comunicação, paralelamente, fazendo com que a linha serial se comporte como um barramento comum. Quando uma estação transmite uma mensagem, todas as outras estações podem recebê-la.

A rede de comunicação é composta de dois fios: sinal de dados e terra.

O adaptador desenvolvido pela Divisão de Estruturas de Processamento em Tempo Real (ETR) do CTI, permite que microcomputadores que possuam o chip de comunicação serial UART 8250 [65] sejam conectados paralelamente à rede de comunicação serial em forma de barramento comum. O adaptador faz com que os dados que são enviados por uma estação sejam recebidos por todas as estações da rede, inclusive pela estação transmissora. O software do CSCRL detecta a existência de colisão. O adaptador é composto por um diodo e um resistor, desenvolvido com o objetivo de alcançar um baixo custo e poder ser embutido dentro do próprio conector DB-25 que, por sua vez, está conectado à interface serial dos microcomputadores, ligando-os à rede de comunicação. O circuito do adaptador, pela sua simplicidade,

não apresenta as mesmas características de imunidade a ruído da RS232-C, contudo para as condições de uso previstas (pouco ruído elétrico ambiente, distâncias pequenas e o máximo de 6 estações) o seu desempenho é satisfatório.

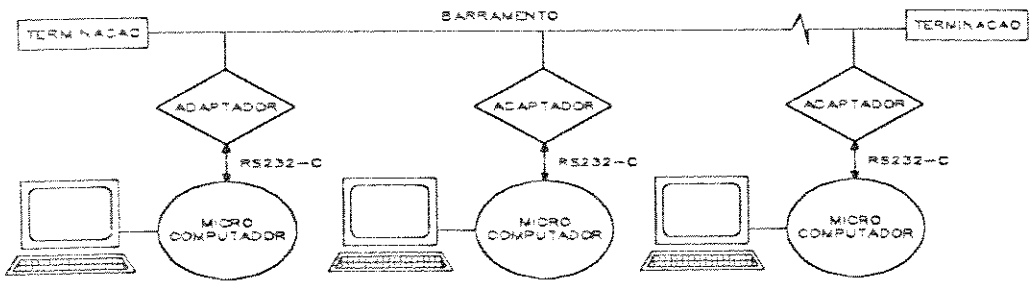

Figura D.2 : Hardware de Comunicação de Rede Local

Como o software de transmissão considera a detecção de colisão, os sinais RTS ("Request To Send" - pino 4) e CTS ("Clear To Send" - pino 5) estão conectados entre si. Da mesma maneira, os sinais DTR ("Data Terminal Ready" - pino 20) e DSR ("Data Set Ready" - pino 6) também estão conectados entre si.

O par de fios trançados pode ser o usado em telefonia, o qual é de baixo custo, inclusive podendo utilizar instalações já existentes.

Sobre essa arquitetura básica é que a Camada CSCRL foi implementada.

D.3 Protocolo de Acesso ao Meio

O meio de transmissão, no caso de barramento, é um recurso compartilhado por diversos usuários e, assim, certas regras específicas de controle de acesso são essenciais.

Para que as estações possam utilizar o barramento comum com eficiência é necessária a existência de um protocolo de acesso ao meio, uma vez que este barramento é compartilhado pelas estações.

A classe de métodos aleatórios foi escolhida para implementação, e é encontrada em aplicações de escritório e aplicações de tempo real para situações não críticas de tempo.

O método escolhido é o CSMA-CD/PR - " Acesso Múltiplo com Detecção de Portadora e Detecção de Colisão com Prioridade para Tráfego de Reconhecimento" [49].

A escolha feita deveu-se à facilidade de implementação e a tentativa de adquirir experiência com outro método de acesso para a topologia barramento, já que está em desenvolvimento no CTI/UNICAMP uma placa para redes locais com implementação do método de acesso que opera por "passagem de uma permissão de acesso" ("token-bus") [5].

D.3.1 Método CSMA-CD/PR

O método CSMA-CD/PR deriva da proposta com a rede ETHERNET (CSMA-CD), incorporando um serviço prioritário de mensagens de reconhecimento, também conhecido como "Acknowleging

ETHERNET".

As redes de comunicação, estruturadas em barramento, não permitem a implantação de um serviço de resposta (reconhecimento) imediata acompanhando a recepção de uma mensagem. Na técnica CSMA-CD as mensagens de controle, por exemplo, mensagens de reconhecimento, são tratadas, do ponto de vista de transmissão, como uma mensagem qualquer. Um tempo de resposta relativamente longo, além de comprometer a eficiência na utilização do meio, exige uma maior capacidade (sobrecarga) das estações responsáveis, no caso, pela gerência de controle do tráfego de reconhecimento.

O protocolo CSMA-CD/PR foi desenvolvido com o propósito de superar os inconvenientes acima citados, providenciando um serviço prioritário de "mensagens de reconhecimento" implementado na própria estação de comunicação.

Esse protocolo tem como características básicas, "ouvir antes de falar" e tambem "ouvir enquanto fala".

