UNIVERSIDADE FEDERAL DA PARAIBA

CENTRO DE CIENCIA E TECNOLOGIA

CURSO DE MESTRADO EM ENGENHARIA ELÉTRICA

IMPLEMENTAÇÃO DE UM SERVIDOR MMS PARA CONTROLADORES LÓGICOS PROGRAMAVEIS

Mônica Valeria Caldas de Aguiar

Campina Grande, Pb

Junho/1992

IMPLEMENTAÇAO DE UM SERVIDOR MMS PARA CONTROLADORES LóGICOS PROGRAMAVEIS

Mônica Valeria Caldas de Aguiar

Dissertação apresentada ao Curso de Mestrado em Engenharia Elétrica da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do grau de Mestre.

Joberto S. Barbosa Martins, Dr. Ing.
Orientador

Campina Grande, Pb
Junho/1992

L892p Loureiro, Ricardo Jorge Aguiar.
Projeto de eletrodos de blindagem para divisores de potencial resistivos / Ricardo Jorge Aguiar Loureiro. - Campina Grande, 1983.
86 f.

Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1983.
"Orientação : Prof. Dr. Klaus Nowacki".
Referências.

1. Linguagem de Programação - Engenharia Elétrica. 2. Eletrodos de Blindagem - Projeto. 3. Dissertação - Engenharia Elétrica. I. Nowacki, Klaus. II. Universidade Federal da Paraíba - Campina Grande (PB). III. Título

CDU 621:004.43(043)

IMPLEMENTAÇÃO DE UM SERVIDOR MMS PARA

CONTROLADORES LÓGICOS PROGRAMÁVEIS

MÔNICA VALERIA CALDAS DE AGUIAR

DISSERTAÇÃO APROVADA EM 05.06.1992

Joberto S. Barbosa Martins, Dr. Ing.
Orientador

Maria Izabel Cavalcanti Cabral, Dr.
Componente da Banca

Mario Toyotaro Hattori, Dr.
Componente da Banca

Gurdip Singh Deep, Ph.D.
Componente da Banca

Campina Grande, 05 de junho de 1992

AGRADECIMENTOS

A Deus, pela oportunidade e estímulo nos momentos mais difíceis.

Ao orientador Joberto Martins pela dedicação com que orientou este trabalho.

Ao meu marido Sérgio pelo apoio, carinho, compreensão e paciência em todos os momentos.

À minha família, pela força e estímulo durante todo o tempo do trabalho.

Agradeço a todos os professores, funcionários, colegas e amigos que contribuíram direta ou indiretamente para este trabalho. Ao professor Peter, pela boa vontade e confiança demonstradas durante os testes da implementação.

Um agradecimento especial ao casal de amigos Luciano e Lita que nos acolheram e ofereceram os recursos e o apoio necessários e essenciais à conclusão deste trabalho.

SUMARIO

Este trabalho descreve a implementação de um subconjunto dos serviços e do protocolo MMS (*Manufacturing Message Specification*), com o objetivo de permitir a troca de mensagens entre equipamentos do nível de chão de fábrica em uma manufatura integrada. O subconjunto de serviços e o protocolo definidos direcionam a utilização da implementação em dispositivos controladores lógicos programáveis (CLPs) operando como servidores em segmentos com arquitetura de protocolos Mini-MAP.

Inicialmente é apresentada uma descrição do MMS, bem como da arquitetura de protocolos para a qual esta implementação foi desenvolvida. É feita, também, uma apresentação das características e capacidades funcionais de um controlador lógico programável. Após a descrição do modelo de implementação é feita uma apresentação da plataforma de testes e dos resultados obtidos.

ABSTRACT

This work describes a service subset and a protocol implementation of the MMS application protocol (Manufacturing Message Specification), that allows message exchange between factory floor devices in an integrated manufacturing environment. The defined services subset and protocol direct the implementation to be used in programmable logic controller devices (PLCs) operating as servers in segments with Mini-MAP protocol architecture.

Initially, the work presents a survey of MMS services and the base architecture, followed by the PLC's functions description. After a description of a model implementation, the testing environment and the obtained results are described.

INDICE

LISTA DE FIGURAS E TABELAS

CAPITULO 4

CAPITULO 5

CAPITULO 6

ANEXO 3

LISTA DE SIGLAS

ACSE - *Association Control Service Element*

API - *Application Program Interface*

ASN.1 - *Abstract Syntax Notation One*

CAD - *Computer Aided Design*

CAE - *Computer Aided Engineering*

CAM - *Computer Aided Manufacturing*

CASE - *Common Application Service Element*

CBB - *Conformance Building Block*

CIM - *Computer Integrated Manufacturing*

CLNP - *Connectionless-mode Network Protocol*

CLNS - *Connectionless-mode Network Service*

CLP - Controlador Lógico Programável

CNC - Comando Numérico Computadorizado

DS - *Directory Service*

EIA - *Electronic Industries Association*

EPA - *Enhanced* Performance *Architecture*

ES-IS - *End Systems to Intermediate Systems Exchange Protocol*

FIFO - *First-in-First-out*

FTAM - *File Transfer Access and Management*

GM - *General Motors*

IEC - *International Electrotechnical Commission*

IEEE - *Institute of Electrical and Electronics Engineers*

IRSEC - *Implementation Release Subject to Errata Changes*

ISA - *Instruments Society of America*

ISO - *International Standards Organization*

LLC - *Logical Link Control*

LSAP - *Link Service Access Point*

LSDU - *Link Service Data Unit*

MAC - *Medium Access Control*

MAP - *Manufacturing Automation Protocol*

MMPM - *Manufacturing Message Protocol Machine*

MMS - *Manufacturing Message Specification*

NEMA - *National Electrical Manufacturing Association*

NM - *Network Management*

ORPM - *Obtain Reply Protocol Machine*

ORSE - *Obtain Reply Service Element*

PP - Processador Preferencial

RIA - *Robotic Industries Association*

RM-OSI - *Reference Model for Open Systems Interconnection*

SASE - *Specific Application Service Element*

UDP - Unidade de Dados de Protocolo

UE - *User Element*

VMD - *Virtual Manufacturing Device*

VTP - *Virtual Terminal Protocol*

CAPITULO 1

INTRODUÇÃO

Os sistemas computadorizados foram introduzidos no ambiente fabril há quase quarenta anos, para auxiliar no controle de processos [DAIGLE,1988]. Desde então, a automação tornou-se um instrumento fundamental, com o qual as indústrias procuram manter sua competitividade.

Com esse objetivo surgiu o conceito de manufatura integrada por computador (CIM - *Computer Integrated Manufacturing*), ou integração operacional. Integrar uma manufatura por computador significa interligar todos os sistemas computadorizados existentes na empresa, incluindo: controladores lógicos programáveis (CLPs), dispositivos de comando numérico computadorizado (CNCs), dispositivos de comando de robôs, etc., além da área administrativa e de projetos [SPROESSER,1989]. A integração operacional torna as empresas altamente competitivas e dinâmicas e entre os fatores que contribuem para isso pode-se citar:

- o aumento da qualidade dos produtos;
- a diminuição dos custos de produção, mão-de-obra e projetos;
- a diminuição do tempo de lançamento de novos produtos.

Uma das principais características da manufatura é a sua estrutura hierárquica [CHINTAMANENI,1988]. Essa estrutura define de três a cinco níveis (dependendo do enfoque de análise e da complexidade do processo), que englobam desde o nível de chão de fábrica (máquinas, robôs, sensores, atuadores, etc.) até o nível dos sistemas de administração e planejamento da fábrica (sistemas CAE/CAD/CAM, planejamento de produção, sistemas administrativos, etc.).

Cada nível possui características próprias de comunicação [DAIGLE,1988]. Nos níveis inferiores (chão de fábrica) predominam mensagens periódicas, curtas (centenas de bits), frequentes e com tempo de resposta pequeno (dezenas de micro segundos). Essas mensagens, ditas **de tempo real**, podem ser: comandos, respostas a comandos, dados de medidas de sensores, alarmes, etc.

Essa heterogeneidade nos requisitos de comunicação leva a uma diversificação dos sistemas automatizados mantidos por cada setor da empresa. A utilização de uma estratégia CIM deve efetuar a integração gradual desses sistemas [DAIGLE,1988].

Entretanto, algumas situações podem tornar-se um problema para a integração operacional [DAIGLE,1988]. São elas:

- comunicação entre aplicações dentro do mesmo computador;

- comunicação entre hospedeiro e periféricos do mesmo fabricante;

- comunicação entre sistemas de fabricantes diferentes: computadores, redes locais, controladores de célula, CLPs, robôs, máquinas-ferramenta e outros dispositivos de controle digital.

O problema no último caso é mais crítico: muitos dos equipamentos dispõem de alguma capacidade de comunicação e interconectá-los (quando possível) requer, frequentemente, projetos extensos de desenvolvimento e um número excessivo de equipamentos de comunicação e computadores [DAIGLE,1988].

Muitos esforços estão sendo feitos nos Estados Unidos e Europa para desenvolver um conjunto comum de protocolos de comunicação, independente de fabricante, que possa ser usado por todos os tipos e marcas de equipamentos da fábrica. Um dispositivo que implemente esses protocolos deve ser capaz de se comunicar de forma transparente com todos os outros [DAIGLE,1988].

Um dos projetos mais importantes de padronização de protocolos para a indústria é o MAP (*Manufacturing Automation Protocol*) [GM/MAP,1988] e as arquiteturas derivadas deste: EPA/MAP (EPA - *Enhanced Performance Architecture*) e Mini-MAP [GM/MAP,1988].

O MAP é uma arquitetura padrão desenvolvida especificamente para prover comunicação em uma manufatura

integrada por computador (ambiente CIM) [KUSIAK,1988]. Entre as normas que o MAP define estão [MENDES,1989]:

- arquitetura e topologias de rede seguindo os padrões IEEE (*Institute of Electrical and Eletronic Engineers*) e ISO (*International Standards Organization*) para as camadas 1 a 7 do modelo RM-OSI (*Reference Model for Open Systems Interconnection*) da ISO [ISO/OSI,1984a];

- funções de gerenciamento de redes;

- formatos e troca de mensagens nos dispositivos programáveis da fábrica, como CLPs, CNCs, CNRs, etc.