Na primeira etapa do protocolo, antes de iniciar a transmissão, uma estação poderá tentar transmitir uma mensagem num certo instante e para tal escuta o canal de transmissão ("ouvir antes de falar"), para verificar se não há ninguém usando o canal, tomando posse do barramento, quando livre. Esta estação aguarda, então, um Tempo Básico de Espera (TBE) e, após este, escuta novamente o canal de transmissão. Se após o TBE, o canal estiver livre, aí sim, a estação inicia a transmissão da sua mensagem. Após o início da transmissão, a estação tem condições de detectar as colisões, porque ao transmitir também está recebendo o que está no canal, podendo assim verificar se o que está na linha é o mesmo dado que foi transmitido.

A colisão ocorre porque, em virtude da velocidade limitada de transmissão e das distâncias entre estações, uma outra estação poderá iniciar a transmissão logo após, sobrepondo duas transmissões e ocasionando uma colisão da informação. No caso de conflito, para evitar uma transmissão desnecessária, cada estação pertencente ao barramento escuta durante a transmissão ("ouvir enquanto fala") e uma vez detectada a colisão, a transmissão é interrompida e existe um algoritmo simples, (algoritmo de Backoff) para o tratamento da colisão e reinício da transmissão, quando então uma nova tentativa é realizada.

D.3.2 Tratamento de Colisões

Na ocorrência de uma colisão (imediatamente detectada), quando da tentativa de transmissão de mensagens, as estações envolvidas chamam a rotina de Backoff. Nessa rotina, o controle de retransmissão é determinado por um controle randômico denominado "truncated binary exponential backoff". Depois da colisão ocorrer, o algoritmo força um atraso antes da nova retransmissão do quadro. O número inteiro r de intervalos é escolhido no intervalor $0 <= r < 2^k$, sendo que k=mínimo (n,10). Após 16 tentativas de transmissão sem sucesso (n=16), o módulo de transmissão indica à camada superior uma situação de erro. Em caso contrário, a estação repetirá o procedimento de tentativa de transmissão descrito no item anterior, após aguardar um intervalo de tempo aleatório, determinado pelo algoritmo de distribuição dos atrasos de retransmissão (Backoff).

D.3.3 Estrutura de Dados do Protocolo

As mensagens que fluem de uma estação à outra estação remota passam pelo CSCRL, o qual realiza a operação desejada (consulta se estações ativas, transmissão de mensagens, recepção de mensagens). Para isso, é utilizada uma estrutura de dados bem definida onde as mensagens são armazenadas.

A sincronização a nível de bit é realizada pela camada física em um modo de transmissão assíncrono

(não exigindo a fixação prévia de um padrão de tempo, entre o transmissor e o receptor). Essa noção de assincronicidade, dada pela transmissão de caracteres tipo "start-stop", pode ainda ser estendida para a transmissão de quadros de dados. Basta considerar os quadros como caracteres com um número maior de bits, operando então a nível de quadros de mensagens de dados e de controle.

O protocolo utilizado usa o método de paridade longitudinal ("Longitudinal Redundancy Check") [49], que consiste em acrescentar ao quadro da mensagem um caractere BCC ("Block Character Check"), que represente uma operação lógica sobre os bits dos diversos caracteres que compõem o quadro da mensagem. Esse método é facilmente implementado por software, uma vez que para calcular o BCC basta fazer o XOR da informação enviada, sem incluir a si mesmo, para detecção e recuperação de erros.

Na camada CSCRL, o cabeçalho do quadro da mensagem é composto por 3 bytes e o final do quadro da mensagem é composto por 4 bytes.

A seguir, são descritos os formatos dos quadros das mensagens de dados e de controle utilizados pela CSCRL e a função de cada campo dentro das mensagens.

D.3.3.1 Mensagens de Dados

O quadro de mensagem de dados transporta dados da aplicação através do meio de comunicação.

O seu formato é:

SOH	ENQ	ENDR	TAMD	DADOS	BCC1	ORG1	ORG2	IDQ1	IDQ2	BCC2

onde:

SOH : Identificador de início de cabeçalho.
Tamanho do campo = 1Byte.

ENQ : Identificador de cabeçalho.
Tamanho do campo = 1Byte.

ENDR : Especifica o endereço da estação remota a quem é endereçada essa mensagem.
Tamanho do campo = 1Byte.

TAMD : Especifica o número de bytes do campo de dados.
Tamanho do campo = 1Byte.

DADOS : Campo de dados, totalmente transparente ao protocolo. A única restrição é que deve ter o número de bytes especificado em TAMD.

BCC1 : Bloco de verificação de erro de transmissão do quadro contendo o cabeçalho, tamanho dos dados e dados da mensagem usando o método de paridade longitudinal.
Tamanho do campo = 1Byte.

ORG1, ORG2 : Identificador do endereço da estação origem. Tamanho do campo = 1Byte.

IDQ1, IDQ2 : Identificador contendo o número do quadro enviado.
Tamanho do campo = 1Byte.