Essas normas tornam o MAP mais adequado às aplicações nos níveis inferiores da manufatura.

Entre os benefícios introduzidos com a utilização do MAP pode-se citar [KUSIAK,1988]:

- facilidade no projeto de sistemas com equipamentos de diferentes fabricantes, podendo-se selecionar sempre o melhor para cada aplicação;

- maior confiabilidade dos sistemas da manufatura, pois os componentes de hardware e software são desenvolvidos e dimensionados de acordo com o padrão MAP;

- possibilidade de instalação dos sistemas de manufatura em relativamente pouco tempo.

Um dos protocolos definidos na camada de aplicação proposta para o MAP é o protocolo MMS (*Manufacturing Message Specification*) [EIA/MMS,1987]. Este protocolo destina-se à transmissão de mensagens entre dispositivos programáveis e controladores de célula numa manufatura integrada por computador [McGUFFIN,1988].

O MMS é considerado por alguns especialistas como o padrão mais importante da arquitetura MAP [McGUFFIN,1988], pois seus serviços são concebidos especialmente para os dispositivos de manufatura, satisfazendo, portanto, aos requisitos CIM de controle e monitoramento de máquinas.

Devido à necessidade de se atender a vários dispositivos com requisitos diferentes, o MMS oferece um grande número de serviços. Entretanto, nas implementações específicas, utiliza-se apenas o subconjunto de serviços adequado ao dispositivo.

Os aspectos específicos do MMS associados a cada tipo de dispositivo são descritos em documentos denominados *Companion Standards*. Esses documentos descrevem, entre outras coisas, o mapeamento das funções do dispositivo em serviços MMS e a especificação das classes de serviço a serem suportadas pelo dispositivo.

Vários desses padrões estão em fase de elaboração: a NEMA (*National Electrical Manufacturing Association*) trabalha com aplicações para controladores lógicos programáveis [IEC/PCMS,1989]; a RIA (*Robotic Industries Association*) com aplicações para controladores de robôs

[ISO/RCMS,1989]; a EIA (*Electronic Industries Association*) com aplicações para dispositivos de comando numérico [EIA/NCMS,1987] e a ISA (*Instruments Society of America*) com aplicações de controle de processos.

O MMS segue um modelo cliente-servidor, onde o cliente solicita serviços e o servidor executa funções específicas do dispositivo que representa. Assim, uma estação cliente pode requisitar serviços a diversas estações servidoras, cada uma modelando determinado dispositivo.

As arquiteturas Mini-MAP e EPA/MAP surgiram como alternativa para o problema de tempo de resposta das redes MAP: a arquitetura completa introduz atrasos críticos na circulação das mensagens nas redes de chão de fábrica [MENDES,1989]. Outro problema associado ao MAP que se procurou solucionar é o volume de software necessário à implantação da arquitetura completa. O resultado foi a arquitetura Mini-MAP, composta de apenas três das camadas do MAP: física, enlace e aplicação.

Esta implementação fornece um subconjunto de serviços MMS que atende, no mínimo, aos requisitos de comunicação de um controlador lógico programável operando como servidor em segmentos Mini-MAP no nível de chão de fábrica. Esse subconjunto de serviços suporta três classes de funções do CLP. Entretanto, a estrutura foi definida de forma a permitir a inclusão do restante das funções (referentes às demais classes) sem alterações significati-

vas no código do programa. As alterações necessárias ficarão praticamente restritas às estruturas de dados.

O sistema foi desenvolvido como parte de um projeto de montagem de uma estação servidora Mini-MAP do Grupo de Redes de Computadores da UFPB (Universidade Federal da Paraíba), e faz uso de um Executivo implementado nesta Universidade [TURNELL,1990], integrado com uma implementação das camadas física e de enlace (*software* LLC e placa MAC) executável no Processador Preferencial (PP) [TELEBRAS,1987] ou em sistemas multi-usuário que apresentam recursos de comunicação interprocessos semelhantes.

A Figura 1.1 ilustra como o MMS (em destaque) se situa na arquitetura Mini-MAP, na qual foi baseado, e no ambiente de execução.

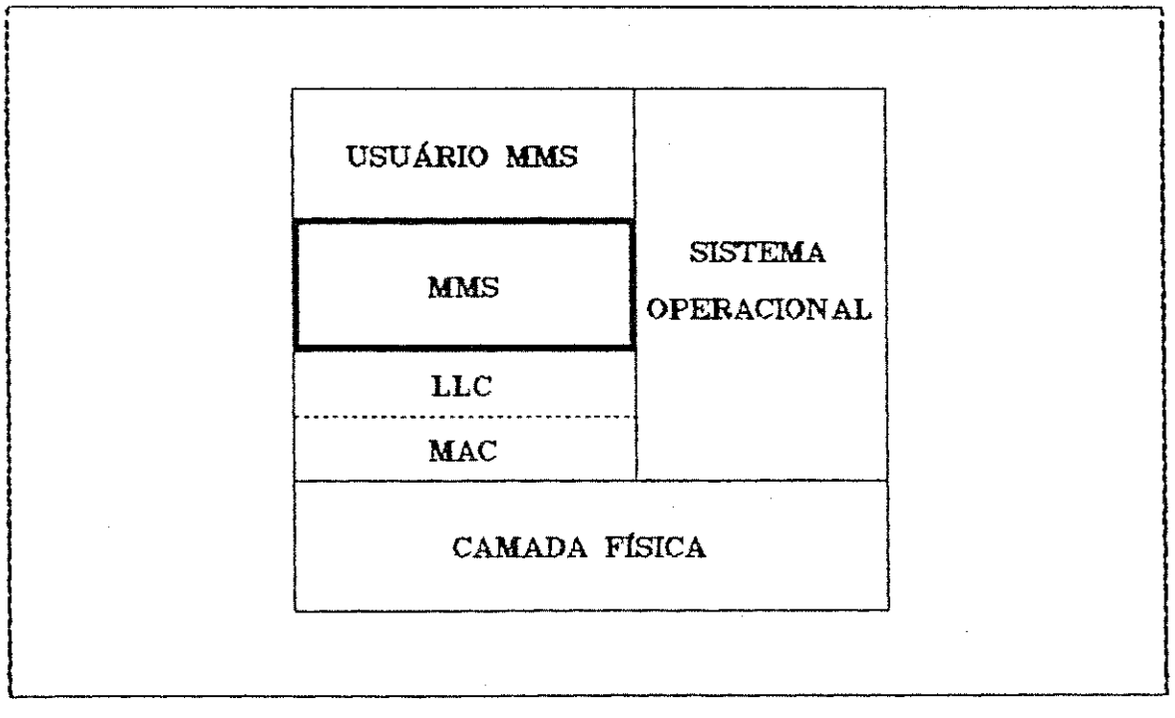


FIGURA 1.1 - O MMS NO MINI-MAP E NO AMBIENTE DE EXECUÇÃO

A seguir, será dada uma visão geral do conteúdo de cada capítulo deste trabalho:

O **CAPITULO 2** descreve as arquiteturas MAP, EPA/MAP e Mini-MAP.

O **CAPITULO 3** descreve os serviços e o protocolo MMS, segundo o documento *EIA Project 1393A Draft 6 - Manufacturing Message Specification* (partes 1 e 2) e as adaptações introduzidas pelo seu uso numa arquitetura Mini-MAP.

O **CAPITULO 4** descreve a estrutura física convencional de um controlador lógico programável, sua funcionalidade, as classes de serviço oferecidas pelo dispositivo e o mapeamento dessa funcionalidade em serviços MMS.

O **CAPITULO 5** detalha a implementação, descrevendo:

- estruturas de dados;
- estrutura de processos;
- interação com as camadas superior e inferior da arquitetura de protocolos;
- a estrutura das primitivas e UDPs (Unidade de Dados de Protocolo) em linguagem C.

O **CAPITULO 6** descreve a forma como foram feitos os testes da implementação e os resultados obtidos.

No **CAPITULO 7** é feita uma análise dos resultados finais, face aos objetivos propostos no início do tra-

balho de implementação. São feitas, também, algumas considerações acerca de implementações futuras baseadas neste trabalho.

O **ANEXO 1** contém uma descrição de cada serviço MMS.

O **ANEXO 2** contém as máquinas de estado do MMS para Mini-MAP.

O **ANEXO 3** contém a descrição do mapeamento das primitivas e UDPs de ASN.1 (*Abstract Syntax Notation One*) para a linguagem C.

CAPITULO 2

A ARQUITETURA DE PROTOCOLOS MAP

O projeto MAP teve início em 1980 [MENDES,1989], quando a GM (*General Motors*) criou um grupo de trabalho MAP, com o objetivo de preparar a especificação de um sistema aberto de comunicação para dispositivos utilizados em ambiente industrial. O primeiro documento MAP foi publicado em outubro de 1982. Em abril de 1984 foi publicada a versão 1.0 das especificações MAP. A versão 2.0 foi publicada em fevereiro de 1985 e, até 1986, foram publicadas as revisões 2.1, 2.1A e 2.2, com a inclusão do Mini-MAP e do EPA/MAP. Em junho de 1987 foi publicada uma versão **MAP 3.0** provisória (IRSEC - *Implementation Release Subject to Errata Changes*). Finalmente, em junho de 1988 foi lançada a versão definitiva do MAP 3.0 [GM/MAP,1988].

O MAP foi desenvolvido para tornar possível a comunicação entre dispositivos como computadores de grande porte, micro/mini/supermini-computadores, controladores de célula, estações CAE/CAD/CAM, dispositivos programáveis (CNCs, CNRs, controladores de solda, etc.), terminais de coleta de dados, monitores, etc.