BCC2 : Bloco de verificação de erro de transmissão do quadro contendo os identificadores de estação origem e número do quadro enviado.
Tamanho do campo = 1Byte.

D.3.3.2 Mensagens de Controle

O quadro de mensagens de controle transporta informações para o controle do meio de comunicação físico, para consulta a estações remotas durante o processo de iniciação do sistema distribuído, para consulta a estações remotas quando requisitado pelo módulo GERCOM do SCR e para notificação de aceitação de um pedido de conexão. Tem um comprimento fixo de bytes (mesmo comprimento e mesmos campos do quadro de cabeçalho de mensagens).

O seu formato é:

SOH	ENQ	ENDR

onde :

SOH : Identificador de início de cabeçalho.
Tamanho do campo = 1Byte.

ENQ : Identificador de cabeçalho.
Tamanho do campo = 1Byte.

ENDR : Especifica o endereço da estação remota a quem é endereçada a mensagem.

Tamanho do campo = 1Byte.

D.3.3.3 Mensagens de Reconhecimento

a) Mensagem de Reconhecimento Positivo (ACK)

A mensagem ACK fornece à estação transmissora uma resposta de reconhecimento positivo. É usada para notificar o transmissor da aceitação de um pedido de estabelecimento de conexão por outra estação; da resposta a uma consulta, se a estação está ativa, e da notificação de recebimento correto de uma mensagem.

A mensagem ACK consiste de um único caractere enviado pelo módulo de recepção de mensagens.

b) Mensagem de Reconhecimento Negativo (NACK)

A mensagem NACK fornece à estação transmissora uma notificação da existência de alguma condição de erro detectada pela estação destino. É usada para notificar a não aceitação de um pedido de conexão; de resposta a uma consulta de estação inativa e de notificação do recebimento incorreto de uma mensagem com erro ou duplicada.

A mensagem NACK consiste de um único caractere enviado pelo módulo de recepção de mensagens ou reconhecida, como tal, em situações tais como "timeout".

D.3.4 Descrição dos Serviços Oferecidos pelo CSCRL

A camada CSCRL fornece os serviços básicos à camada do SCR, sua camada superior, através de

dois módulos escritos em LPM e responsáveis pela transmissão e recepção de quadros de mensagens enviados pela (ou para a) camada SCR.

O módulo de transmissão (TXCCPE) é responsável pela transmissão da mensagem enviada pela camada SCR para a estação remota através do meio de comunicação, isenta de erros de transmissão.

O módulo de recepção (RXCCPE) é responsável pela recepção da mensagem que chega do meio de comunicação e, se correta, pelo envio dessa mensagem para a camada SCR.

Os módulos citados acima são apresentados nos próximos itens de duas maneiras: uma descrição informal através do uso da linguagem natural para uma melhor compreensão de sua operação e uma especificação formal mais concisa e precisa, ausente de ambigüidades. A especificação formal dos módulos é descrita no APÊNDICE C. O modelo da Máquina de Estados Finita (MEF) foi escolhido para ilustração dos modelos de transição.

Esses módulos, decritos a seguir, utilizam-se de rotinas escritas em linguagem Assembly que programam os níveis mais baixos de interface com o hardware, tais como:

- Iniciação da serial;
- Transmissão ou recepção de caracteres;
- Transmissão ou recepção de quadros de mensagens;
- Habilitação ou desabilitação de interrupções da serial;
- Verificação de meio ocupado.

D.3.4.1 Módulo de Transmissão (TXCCPE):

O módulo de transmissão (TXCCPE) oferece os serviços de transmissão de mensagens e de consulta a estações ativas na rede.

O módulo TXCCPE é um módulo privilegiado, escrito em LPM, contendo três portas de comunicação, conforme ilustra a figura D.3.

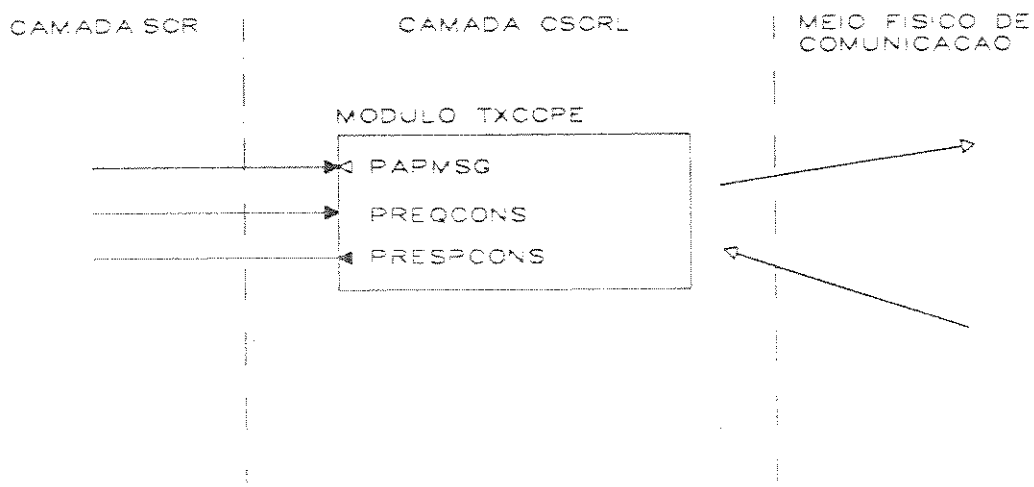

Figura D.3 : Portas de Comunicação de Interface do Modulo TXCCPE

A sua interface, descrita em LPM, é a seguinte:

```
MODULE TXCCPE;

    USE undefrs2.inc;
    ENTRYPORT
        PApMsg : aptUPSC REPLY tiposignal;
        PReqCons : INTEGER;
    EXITPORT
        PRespCons : tiposignal;

END-MODULE.
```

A sua porta de entrada "PApMsg" é uma porta síncrona, utilizada por TXCCPE para esperar por apontador de UPSC enviado pela Camada SCR.

A sua porta de entrada "PReqCons" é uma porta assíncrona, utilizada por TXCCPE para esperar por requisições de consulta de estações ativas na rede.

A sua porta de saída "PRespCons" é uma porta assíncrona, utilizada por TXCCPE para enviar respostas de sucesso ou fracasso às consultas requisitadas pelo módulo Gerente de Comunicação.

ESPECIFICAÇÃO INFORMAL :

Inicialmente, TXCCPE espera por um pedido de serviço de consulta ou espera por um apontador de UPSC a ser transmitido.

O módulo TXCCPE, desejando transmitir uma mensagem, aguarda primeiro o meio ficar livre para, então, aguardar novamente por um intervalo de tempo denominado Tempo Básico de Espera(TBE), que é superior ao dobro do tempo máximo de propagação no barramento. Se, após o intervalo TBE, o meio ainda continuar livre, aí sim, o nó de comunicação transmitirá sua mensagem, onde enviará um quadro contendo o cabeçalho e significando um pedido de conexão que consiste em enviar :

- Um caractere de início de transmissão (SOH);
- Um caractere ENQ;
- O endereço físico da estação remota.

Para realizar o envio do caractere, o módulo TXCCPE faz a verificação do estado da linha de transmissão através da leitura do estado da interface serial de comunicação. Enquanto o canal de transmissão estiver no estado ocupado e o tempo ("timeout") que o módulo tiver para tentar a transmissão não se esgotar, irá permanecer verificando o estado da linha. Quando este tempo se esgotar, o módulo retorna um código de transmissão não realizada.

Nessa etapa são detectadas as colisões. No caso de ocorrer uma colisão (imediatamente detectada e tratada da forma descrita anteriormente), o módulo TXCCPE repetirá o procedimento de transmissão inicial, após aguardar um intervalo de tempo aleatório determinado por um algoritmo de distribuição dos atrasos de retransmissão ("BACKOFF") depois de um certo número máximo de tentativas de estabelecimento de conexão.

Se não receber a confirmação (NACK), o módulo TXCCPE repetirá o procedimento de transmissão inicial, esperando o meio ficar livre. Se não conseguir transmitir o pedido de conexão, depois de um certo

número máximo de tentativas de conexão, envia uma mensagem "signal" de fracasso para o módulo GERCOM do SCR.

No caso de receber uma confirmação de conexão (ACK), isto é, após ter transmitido a parte referente ao cabeçalho da mensagem, o módulo TXCCPE realizará a transferência dos dados a serem transmitidos. O número de bytes a serem transmitidos é indicado no campo TAMD. O remetente envia um quadro de mensagem contendo o tamanho dos dados da mensagem e os dados propriamente ditos e também BCC1 e o final do quadro da mensagem com BCC2. Caso ocorra um byte de dados com código igual aos bytes de controle DLE ou ENQ, os mesmos serão precedidos por um DLE.

Durante a transmissão dos dados, qualquer impossibilidade de transmissão (estação inativa, detecção de erros através do BCC, etc.) implicará na tentativa de nova transmissão do quadro, dentro de um número máximo de tentativas de transmissão. Caso receba a confirmação de quadro, envia uma mensagem ("signal") de sucesso para o módulo ENVREM do SCR.

Após a transmissão do quadro, o módulo espera pela confirmação de recebimento (ACK ou NACK), que é enviada pela camada CSCRL remota. A transmissão terá sucesso quando todos os bytes de dados forem transmitidos corretamente.

Caso receba a confirmação de quadro (ACK), envia uma mensagem "signal" de sucesso para o módulo ENVREM do SCR.

Se não conseguir transmitir a mensagem, depois de um certo número máximo de tentativas de transmissão, envia uma mensagem ("signal") de fracasso para o módulo ENVREM do SCR.

D.3.4.2 Módulo de Recepção (RXCCPE)

O módulo de recepção (RXCCPE) oferece o serviço de recepção de mensagens do meio de comunicação.

O módulo RXCCPE é um módulo de sistema, escrito em LPM, contendo duas portas de comunicação, conforme ilustra a figura D.4.