Apesar do MAP destinar-se aos níveis mais baixos da hierarquia de controle da fábrica (controle de processos e controle de células), observou-se que os tempos de resposta não satisfaziam aos requisitos de tempo real destes níveis [GM/MAP,1988]. Surgiram, assim, propostas, como a

arquitetura EPA/MAP (EPA - *Enhanced Performance Architecture*) e a arquitetura Mini-MAP, que fazem parte da especificação MAP a partir da versão 2.2.

Essas arquiteturas são descritas a seguir.

2.1 A ARQUITETURA MAP

A arquitetura MAP, mostrada na Figura 2.1, consiste de protocolos padronizados pelo IEEE/ISO, estruturados segundo o modelo de referência OSI da ISO [ISO/OSI,1984]. A descrição de cada camada da arquitetura é dada a seguir.

APLICAÇÃO	- SASE: MMS, Diretório, Gerenc.de Rede, FTAM - CASE: ACSE
APRESENTAÇÃO	ISO 8822/8823
SESSÃO	ISO 8326/8327
TRANSPORTE	ISO 8072/8073, Classe IV
REDE	- ISO 8348/8473 - ISO 9542
ENLACE	- LLC IEEE 802.2 Tipo 1 - MAC IEEE 802.4
FÍSICA	- IEEE 802.4 Banda Larga, a 10 MBPS - IEEE 802.4 Banda Portadora, a 5 MBPS

FIGURA 2.1 - ARQUITETURA MAP

2.1.1 CAMADA FISICA

Suporta dois conjuntos de especificações:

Transmissão em Banda Larga

- Tipo de modulação: duo-binária AM-PSK (*Amplitude Modulation - Phase Shift-Keying*)
- Meio de transmissão: cabo coaxial de 75 Ω

- Capacidade de transmissão: 10 Mbps
- Topologia: barramento com ficha segundo o padrão IEEE 802.4 [IEEE,1982]
- Alcance máximo do segmento: 3750 m
- Número máximo de nós do segmento: 1024

Transmissão em Banda Portadora

- Tipo de modulação: fase coerente FSK (*Frequency Shift-Keying*)
- Meio de transmissão: cabo coaxial de 75 Ω
- Capacidade de transmissão: 5 Mbps
- Topologia: barramento com ficha segundo o padrão IEEE 802.4 [IEEE,1982]
- Alcance máximo do segmento: 1000 m
- Número máximo de nós do segmento: 32

2.1.2 CAMADA DE ENLACE

É subdividida em duas sub-camadas:

A **sub-camada MAC** (*Medium Access Control*) utiliza o padrão IEEE 802.4, o qual especifica um protocolo de passagem de ficha num meio físico com topologia em barramento (*Token Passing Bus*).

A **sub-camada LLC** (*Logical Link Control*) utiliza o padrão IEEE 802.2 [IEEE,1983] Tipo 1, que oferece serviços sem conexão, sem reconhecimento, sem controle de fluxo e sem recuperação de erros. Nesse caso, esses recursos são providos pela camada de transporte.

2.1.3 CAMADA DE REDE

A camada de rede do MAP suporta apenas serviços sem conexão, descritos no documento ISO 8348 (CLNS - *Connectionless-mode Network Service*) [ISO/OSI,1984b]. O protocolo adotado segue a especificação contida no documento ISO 8473 (CLNP - *Connectionless-mode Network Protocol*) [ISO/OSI,1984b], que define três tipos de funções:

- **Tipo 1:** mandatário;
- **Tipo 2:** funções de segurança e roteamento completo;
- **Tipo 3:** funções de roteamento parcial, prioridades, memorização de rotas, etc.

O MAP suporta apenas o tipo 1 e a função de memorização de rotas do tipo 3.

Além desse protocolo, o MAP também especifica um protocolo de roteamento entre sistemas finais (nós MAP) e sistemas intermediários: o ISO 9542 (*ES-IS - End Systems to Intermediate Systems Exchange Protocol*) [ISO/OSI,1984c]. Os sistemas finais implementam as sete camadas de protocolo do

modelo OSI [MENDES,1989]. Os sistemas intermediários são utilizados como retransmissores ou roteadores, implementando, em geral, até a camada de rede, mas podendo suportar também as quatro camadas superiores do modelo OSI. Os sistemas intermediários são classificados em repetidores, pontes, roteadores e comportas.

2.1.4 CAMADA DE TRANSPORTE

Adota os serviços e o protocolo conforme definidos no documento ISO 8072 e 8073 [ISO/OSI,1984d], Classe IV (*Error Detection and Recovery Class*). Essa classe oferece serviços com conexão, reconhecimento e controle de fluxo, efetua multiplexagem de várias conexões da camada superior na mesma conexão de transporte e oferece mecanismos de detecção e recuperação de erros para pacotes fora de sequência, perdidos, duplicados ou destruídos.

2.1.5 CAMADA DE SESSAO

Adota a especificação ISO 8326 para os serviços e a ISO 8327 para o protocolo [ISO/OSI,1984e]. As unidades funcionais requeridas para o MAP são: *Kernel*, que oferece serviços básicos de estabelecimento e término de conexão, bem como de transferência de dados normais; *Duplex*, para transferência de dados nos dois sentidos; e *Resynchronize*, recomendado apenas para alguns protocolos de aplicação com definição de pontos intermediários de sincronização. Um

exemplo de protocolo de aplicação que utiliza a unidade *Resynchronize* é o protocolo FTAM (*File Transfer Access and Management*) para transferência de arquivos [ISO/FTAM,1987].

2.1.6 CAMADA DE APRESENTAÇÃO

Utiliza o protocolo e os serviços especificados nos documentos ISO 8823 e 8822 [ISO/OSI,1986a] respectivamente, adotando-se a unidade funcional *Kernel*. É sugerido o uso da sintaxe de transferência ASN.1 (*Abstract Syntax Notation One*) [ISO/ASN1,1986] para codificar as unidades de dados.

2.1.7 CAMADA DE APLICAÇÃO

A camada de aplicação no MAP segue o modelo OSI [ISO/OSI,1986b] com uma estruturação composta por três tipos de elementos:

- elementos de serviço comuns (CASE - *Common Application Service Elements*) [ISO/OSI,1986c] [ISO/OSI,1986d];
- elementos de serviço específicos (SASE - *Specific Application Service Elements*);
- elementos de usuário (UE - *User Elements*).

A sub-camada definida pelo **CASE** provê serviços básicos comuns às diversas aplicações, tais como: estabelecimento e término de associação, transferência de informação e sincronização, dentre outros. Dos protocolos definidos pelo CASE, o MAP 3.0 adota apenas o ACSE (*Association Control Service Elements*). O ACSE forma o núcleo básico e obrigatório do CASE, sendo responsável pelo estabelecimento e término de associações entre protocolos de aplicação específicos (SASEs).

A sub-camada definida pelo **SASE** inclui protocolos específicos para cada aplicação. O MAP 3.0 define para o SASE:

- **MMS** (*Manufacturing Message Specification*), para troca de mensagens entre dispositivos programáveis da fábrica, segundo o padrão ISO 9506 [GM/MMS,1987];

- **Serviço de Diretório** (DS - *Directory Service*), para gerenciamento de nomes simbólicos usados para identificar sistemas e aplicações na rede, segundo o padrão ISO 9594/1-8 [ISO/OSI,1986d];

- **Serviço de Gerenciamento de Rede** (NM - *Network Management*), que permite gerenciar aspectos funcionais da rede, como desempenho, segurança, configuração da rede, etc, segundo o padrão ISO 9595 e 9596 [ISO/OSI,1986e].

- **FTAM** (*File Transfer Access and Management*), para transferência de arquivos, segundo o padrão ISO 8571 1-4 [ISO/FTAM,1987].

Os **elementos de usuário** (ou **aplicações de usuário**) incluem as APIs (*Application Program Interface*) e os programas escritos pelo usuário. A API é uma biblioteca de funções utilizada pelo programa do usuário para obter serviços da camada de aplicação, sem ser necessário o conhecimento, por parte do usuário, dos detalhes de implementação da camada de aplicação.

Nas fases de estabelecimento e liberação de associações, a aplicação do usuário utiliza os serviços ACSE (por intermédio do SASE). Na fase de transferência de dados, a aplicação do usuário utiliza os serviços da camada de apresentação (através do SASE) (Figura 2.2).

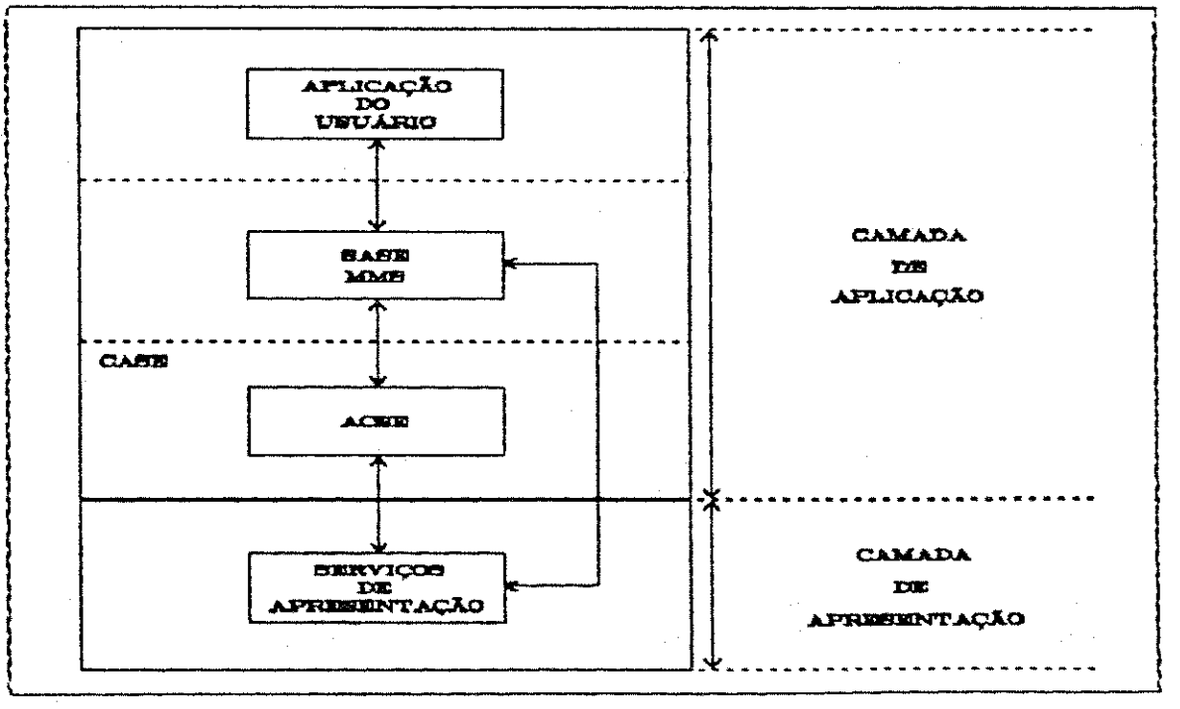


FIGURA 2.2 - A CAMADA DE APLICAÇÃO DO MAP

2.2 A ARQUITETURA MINI-MAP

A arquitetura Mini-MAP é constituída por apenas três das camadas da arquitetura MAP: camada física, camada de enlace e camada de aplicação.