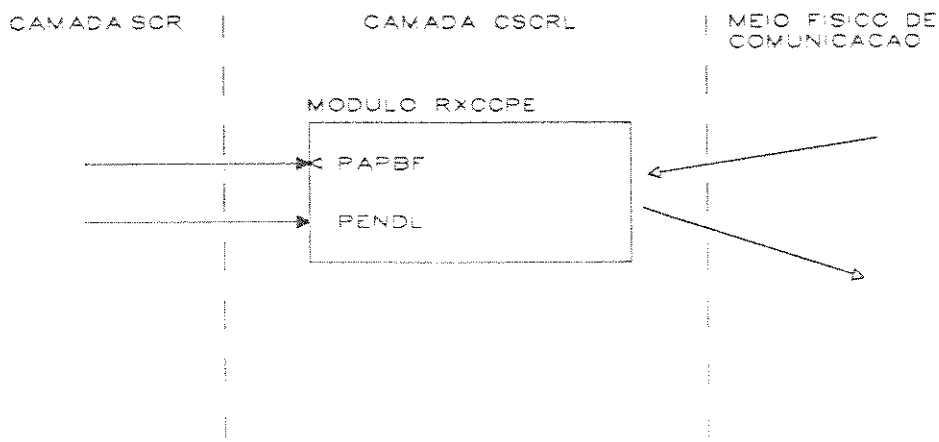

Figura D.4 : Portas de Comunicação da Interface do Módulo RXCCPE

A sua interface, em LPM, é a seguinte :

```
MODULE RXCCPE;

    USE undefrs2.inc;
    ENTRYPORT
        PApBf : aptUPSC REPLY tiposignal;
        PEndL  : BYTE;

END_MODULE.
```

A sua porta de entrada "PEndL" é uma porta assíncrona, utilizada por RXCCPE para esperar pelo endereço local dessa estação, transmitido pelo módulo Gerente de Comunicação da estação (GERCOM) da camada SCR.

A sua porta de entrada "PApBf" é uma porta síncrona, utilizada por RXCCPE para esperar por um apontador a um buffer ativo de recepção, enviado pelo módulo RECREM da camada SCR.

ESPECIFICAÇÃO INFORMAL:

Inicialmente, RXCCPE espera pelo endereço local dessa estação em sua porta de entrada "PEndL". Este, ao ser enviado por GERCOM, termina a fase de iniciação das estações, ativando o módulo de recepção, o qual repete os passos abaixo.

Espera por um apontador de buffer ativo disponível pertencente ao módulo RECREM da camada SCR.

Ao chegar, fica esperando por um pedido de conexão requisitado por qualquer estação pertencente ao meio de comunicação.

O primeiro caractere chega via interrupção. Após este, os próximos caracteres que chegam através da rede de comunicação são obtidos por "pooling". Para realizar a recepção de caracteres, o módulo RXCCPE faz a verificação do estado da linha de transmissão. Se estiver livre, tenta a recepção do próximo byte do quadro de mensagem. Enquanto não receber nenhum caractere e o tempo "timeout" não se esgotar, o módulo espera pela recepção. Quando o tempo se esgotar, o módulo retorna um código de recepção não realizada. Verifica os caracteres de controle (DLE) e de sincronismo (SOH,ENQ) recebidos, descartando os caracteres de controle inseridos pelo módulo de transmissão.

Verifica os caracteres recebidos de sincronismo, e quando ocorre o recebimento do caractere "SOH", o módulo inicia a recepção do cabeçalho do quadro, que é composto de 3 bytes. Do cabeçalho é extraída a informação referente ao endereço destino para o qual o quadro é enviado; sendo verificado se o quadro é endereçado a essa estação receptora. Sendo o endereço destino recebido no quadro, o mesmo que o endereço local da estação, envia a confirmação de conexão ("Acknowledge") para a estação transmissora. Recebe, também, os dois endereços do remetente e os dois identificadores de mensagem.

Caso o identificador de mensagem recebida corresponda a uma mensagem já aceita pela estação, e, portanto, significando a chegada de uma mensagem duplicada, o módulo trata-a como uma mensagem com erro e retorna para esperar novo pedido de conexão.

Caso a mensagem seja aceita com sucesso, envia a mensagem ("signal") de sucesso para o módulo RECREM da camada do SCR.

Nas rotinas escritas em linguagem Assembly são detectados os erros possíveis durante a transmissão, lendo-se o registrador de status da interface serial de comunicação.

D.4 Configuração do CSCRL

A figura D.5. apresenta o arquivo RS2NET.LCM, contendo a LCM a nível STATION, onde são descritas as conexões entre as portas das instâncias dos módulos de comunicação CSCRL com as portas dos módulos da camada SCR. O arquivo RS2NET.LCM deverá estar na área de trabalho do usuário que utilize a diretiva de rede correspondente na especificação NETWORK.