O Mini-MAP é mais simples e de menor custo em relação à estrutura completa MAP e, além disso, permite que alguns dispositivos atuem como nós passivos no segmento. Nós passivos são estações que não participam do anel lógico de circulação da ficha, mas são capazes de responder, através do mecanismo de resposta imediata, a uma requisição do nó possuidor da ficha. Assim, dispositivos de baixo nível (como sensores), que só enviam informações quando solicitados, podem atuar como nós passivos.

Os protocolos definidos para cada camada desta arquitetura são mostrados na Figura 2.3 e descritos a seguir.

APLICAÇÃO	SASE : MMS
ENLACE	- LLC IEEE 802.2 Tipos 1 e 3 - MAC IEEE 802.4
FÍSICA	- IEEE 802.4 Banda Larga, a 10 MBPS - IEEE 802.4 Banda Portadora, a 5 MBPS

FIGURA 2.3 - ARQUITETURA MINI-MAP

2.2.1 CAMADA FISICA

Suporta as duas opções especificadas no MAP (item 2.1). A opção de banda portadora, por permitir um tempo de resposta da ordem de 25 ms e ser de menor custo, é a mais recomendada [GM/MAP,1988].

2.2.2 CAMADA DE ENLACE

A **sub-camada MAC** utiliza o padrão IEEE 802.4, sendo usadas as seguintes opções no caso de redes com restrição de tempo: serviços com prioridade, serviços de resposta imediata e endereços de 48 bits.

A **sub-camada LLC** é definida pelo padrão IEEE 802.2. Para situações normais de transmissão de dados utiliza-se o LLC Tipo 3, que oferece serviços sem conexão, com reconhecimento, com controle de fluxo e com recuperação de erros. Para mensagens tipo difusão utiliza-se o LLC Tipo 1.

2.2.3 CAMADA DE APLICAÇÃO

Além do protocolo MMS e de alguns serviços simplificados de diretório, especifica também o ORSE (*Obtain Reply Service Element*), um SASE adicional ao MMS. O ORSE dá suporte ao serviço de *polling* efetuado pelo LLC Tipo 3, recurso usado pelo usuário MMS para solicitar serviços de estações não participantes do anel lógico (formado pela estrutura do IEEE 802.4). Tais serviços são ditos **de resposta imediata.**

2.3 A ARQUITETURA EPA/MAP

Um nó EPA/MAP é composto de uma pilha dupla de protocolos, como mostra a Figura 2.4. Uma pilha é a arquitetura MAP completa (*FULL MAP*) e a outra é a estrutura Mini-MAP.

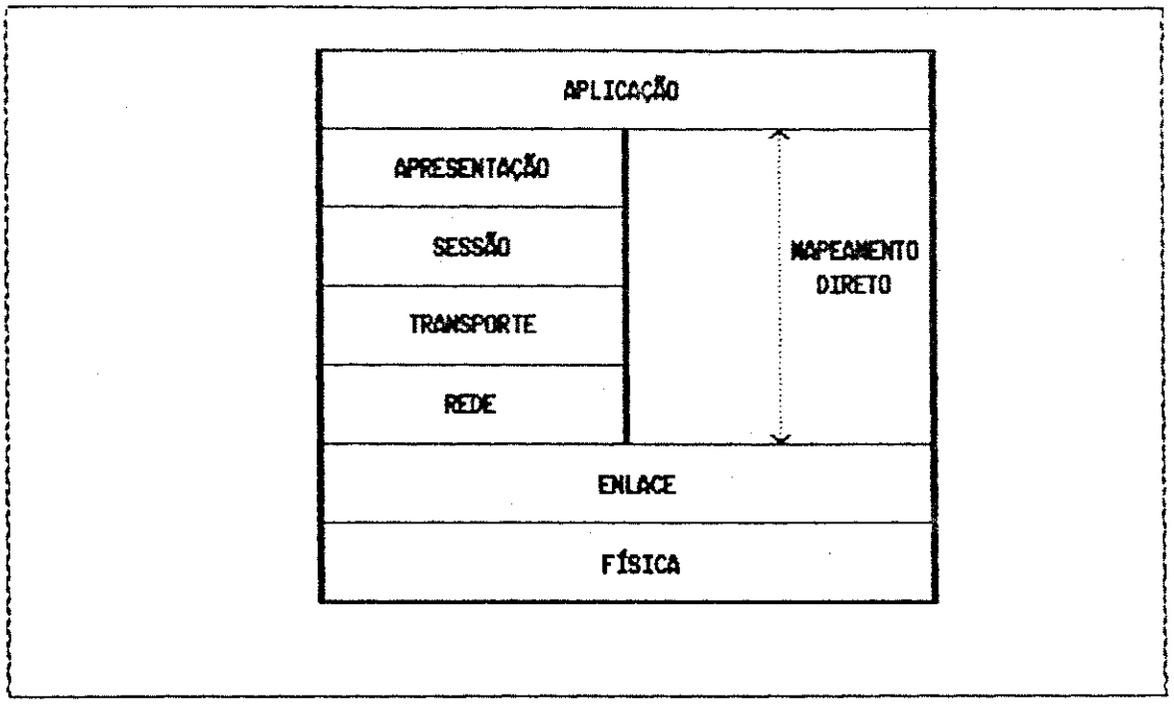


FIGURA 2.4 - ARQUITETURA EPA/MAP

Se um nó requer a utilização de determinados serviços de comunicação não encontrados no Mini-MAP, deve-se utilizar a arquitetura completa (sete camadas). Para respostas mais rápidas em aplicações de tempo real utiliza-se a arquitetura Mini-MAP.

Um segmento Mini-MAP não se comunica diretamente com um segmento MAP e, por isso, a tendência é se utilizar

os nós EPA/MAP quase exclusivamente como comportas, interligando esses segmentos.

Esta implementação baseia-se na utilização de um segmento Mini-MAP cujo mecanismo de controle de acesso ao meio segue o padrão IEEE 802.4, com suporte para serviços prioritários e de resposta imediata, permitindo, assim, a utilização de nós passivos na rede; a sub-camada LLC segue o padrão IEEE 802.2 Tipo 3.

CAPITULO 3

O PROTOCOLO MMS
DE
MENSAGENS NA MANUFATURA

Este capítulo introduz o conceito de dispositivo virtual associado ao MMS e descreve os serviços e o protocolo especificados para o MMS.

Como a implementação é voltada para o Mini-MAP, descreve-se, também, as alterações introduzidas pelo Mini-MAP no MMS para adaptá-lo ao LLC [GM/MAP,1988], bem como a interface necessária para a troca de mensagens com a camada de enlace (LLC).

3.1 O MMS NO MAP

Esta secção descreve a especificação EIA Project 1393A Draft 6 [EIA/MMS,1987] para os serviços e o protocolo MMS de mensagens na manufatura definidos para as arquiteturas padronizadas e, em particular, para a arquitetura MAP. Essa especificação é idêntica à ISO 2nd DP 9506 Parte 1 para os serviços e Parte 2 para o protocolo.

3.1.1 O DISPOSITIVO VIRTUAL DE MANUFATURA

A padronização da comunicação entre os dispositivos de controle e monitoramento da manufatura torna necessária a utilização de modelos abstratos. Esses modelos representam cada dispositivo de forma equivalente, e são denominados Dispositivos Virtuais da Manufatura (VMD = *Virtual Manufacturing Device*) [EIA/MMS,1987].

Como já foi dito, os serviços MMS são especificados com referência a um modelo **cliente-servidor**. O servidor é o sistema que se comporta externamente como um VMD para cada solicitação de serviço [EIA/MMS,1987]. O cliente é o sistema que utiliza o VMD com determinado propósito, através de uma solicitação de serviço [EIA/MMS,1987].

O VMD modela o comportamento do servidor MMS visível ao usuário, representando, de forma abstrata, um conjunto específico de recursos e funções do dispositivo real. Pode-se representar um dispositivo real com um ou mais VMDs. Entretanto, na representação de um dispositivo com

vários VMDs os recursos modelados por cada VMD são distintos e independentes dos recursos modelados pelos outros VMDs. Por exemplo: um dispositivo composto de memória, unidade de processamento e interfaces de entrada e saída pode ser representado com três VMDs, onde um VMD representa a memória, outro representa a unidade de processamento e outro representa as interfaces de entrada e saída.

Na notação OSI o VMD reside no programa de aplicação do servidor. Cada programa de aplicação pode conter zero ou vários VMDs, representando cada dispositivo real e logicamente separado dos demais. Um programa de aplicação que não define pelo menos um VMD não pode funcionar como um servidor, apenas como cliente.

Um nó MMS, no qual estão conectados vários dispositivos da manufatura, poderá ser modelado com um único programa de aplicação, contendo um VMD para cada dispositivo, ou por vários programas de aplicação, cada um com um VMD distinto. Os clientes dos VMDs verão cada dispositivo como um único VMD ou como vários VMDs.

A Figura 3.1 mostra a estrutura do VMD, que é composto de uma Função Executiva, um ou mais Domínios, zero ou mais Estações do Operador e um Sistema de Arquivo Virtual (opcional). Esses elementos são descritos a seguir:

Função Executiva

Administra o acesso do MMS aos recursos do VMD (tais como memória, processadores, portas de I/O, etc.), tornando esses recursos visíveis aos clientes-usuários. A

Função Executiva exprime os recursos do VMD na forma de **capacidades**, que são usadas para descrever os elementos constituintes do domínio e para verificar a validade do acesso aos objetos MMS. Os objetos MMS são:

- semáforos;
- condições de eventos;
- ações de eventos;
- "jornais";
- domínios;
- invocações de programas;
- estações de operador;
- variáveis nomeadas;
- variáveis sem nome;
- variáveis dispersas;
- lista de variáveis com nome;
- tipos com nome.

Domínio (s)

O VMD tem no mínimo um domínio, chamado ***M_Executive***, que contém todos os recursos do VMD. Se o VMD contém mais de um domínio, esses recursos estarão distribuídos entre os domínios e não mais concentrados no ***M_Executive***.

Cada domínio representa um subconjunto dos recursos do VMD, o qual é usado com uma finalidade específica. Esse subconjunto refere-se a aspectos do VMD associa-

dos a um elemento específico (possivelmente todos) de uma estratégia coordenada de controle e/ou monitoramento.

A alocação de domínios pode ser estática ou dinâmica. Na alocação estática o domínio é pré-definido no servidor MMS, não pode ser apagado e seu nome é conhecido. Na alocação dinâmica o domínio é criado e removido do VMD por serviços MMS. Em um VMD específico pode-se encontrar um dos dois ou ambos os tipos de domínio.

Estação(ões) do Operador

O VMD pode ter zero ou mais estações de operador.

A estação do operador é uma classe especial de dispositivos, que efetua a troca de informações com outras estações de operador, as quais fazem parte de outros VMDs.

A estação do operador é evocada como um objeto separado do VMD em virtude de sua importância operacional, recebendo posteriormente primitivas de serviço específicas.

Memória Virtual de Arquivos (Opcional)

Consiste de uma coleção de arquivos com nome e age como um armazenador de dados e programas. Esses arquivos podem ser usados pelos serviços de Gerenciamento de Domínio e de Gerenciamento de Programa.

Como o nome indica, a Memória Virtual de Arquivos é um objeto virtual e, portanto, a implementação deve mapeá-lo para a memória real de arquivos.

Para que haja compatibilidade com o padrão ISO de transferência de arquivos (FTAM) [ISO/FTAM,1987], a memória virtual de arquivos do MMS é definida como um subconjunto do sistema de arquivo virtual do FTAM. Assim, um arquivo real pode ser mapeado tanto para o FTAM como para o sistema de arquivo virtual do MMS.

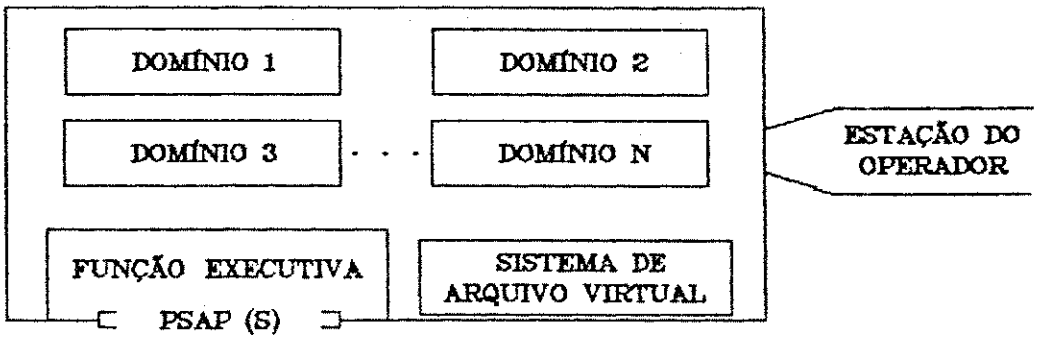


FIGURA 3.1 - ESTRUTURA DO VMD

3.1.2 OS SERVIÇOS MMS

Os serviços MMS são fornecidos pelo PROVEDOR MMS e estão distribuídos em Unidades Funcionais.

Na Tabela 3.1 são mostradas as Unidades Funcionais MMS e os serviços que incluem. Uma descrição de cada unidade é dada a seguir e a apresentação detalhada de cada serviço MMS pode ser encontrada no Anexo 1.

TABELA 3.1 - UNIDADES FUNCIONAIS MMS E SERVIÇOS CORRESPONDENTES

UNIDADE FUNCIONAL	SERVIÇOS MMS
GERENCIAMENTO DE CONTEXTO	Initiate, Conclude, Abort*, Cancel, Reject**
SUPORTE AO VMD	Status, Unsolicited Status*, Get Name List, Identify, Rename
ACESSO A VARIAVEIS	Read, Write, Information Report*, Define Scattered Access, Define Named Type, Define Named Variable, Define Named Variable List, Get Named Type Attributes, Get Variable Access Attributes, Get Named Variable List Attibutes, Get Scattered Access Attributes, Delete Variable Access, Delete Named Variable List, Delete Named Type
GERENCIAMENTO DE EVENTOS	Define Event Condition, Define Event Action, Define Event Enrollment, Delete Event Action, Delete Event Condition, Delete Event Enrollment, Get Event Action Attributes, Get Event Condition Attributes, Get Event Enrollments, Get Alarm Summary, Get Alarm Enrollment Summary, Report Event Action Status, Report Event Condition Status, Report Event Enrollment Status, Alter Event Condition Monitoring, Alter Event Enrollment, Trigger Event, Event Notification*, Acknowledge Event Notification, Attach To Event Condition Modifier
GERENCIAMENTO DE SEMAFOROS	Define Semaphore, Delete Semaphore, Report Pool Semaphore Status, Attach To Semaphore Status, Take Control, Report Semaphore Entry Status, Report Semaphore Status, Relinquish Control
GERENC. DE INVOCAÇÃO DE PROGRAMA	Create Program Invocation, Delete Program Invocation, Start, Stop, Resume, Reset, Kill, Get Program Invocation Attribute
COMUNICAÇÃO ENTRE OPERADORES	Input, Output
GERENCIAMENTO DE "JORNAL"	Read Journal, Write Journal, Initialize Journal, Report Journal Status
GERENCIAMENTO DE DOMINIO	Initiate Download Sequence, Download Segment, Terminate Download Sequence, Initiate Upload Sequence, Upload Segment, Terminate Upload Sequence, Request Domain Download, Request Domain Upload, Load Domain Content, Store Domain Content, Delete Domain, Get Domain Attribute, Obtain File
GERENCIAMENTO DE ARQUIVOS	File Open, File Read, File Close, File Rename, File Delete, File Directory

* - Serviço não confirmado: gera apenas primitivas *request* e *indication*;

** - Serviço gera apenas primitiva *indication*.

Gerenciamento de Contexto

Inclui serviços que permitem ao cliente MMS a abertura e encerramento de associações, de forma normal ou abrupta, o cancelamento de serviços pendentes e a rejeição de UDPs (Unidades de Dados de Protocolo) inválidas.

Suporte ao VMD

Possibilita ao usuário MMS obter informações diversas do VMD: *status* do VMD, identificação do dispositivo representado pelo VMD e lista de nomes dos objetos MMS definidos no VMD. Permite também a troca do nome de um objeto do VMD.

Acesso a Variáveis

Define serviços que permitem ao usuário MMS ter acesso a variáveis do VMD através de leitura (aquisição de medidas) ou escrita (regulagem de parâmetros).

No MMS há cinco tipos de objetos de acesso a variáveis: variável com nome, variável sem nome, variáveis dispersas, lista de variáveis com nome e tipos com nome.

Os objetos **variável com nome** (ou **nomeadas**) e **variável sem nome** descrevem o mapeamento entre variáveis MMS e variáveis reais no VMD. As variáveis sem nome estão mais próximas da arquitetura física do dispositivo real e são obtidas através de endereços do dispositivo. As variáveis nomeadas são parte da aplicação e são referenciadas através de nomes definidos no escopo da aplicação.

Os objetos **variáveis dispersas** e **lista de variáveis com nome** descrevem o acesso a múltiplas variáveis MMS

usando um único nome. As variáveis dispersas são variáveis independentes compondo uma estrutura que é mapeada em uma variável MMS. A lista de variáveis com nome é uma lista de variáveis independentes, que é mapeada em uma variável MMS.

O objeto **tipo com nome** permite a atribuição de um nome a uma descrição de tipo MMS. O nome é formado por letras, símbolos ou números e ocupa, no máximo, 16 bytes.

Gerenciamento de Eventos

O cliente MMS pode definir e gerenciar eventos no VMD e obter notificações sobre a ocorrência de eventos.

Os serviços desta unidade incluem três objetos: condição de evento (*event condition*), ação de evento (*event action*) e registro de evento (*event enrollment*). Cada um desses objetos modela um aspecto específico das informações de estado associadas ao gerenciamento de eventos MMS.

O objeto **condição de evento** modela os aspectos das informações de estado referentes à detecção e determinação de prioridades de eventos, incluindo informações que auxiliam na determinação de alarmes ativos. Define-se duas classes de condição de evento: *network-triggered* e *monitored*. Na classe *network-triggered* o evento ocorre a partir de uma requisição explícita do cliente. A classe *monitored* define uma condição de evento cujas transições de estado são detectadas pelo VMD.

O objeto **ação de evento** modela os aspectos das informações de estado referentes à execução de serviços MMS a partir da ocorrência de um evento.

O objeto **registro de evento** é usado para vincular condições e ações de evento, bem como para relacionar a um cliente notificações resultantes de uma transição de evento. Inclui também informações que podem ser usadas para localizar respostas de clientes a notificações de eventos do tipo alarme.