A configuração do CSCRL não tem os módulos de interface, isto porque, como a rede de comunicação serial foi desenvolvida no CTI, a interface está embutida nos módulos do CSCRL. No caso de haver uma rede local comercial nessa camada, as conexões seriam feitas entre o CSCRL e os módulos de interface.

```
STATION EST;

    INSTANCE
        envr : ENVREM;
        gcom : GERCOM;
        rcpt : RXCCPE;
        tmt  : TXCCPE;
        rec1,rec2,rec3,rec4 : RECREM;

    CREATE
        envr /P=SYSTEMLOWPR,
        gcom /P=SYSTEMLOWPR,
        rcpt /P=SYSTEMHIGHPR /S=64 /H=2,      tmt /P=SYSTEMLOWPR,
        rec1 /P=SYSTEMLOWPR,     rec2 /P=SYSTEMLOWPR,
        rec3 /P=SYSTEMLOWPR,     rec4 /P=SYSTEMLOWPR,

    LINK
        envr.transm      TO   tmt.PApMsg;
        envr.mgersai     TO   gcom.envent;
        gcom.gersai      TO   rcpt.PEndL;
        gcom.gersai1     TO   tmt.PReqCons;
        tmt.PRespCons    TO   gcom.monent;
        rec1.rec         TO   rcpt.PApBf;
        rec1.transm      TO   tmt.PApMsg;
        rec2.rec         TO   rcpt.PApBf;      rec2.transm   TO   tmt.PApMsg;
        rec3.rec         TO   rcpt.PApBf;      rec3.transm   TO   tmt.PApMsg;
        rec4.rec         TO   rcpt.PApBf;      rec4.transm   TO   tmt.PApMsg

END_STATION
```

Figura D.5 : Configuração Nível STATION do arquivo RS2NET.LCM

BIBLIOGRAFIA

[1] Adán Coello, J.M., "Suporte de Tempo Real para um Ambiente de Programação Concorrente ". Dissertação de Mestrado em Engenharia Elétrica. Campinas, UNICAMP, 1986.

[2] Amorim, C.L., Barbosa, V.C. e Fernandes, E.S.T., "Uma Introdução à Computação Paralela e Distribuída". VI Escola de Computação. Campinas, 1988.

[3] Andrews, G.R., "The Distributed Programming Language SR - Mechanisms, Design and Implementation". Software, Practice and Experience. 12(8), 1982.

[4] Andrews, G.R., Olsson, R.A., "The Evolution of the SR Language". Distributed Computing. 1(3), 1986.

[5] Appezato, L., "Um Processador de Comunicação de Rede Local para Controle de Processos". V Simp. Bras. de Redes de Computadores. São Paulo, Abril, 1987.

[6] Atkinson, C., Moreton, T., Natali, A., "Ada for Distributed Systems". Ada Companion Series. Cambridge University Press, 1988.

[7] Barak,A., Litman,A., "MOS: A Multicomputer Distributed Operating Systems". Software-Practice and Experience, 15(8), p.725-737, Aug., 1985.

[8] Black, A., Hutchison, N., Jul, E., Levy, H., Carter, L., "Distribution and Abstract Types inEmerald". IEEE Trans.on Software Eng. SE-13(1), jan., 1987.

[9] Braden, R.T., "Requirements for Internal Host-Communication Layers". RFC 1122, October, 1989.

[10] Bressan, G., "Ambiente de Programação Distribuída: Definição, Análise e Síntese". Dissertação de Doutorado, USP, 1985.

[11] Brinch Hansen, P., "The Architecture of Concurrent Programs". Englewood Cliffs, N.J., Prentice-Hall, Inf., p.317, 1977.

[12] Burns, A., Lister, A.M., Wellings, A.J., "A Review of Ada Tasking". In Lecture Notes in Computer Science - Berlim . Springer-Verlag, 1987.

[13] Burns, A., "Programming in OCCAM-2.". Wokingham, Addison Wesley, 1988.

[14] Burns, A., Wellings, A., "Real-Time Systems and Their Programming Languages". Addison-Wesley Publishing Company, 1989.

[15] Cardozo, E., "DPSK - A Kernel for Distributed Problem Solving". Dissertation for the degree Doctor in Philosophy in Electrical and Computer Engineering. Carnegie Mellon University, 1987.

[16] Cook, R.P., "MOD - A Language for Distributed Programming". In Proceedings of the 1st International Conference on Distributed Computing Systems. Huntsville, Alabama, 1979.

[17] Cornhill, D., "A Survivable Distributed Computing System for Embedded Application Programs written in ADA". Ada Letters, 3(3), 1983.

[18] Daniels,H.A., Mayur,N., "More Than Mainframes". IEEE Spectrum, Aug. 1985.

[19] Danthine, A.S., "Protocol Representation with Finite State Models". IEEE Trans. Comm., V.Com - 28, no. 4, p. 632 - 43, Abr, 1980.

[20] Dias, M., "Petri Net Based Models in the Specification and Verification of Protocols". Petri Nets Applications and Relationship to other Models of Concurring. Lecture Notes in Computer Science, Springer Verlag, 1986.

[21] DTIA 001/88, "STER - Um Ambiente para o Desenvolvimento de Software Tempo Real". Relatório Técnico, CTI/IA, 1988.