Gerenciamento de Semáforos

Coordena, sincroniza e controla recursos compartilhados por vários usuários MMS, através de semáforos do tipo *token* (com um ou mais proprietários) e *pool* (para alocação explícita, dinâmica, de *tokens* nomeados).

Gerenciamento de Invocação de Programa

Agrega conjuntos de elementos de dados e comandos (instruções) contidos em domínios do VMD.

Os serviços desta unidade permitem ao cliente a criação, eliminação, mudança de estados e obtenção de atributos de invocações de programa no VMD. Mudando o estado de uma invocação de programa o cliente pode iniciar uma execução de programa, suspender uma execução, reiniciar uma execução suspensa, **matar** uma execução ou colocar o programa em um estado que permita iniciar sua execução.

Comunicação entre Operadores

Permite a comunicação entre operadores de terminais alfa-numéricos através de operações de entrada e saída, sendo bem mais simples que o protocolo de terminal virtual (VTP - *Virtual Terminal Protocol*).

Gerenciamento de *Journal*

O cliente pode gravar e recuperar (no servidor) informações em ordem cronológica de eventos, variáveis de interesse relativas a eventos e textos (comentários e explicações).

O *journal* é um arquivo com nome e zero ou vários registros. Cada registro contém campos de identificação e campos de informação. Os campos de informação podem conter:

- Comentários colocados pelo MMS cliente para documentar determinada condição de evento ou conjunto de condições de evento, ou
- O nome de uma condição de evento e/ou o nome e o valor de uma ou mais variáveis.

Gerenciamento de Domínio

Permite a manipulação dinâmica de domínios, com criação e eliminação de domínios a partir do cliente ou localmente no servidor, além da transferência de domínios do servidor para o cliente (*upload*) e vice-versa (*download*).

Gerenciamento de Arquivos

Inclui serviços opcionais utilizados para gerenciamento de arquivos remotos (contendo programas e dados) localizados em dispositivos de controle ou em servidores de arquivos.

3.1.3 BLOCOS DE CONFORMIDADE

As unidades funcionais MMS são subdivididas em blocos verticais de conformidade (CBB - *Conformance Building Block*), como mostra a Tabela 3.2. Os serviços agrupados em um mesmo CBB estão relacionados através das operações que efetuam em um objeto MMS.

TABELA 3.2 - BLOCOS DE CONFORMIDADE VERTICAL MMS

UNIDADE FUNCIONAL	CBB	SERVIÇOS
GERENCIAMENTO DE CONTEXTO	CON1	Initiate, Conclude
	CON2	Initiate, Conclude
	CON3	Cancel
	MMS1	Abort, Reject
SUPORTE AO VMD	VMD1	Status, UnsolicitedStatus
	VMD2	Rename
	MMS2	Status, GetNameList, Identify
COMUNICAÇÃO ENTRE OPERADORES	OCS1	Output
	OCS2	Input
GERENCIAMENTO DE SEMAFOROS	SEM1	Take Control, Relinquish Control, Report Semaphore Status, Report Pool Semaphore Status, Report Semaphore Entry Status
	SEM2	Take Control, Relinquish Control, Report Semaphore Status, Report Pool Semaphore Status, Report Semaphore Entry Status, Attach To Semaphore Modifier
	SEM3	Take Control, Relinquish Control, Define Semaphore, Delete Semaphore, Report Semaphore Status, Report Pool Semaphore Status, Report Semaphore Entry Status
GERENCIAMENTO DE DOMINIO	DOM1	Initiate Download Sequence, Download Segment, Terminate Download Sequence
	DOM2	Initiate Upload Sequence, Upload Segment, Terminate Upload Sequence
	DOM3	Request Domain Download, Initiate Download Sequence, Download Segment, Terminate Download Sequence
	DOM4	Request Domain Upload, Initiate Upload Sequence, Upload Segment, Terminate Upload Sequence
	DOM5	LoadDomainContent
	DOM6	StoreDomainContent
	DOM7	DeleteDomain
	DOM8	GetDomainAttribute
	FIL1	ObtainFile

(cont. da tabela 3.2)

GERENCIAMENTO DE EVENTOS	EVN1	Get Event Condition Attributes, Report Event Condition Status
	EVN2	Define Event Condition, Event Notification, Delete Event Condition, Alter Event Condition Monitoring
	EVN3	Acknowledge Event Notification, Alter Event Enrollment, Define Event Action, Define Event Enrollment, Delete Event Action, Delete Event Enrollment, Event Notification, Get Event Action Attributes, Get Event Enrollments, Report Event Action Status, Report Event Enrollment Status
	EVN4	Attach To Event Condition Modifier, Trigger Event
	EVN5	Get Alarm Summary
	EVN6	Get Alarm Enrollment Summary
ACESSO A VARIAVEIS	VAR1	Get Variable Access Attributes
	VAR2	Read
	VAR3	Write
	VAR4	Information Report
	VAR5	Define Named Variable List, Delete Named Variable List, Get Named Variable List Attributes
	VAR6	Define Named Variable, Delete Variable Access
	VAR7	Define Scattered Access, Get Scattered Access Attributes
	VAR8	Get Named Type Attributes
	VAR9	Define Named Type, Delete Named Type
GERENCIAMENTO DE "JORNAL"	JOU1	Write Journal
	JOU2	Read Journal, Initialize Journal, Report Journal Status
GERENCIAMENTO DE INVOCAÇÃO DE PROGRAMA	PRG1	Create Program Invocation, Delete Program Invocation
	PRG2	Start
	PRG3	Stop
	PRG4	Resume
	PRG5	Reset
	PRG6	Kill
	PRG7	Get Program Invocation Attribute
GERENCIAMENTO DE ARQUIVOS	FIL2	File Open, File Read, File Close
	FIL3	File Rename, File Delete, File Directory

Além dos CBBs verticais, o MMS define também CBBs horizontais. Esses CBBs agrupam serviços MMS contendo determinados parâmetros que caracterizam a implementação do ponto de vista de definição de tipos de objetos MMS, definição de acesso aos objetos MMS e definição de aspectos de funcionamento do VMD. Apenas as unidades funcionais de Gerenciamento de Domínio e Acesso a Variáveis possuem CBBs

horizontais. Na Tabela 3.3 pode-se observar os CBBs horizontais e seus efeitos na implementação quando são habilitados nas primitivas de serviços.

TABELA 3.3 - BLOCOS DE CONFORMIDADE HORIZONTAL MMS

UNIDADE FUNCIONAL	CBB	EFEITOS
GERENCIAMENTO DE DOMINIO	SOCS	Habilita a implementação a se comunicar com servidores subordinados, que não sejam clientes MMS ou servidores MMS.
	RCV1	Habilita o VMD a manter um domínio no estado *LOADING*, durante uma falha de comunicação na associação, e recuperar a sequência de *download*.
	DTR1	Habilita o VMD a colocar um domínio em operação e intercalar um pedido de serviço do tipo *download segment* com outras operações do VMD.
ACESSO A VARIAVEIS	STR1	Permite a utilização de variáveis do tipo *ARRAY* e a seleção de um ou de vários elementos de um *array* no acesso alternativo.
	STR2	Permite a utilização de variáveis do tipo *STRUCTURE* e a seleção de um componente de uma estrutura no acesso alternativo.
	VNAM	Permite a utilização de variáveis nomeadas.
	VALT	Permite a utilização de acesso alternativo e nome de componente, o qual identifica univocamente o componente de uma estrutura.
	VADR	Valida o parâmetro endereço para utilização de variáveis não nomeadas.
	VSCA	Permite a utilização de variáveis dispersas.
	NEST	Especifica o grau máximo de aninhamento de um tipo. Se STR1 ou STR2 são suportadas, o valor de *nest* deve ser maior que zero.

A estruturação das unidades funcionais em CBBs facilita a especificação de conformidade, permitindo que sejam definidos subconjuntos de serviços de forma a obter uma implementação com funcionalidade bem definida e coerente.

3.1.4 O PROTOCOLO MMS

A estrutura do protocolo MMS permite que para cada aplicação seja definido um subconjunto adequado dos

serviços oferecidos. Isso torna o MMS capaz de atender às necessidades de uma ampla gama de aplicações.

O objetivo principal do padrão MMS é prover um conjunto de regras de comunicação expressas em termos de procedimentos a serem executados pelas entidades pares MMS[1] durante a comunicação.

O protocolo MMS especifica:

- Procedimentos para a transferência de dados e informações de controle entre entidades pares de aplicação no contexto MMS;
- A seleção dos serviços a serem usados pelas entidades pares de aplicação enquanto estiverem comunicando-se no contexto MMS.
- A estrutura das Unidades de Dados de Protocolo, UDPs, (PDU - *Protocol Data Unit*) MMS, utilizadas para a transferência de dados e informações de controle.

Os procedimentos especificados no MMS são definidos em termos de (Figura 3.2):

- Interação entre entidades pares da camada de aplicação (no caso, a própria implementação MMS) através de UDPs MMS;

1 Entidades pares MMS são implementações MMS executando em estações distintas.

- Interação entre o provedor MMS e o usuário MMS do mesmo sistema, através de primitivas de serviços MMS;

- Interação entre o provedor MMS e o ACSE através de primitivas dos serviços de Controle de Associação.

- Interação entre o provedor MMS e o provedor de serviços de apresentação através de primitivas dos serviços de apresentação.