[22] Dulay, N., Kramer J., Magee, J., Sloman M., Twidlek, "The CONIC Configuration Language ". Version 2.3. Doc 84/20. Department of Computing, Imperial College of Science & Technology, UK, 1985.

[23] Dutra, L.S.V., Catto, A.J., "MIMD, Nível Triplo Aplicativo, ... ? Como Classificar uma Arquitetura". VI Cong. Soc. Bras. de Computação (XIII SEMISH). Recife - Olinda, julho 1986.

[24] Enslow, P.H., "What is a Distributed Data Processing System". IEEE Computer, jan. 1978.

[25] Glass, R.L., "Real-Time Software". Prentice-Hall, Inc. New Jersey, 1983.

[26] Guimarães, E.G., Lopes, A.B., Adán Coello, J.M., Magalhães, M.F., "A Distribuição do Ambiente para Desenvolvimento de Software de Tempo Real - STER". Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos. Florianópolis, SC., 1989.

[27] Herlihy, M. e Liskov B., "A Value Transmission Method for Abstract Data Types". ACM Toplas, 4(4), 1982.

[28] Hexel, R.A., "Núcleo Multiprocessado para Aplicações em Tempo Real", Dissertação de Mestrado em Ciência da Computação. IMECC - UNICAMP, Agosto de 1988.

[29] IBM-PC Pascal Compiler Language Reference, Version 2.0. Personal Computer Languages Series, IBM. Corp., 1984.

[30] IBM-PC Pascal Compiler Fundamentals, Version 2.0. Personal Computer Languages Series, IBM. Corp., 1984.

[31] ISO 7185, "Specification for Computer Programming Language PASCAL". International Standards Organization, 1983.

[32] Jha, R., Eisenhauer, G., Kamrad II J.M, Cornhill, D., "An Implementation Supporting Distributed Execution of Partitioned Ada Programs". Ada Letters, 9(1), 1989.

[33] Kamada, A., Magalhães, M.F., "Um Servidor de Arquivos para Programação Tempo Real". XX Congresso Nacional de Informática, 1987.

[34] Kurmer, C., Mendes, S.B.T., "Sistemas Operacionais Distribuídos". Editora CampusLtda, 1988.

[35] Kramer, J., Magee, J., Sloman, M. S. e Lister, A.M., "CONIC - An Integrated Approach to Distributed Computer Control Systems". IEEE Proceedings (Part E), 1983.

[36] Kramer, J., Magee, J., Sloman M., Twidle K., Duly, N., "The CONIC Programming Language". DOC 84/19, Department of Computing. Imperial College of Science and Technology, UK, 1984.

[37] Lages, N.A.C. , Nogueira, J.M.S. ,"Introdução aos Sistemas Distribuídos". Ed. Unicamp, EBAI - 1986.

[38] Lampson, B.W., Paul, M., Siegert, H.J., "Distributed Systems - Architecture and Implementation". Springer-Verlag, Berlim, 981.

[39] Lauer, H., Needham R., "On the duality of Operating System Structure". In Proceedings of the Second International Symposium on Operating System Principles". pg. 3-19, IRIA.

[40] Liskov, B., Scheifler, R., "Guardians and Actions: Linguistic Support for Robust, Distributed Programs". ACM Transactions on Programming Languages and Systems. 5(3), 1983.

[41] Liskov, B., "The ARGUS Language and System". In Distributed Systems Methods and Tools for Specification An Advanced Course. Springer-Verlag , Berling, 1985.

[42] Lopes, A.B., Adán Coello, J.M., " Um Ambiente para o Projeto e Implementação de Software para Sistemas Distribuídos de Controle em Tempo Real". Anais do 2o. Congresso Nacional de Automação Industrial (CONAI). São Paulo, 1985, p. 467-75.

[43] Lopes, A.B., "LPM e LCM: Linguagens para Programção e Configuração de Aplicações de Tempo Real". Dissertação de Mestrado, Curso de Mestrado em Sistemas e Computação, Centro de Ciências e Tecnologia. Universidade Federal da Paraíba, Ago, 1986.

[44] Lopes, A.B., Adán Coello, J.M., Magalhães, M.F., "Um Ambiente para o Desenvolvimento de Software de Tempo Real e sua Implementação". VII Cong. Soc.Bras. de Computação (XIV SEMISH). Salvador, julho, 1987.

[45] Magalhães, M.F., "Software para Tempo Real". Editora da UNICAMP, Campinas, 1986.

[46] Magee, J.N., "Provision of Flexibility in Distributed Systems". P.H.D.Thesis, Department of Computing. Imperial College of Science S. Technology. London, apr. 1984.

[47] Mendes, M.J., "A Pesquisa em Sistemas Distribuídos no Brasil". Seminário Franco-Brasileiro em Sistemas Informáticos Distrib.".Florianópolis, SC , Brasil, 1989.

[48] Merlin, P.M. Faber, D.J., "Recoverability of Communication Protocols Implications of a Theory Study". IEEE Trans. Comm. V.Com 24, p. 1036 - 43, set. 1976.