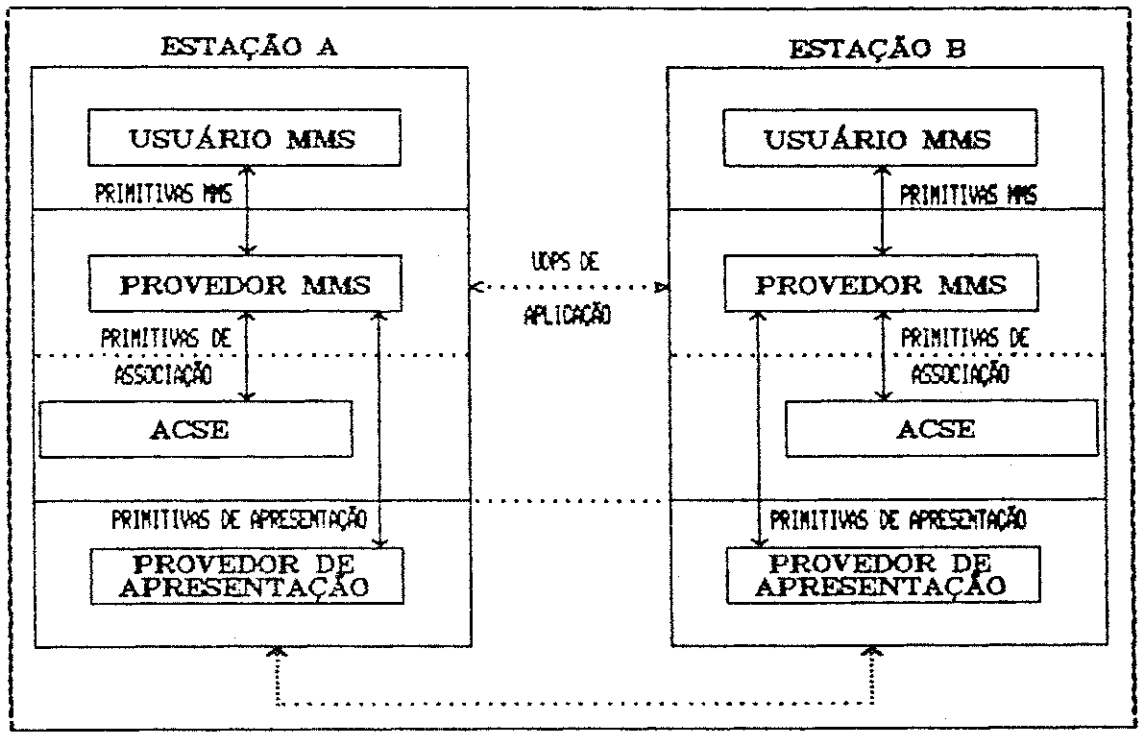


FIGURA 3.2 - COMUNICAÇÃO ENTRE ESTAÇÕES MMS

O MMS define 14 tipos de UDPs, mostrados na Tabela 3.4.

TABELA 3.4 - TIPOS DE UDPS MMS

UDP MMS	TIPO DE PRIMITIVA MAPEADA
CONFIRMED_REQUEST	primitiva *REQUEST* de um serviço confirmado.
CONFIRMED_RESPONSE	primitiva *RESPONSE* (positiva) de um serviço confirmado.
CONFIRMED_ERROR	primitiva *RESPONSE* (negativa) de um serviço confirmado.
UNCONFIRMED_REQUEST	primitiva *REQUEST* de serviço não confirmado.
REJECT	indica erro de protocolo detectado na estação remota.
CANCEL_REQUEST	primitiva *REQUEST* de cancelamento de um serviço confirmado pendente.
CANCEL_RESPONSE	primitiva *RESPONSE* (positiva) de cancelamento de serv. confirm. pendente.
CANCEL_ERROR	primitiva *RESPONSE* (negativa) de cancelamento de serv. confirm. pendente.
CONCLUDE_REQUEST	primitiva *REQUEST* de encerramento de contexto MMS.
CONCLUDE_RESPONSE	primitiva *RESPONSE* (positiva) de encerramento de contexto.
CONCLUDE_ERROR	primitiva *RESPONSE* (negativa) de encerramento de contexto.
INITIATE_REQUEST	primitiva *REQUEST* de abertura de contexto MMS.
INITIATE_RESPONSE	primitiva *RESPONSE* (positiva) de abertura de contexto MMS.
INITIATE_ERROR	primitiva *RESPONSE* (negativa) de abertura de contexto MMS.

As onze primeiras UDPs são mapeadas nas primitivas dos serviços da camada de apresentação *P_Data_Request* ou *P_Data_Indication*. As demais UDPs são mapeadas nas primitivas dos serviços do ACSE:

- UDP Initiate_Request

  Na primitiva de serviço *A_Associate_Req* ou *Ind*.

- UDP Initiate_Response

  Primitiva de serviço *A_Associate_Resp* ou *Conf*.

- UDP Initiate_Error

  Primitiva de serviço *A_Associate_Resp* ou *Conf* negativo.

A primitiva do serviço ***ABORT*** é mapeada diretamente na primitiva de serviço *A_U_Abort* do ACSE, sem ser mapeada em UDP MMS.

3.1.5 COMUNICAÇAO ENTRE ENTIDADES PARES MMS

Os procedimentos de serviço para troca de mensagens entre entidades pares MMS consistem de três etapas: estabelecimento de associação, troca de primitivas de serviços (contexto MMS) e término de associação.

As primitivas dos serviços de Gerenciamento de Contexto *Initiate*, *Conclude* e *Abort* permitem iniciar e encerrar uma associação. Uma vez estabelecido o contexto MMS, as entidades pares passam a trocar primitivas de serviço MMS, podendo haver, em um dado instante, serviços pendentes em ambas as estações.

O processo de **estabelecimento de associação** inicia-se com o recebimento de uma primitiva de serviço ou UDP *Initiate_Request*. Durante esse processo é feita uma negociação dos parâmetros que configuram a comunicação: nível de complexidade permitida nos dados trocados (simples, arranjo ou estrutura), tamanho máximo de mensagem permitido, blocos de conformidade (CBBs) verticais e horizontais suportados, número máximo de serviços pendentes permitido no

MMS chamador e no MMS chamado e versão do MMS utilizado. A negociação é feita da seguinte forma: ao receber uma UDP ou primitiva do serviço *Initiate* (*request* ou *response*), o provedor MMS pode diminuir ou manter inalterados os valores dos parâmetros recebidos, de acordo com a própria capacidade.

O provedor MMS pode manter ativas várias associações ao mesmo tempo. Cada uma possui suas próprias características, negociadas durante a abertura da associação.

Enquanto o **contexto MMS** existir, as entidades pares MMS podem trocar qualquer primitiva de serviço que esteja incluída na CBB negociada na abertura da associação. Os tipos de dados trocados, o tamanho máximo da primitiva de serviço, o número máximo de serviços pendentes e a versão MMS utilizada permitem a cada provedor limitar e controlar a comunicação nesta fase.

Ao receber uma UDP ou primitiva do serviço *Conclude_Request*, o provedor inicia a fase de **encerramento da associação**, após a qual nenhuma primitiva de serviço poderá ser trocada entre as entidades pares sem uma nova abertura de associação. Esse encerramento é dito normal. A primitiva do serviço *Abort* provoca um encerramento abrupto de contexto e, nesse caso, as solicitações de serviços ainda pendentes serão descartadas e considerados sem efeito.

O diagrama de estados da Figura 3.3 ilustra as transições envolvidas nessas três etapas.

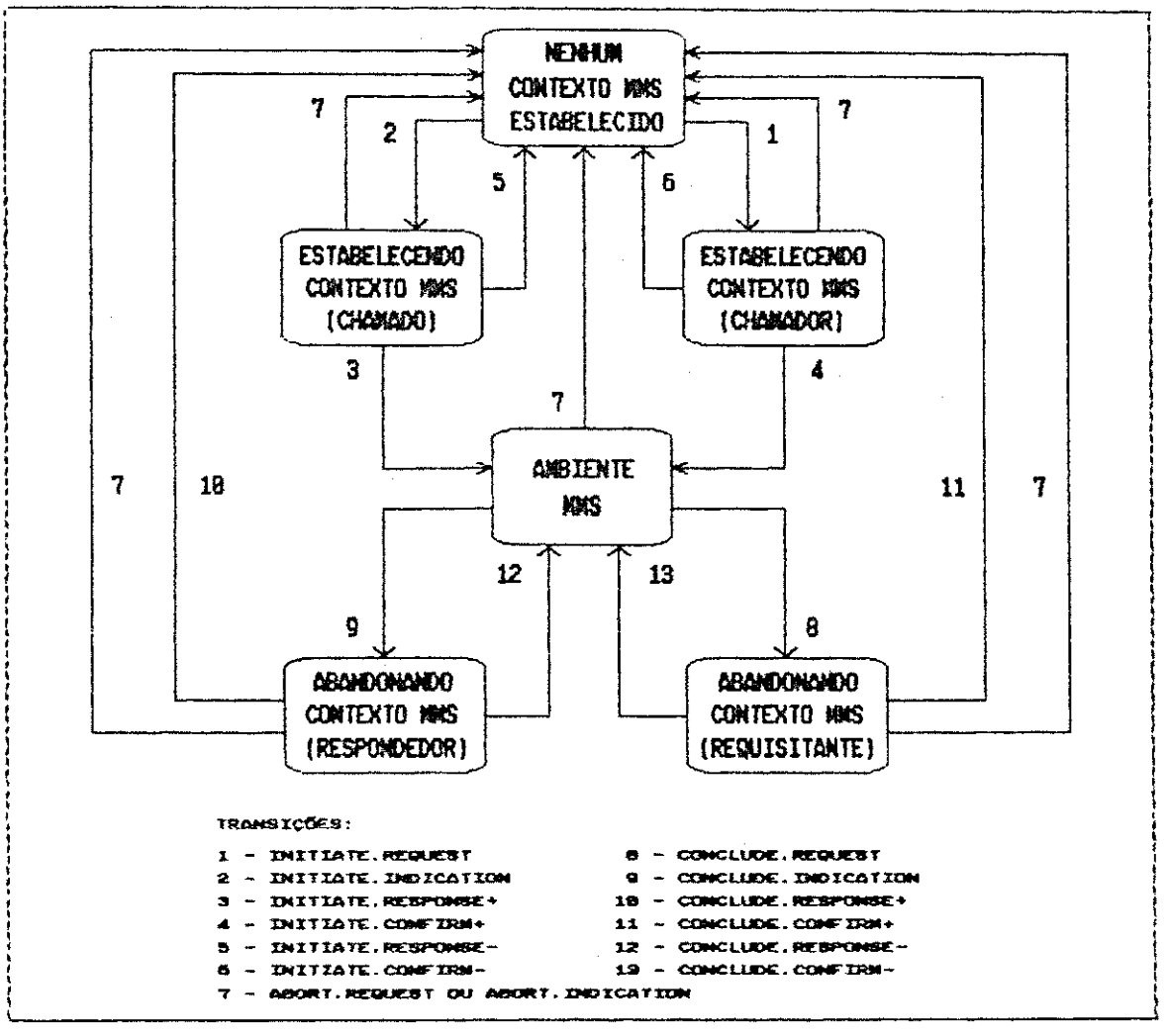


FIGURA 3.3 - DIAGRAMA DE ESTADOS MMS (GERENCIAMENTO DE CONTEXTO)

3.2 O MMS NO MINI-MAP

A especificação Mini-MAP [GM/MAP,1988] descreve várias alterações nos serviços e protocolo MMS, além de um SASE complementar para a camada de aplicação, o ORSE, que interage com o MMS e com as camadas superior e inferior.