[49] Moura, J.A. Sauvé, J.P., Araújo, J.F., Giozza, W.F., "Redes Locais de Computadores - Tecnologia e Aplicações". McGraw -Hill, 1986.

[50] Moura, J.A. Sauvé, J.P., Araújo, J.F., Giozza, W.F., "Redes Locais de Computadores - Protocolos de Alto Nível e Avaliação de Desempenho". McGraw- Hill, 1986.

[51] Parnas, D.L., "On the Criteria to be Used in Decomposing Systems into Modules" Communications of the ACM 15(12), Dec. 1972.

[52] Pressman, R.S., " Software Engineering". MacGraw-Hill, Inc. 1982.

[53] Rector, R. Alexy, G., "The 8086 Book". Berkeley, CA, Osborne, 1980.

[54] Reed, G.M., Roscoe, A.W., "A Timed Model for Communicating SequentialProcesses". Programming Research Group, Oxford University, 1988.

[55] Sargent III, M., Shoenaker, L., "The IBM Personal Computer from the Inside Out". Addison-Wesley, 1984.

[56] Sha, L., "Real-Time System Engineering Methodology". IEEE Computer Society Real-Time Systems Newletter, Summer 1988.

[57] Sloman, M., Magee, J., Kramer, J., "Constructing Distributed Systems in CONIC". Department of Computing, Imperial College of Science and Technology. London, May 1989.

[58] Special Issue on SNA, IBM Systems Journal 22(4), 1983.

[59] Stankovic, J.A., "Real Time Computing Systems: The Next Generation". Hard Real-Time System-Tutorial. IEEE Computer Society Press, Capítulo 2.

[60] Stevens, W.P., Myers, G.F., Constantine, L.C., "Structured Design". IBM Systems Journal, vol. L3, no. 2, p. 115-139, 1974.

[61] Strom, R., Jemini, S.,"The Nil Distributed Systems Programming Language: A Status Report". ACM SIGPLAN Notices, 20(5), may, 1985.

[62] Tanembaum, A.S., Van Renesse, R., "Distributed Operating Systems". ACM Computing Serveny, 17(4), 419-70, 1985.

[63] Tanembaum, A.S., "Computer Networks". Englewood Cliffs, N.J. Prentice-Hall, 1988.

[64] Teng, A.Y, Lin, M.T., "A Formal Approach to the Design and Implementation of Network Communication Protocol". Anais da COMPSAC, 1978.

[65] UART 8250, Componentes Handbook, Western Digital Corporation, 1983.

[66] Walker, B., et al., " The LOCUS Distributed Operating System". Ninth ACM Symposium on Operating System Principles. ACM Operating System Review, vol. 17(5), p. 49-70, 1983.

[67] Wecker, S., "DNA: The Digital Network Architecture". IEEE Trans. on Comms, COM. 28(4). Apr., 1980.

[68] Wilkes, M.V., Needham, R.M., "The Cambridge Model Distributed System". Operating System Review, vol.14(1),p. 21-29, 1980.

[69] Wilson, P., "OCCAM Architecture Eases System Design ". Computer Design, Nov. 1983.

[70] Wirth, N., "MODULA: A Language for Modular Multiprogramming". Software, Practice and Experience, 7(1) : 3 - 35, Jan./Feb., 1977.

[71] Wirth,N., "Design and Implementation of Modula". Software,Practice and Experience, vol.7, 1977.

[72] Wirth,N., "Programming in Modula-2". New York, Heidelberg, Berlim: Springer-Verlag,1982.

[73] Wirth,N., "Toward a Discipline of Real-Time Programming". CACM, 20(8), Ago., 1977.

[74] Wittie,L.D., "MICROS, A distributed Operating System for MICRONET, A Reconfigurable Network Computer". IEEE Trans. on Computers, vol. 29(12), p. 1133-1144, 1980.

[75] Wulf,W.A.,Bell,C.G., "C.mmp - A Multi-Mini-Processor". Proceedings AFIPS 1972, Vol. 41, AFIPS Press, Montvale, p. 765-777, 1972.

[76] Wulf,W.A.,et al., "HYDRA: the Kernel of a Multiprocessor Operating System". CACM, 17(6), jun. 1974.

[77] Xerox Corporation. "MESA Language". Version 5.0, Xerox Corporation, 1985.

[78] Young, S.J., "Real Time Languages: Design and Development". Ellis Horwood Ltda., 1982.

[79] Zhao, W., Ramanrithan, K., Stankovic, J.A., "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems". IEEE Trans. on Software Engineering, May, 1987.

[80] Zimmermann, H., "OSI Reference Model - The ISO Model of Architecture for Open System Interconnection". IEEE Transactions on Communication, vol. COM-28, p. 425-432, 1980.

[81] Zwaenepoel, W., Lantz,K.A., "Perseus: Retrospective on a Portable Operating System". Software-Praictice and Experience, 14(1), p. 31-48, jan. ,1984.