A interface entre a camada de aplicação e a camada de enlace (LLC) também faz parte do Mini-MAP, e será discutida logo após a descrição das alterações no MMS.

3.2.1 TIPOS DE COMUNICAÇÃO

O MMS para Mini-MAP permite três tipos de comunicação: com associação, sem associação e de resposta imediata.

A comunicação com associação envolve os procedimentos de abertura e fechamento de associação descritos anteriormente, na secção 3.1.

A comunicação sem associação dispensa o estabelecimento de associação entre as entidades pares MMS para efetuar a comunicação entre elas. Entretanto, essa opção só pode ser utilizada para os seguintes serviços: *Status*, *UnsolicitedStatus*, *InformationReport*, *Identify*, *Read* e *Write*. Nesse tipo de comunicação, como não existe negociação de parâmetros, são utilizados valores *default* mantidos pelo próprio MMS.

A comunicação de resposta imediata permite a uma estação não membro do anel lógico responder a solicitações

de estações membros do anel ou solicitar destas a execução de serviços não confirmados. O anel lógico é formado pelo protocolo IEEE 802.4 (Capítulo 2) da sub-camada MAC.

A Figura 3.4 ilustra uma estação MMS completa capaz de suportar serviços de resposta imediata e as relações entre o usuário MMS (*user element*), o MMS e o ORSE.

Uma estação que participa do anel lógico possui, ativos, os SASEs MMS e ORSE. Na estação que não faz parte do anel somente o SASE MMS está ativo, apesar de existir o SASE ORSE.

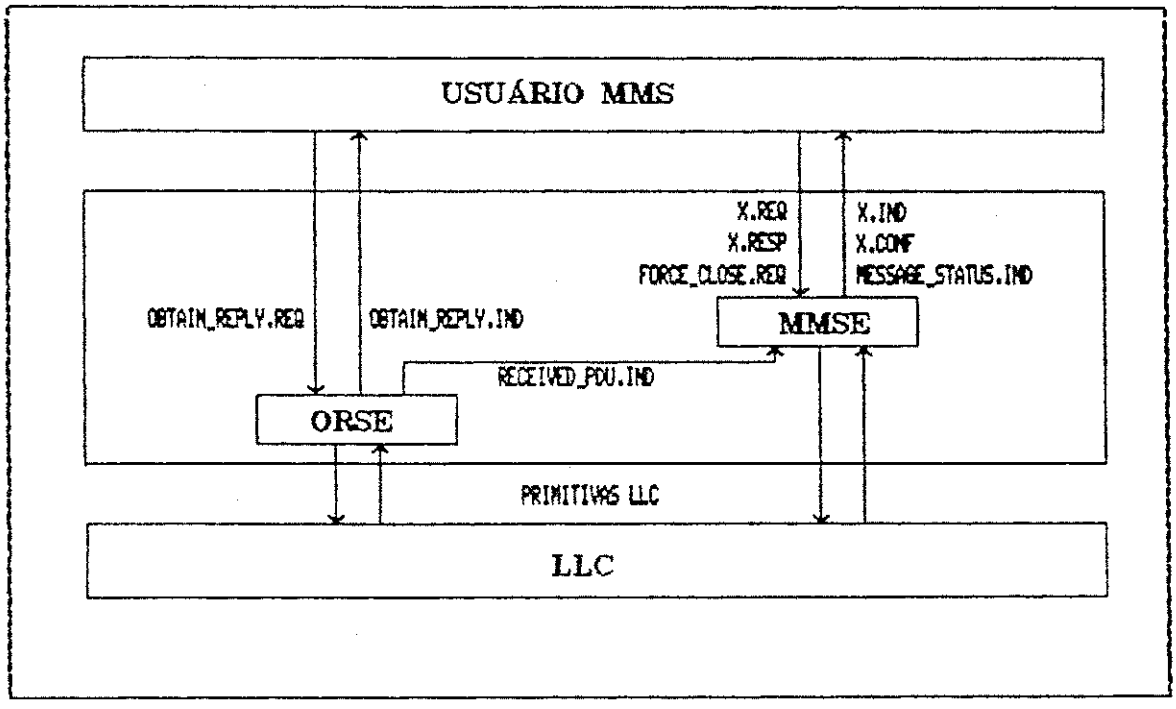


FIGURA 3.4 - ESTAÇÃO COMPLETA MMS/MINI-MAP

Os diagramas de estados das Figuras 3.5 e 3.6 ilustram como se processa uma comunicação de resposta imediata, onde a estação A participa do anel lógico e a estação B não participa.

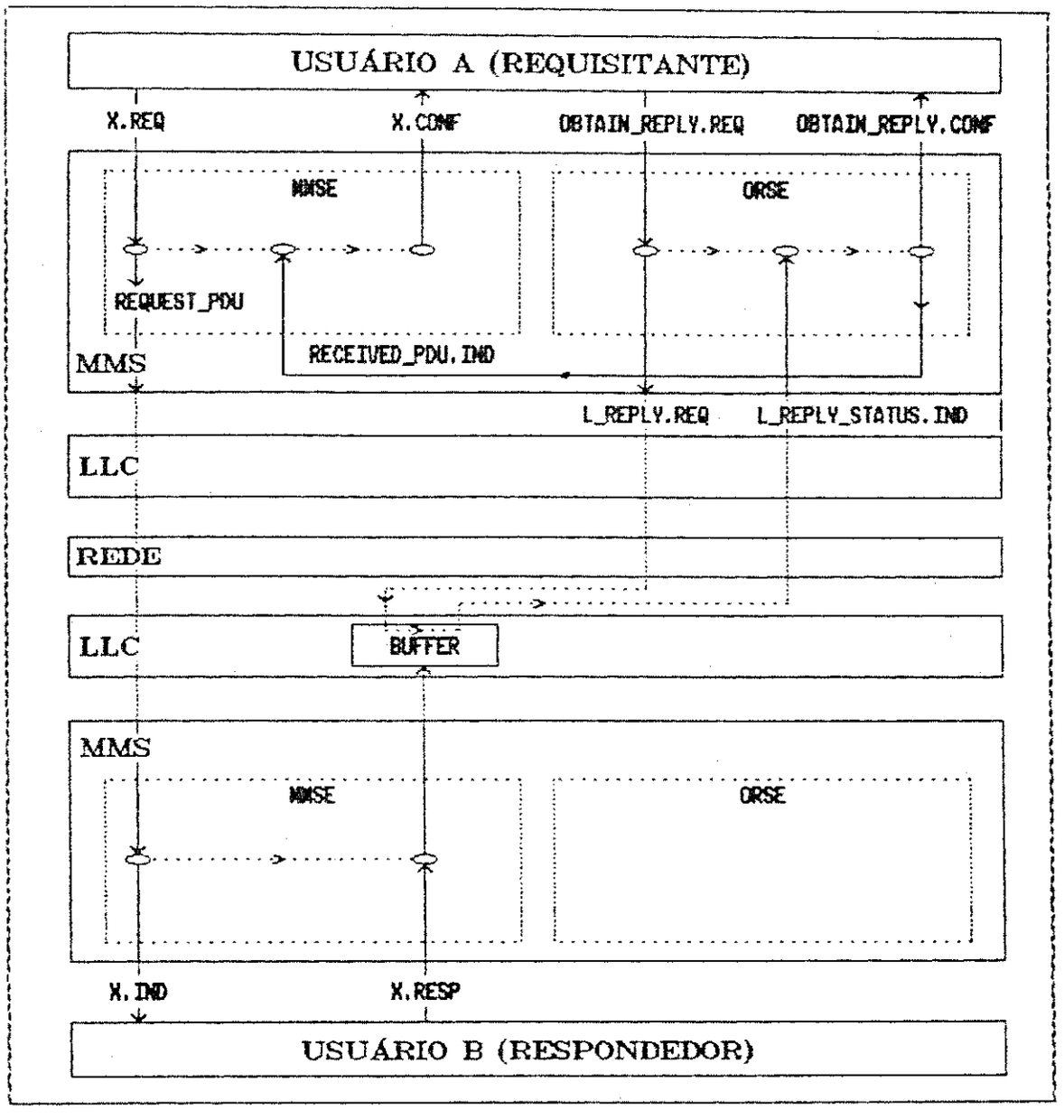


FIGURA 3.5 - DIAGRAMA DE ESTADOS DA COMUNICAÇÃO DE RESPOSTA IMEDIATA (I)

Na Figura 3.5, a estação A requisita um serviço confirmado da estação B. Como esta não participa do anel, a camada LLC_B coloca a resposta em um *buffer* e aguarda a chegada de uma *L_Reply.Request* da estação A. Para obter a *RESPONSE_PDU*, o usuário MMS_A envia uma *ObtainReply.Request*, que é mapeada na primitiva do serviço *L_Reply.Req*. Ao receber essa primitiva, a estação B envia o conteúdo do *buffer*

para a estação A. Esta então, envia uma *ObtainReply.Confirm* para o usuário e manda a resposta da estação B para o MMS, via *ReceivedPDU.Indication*.

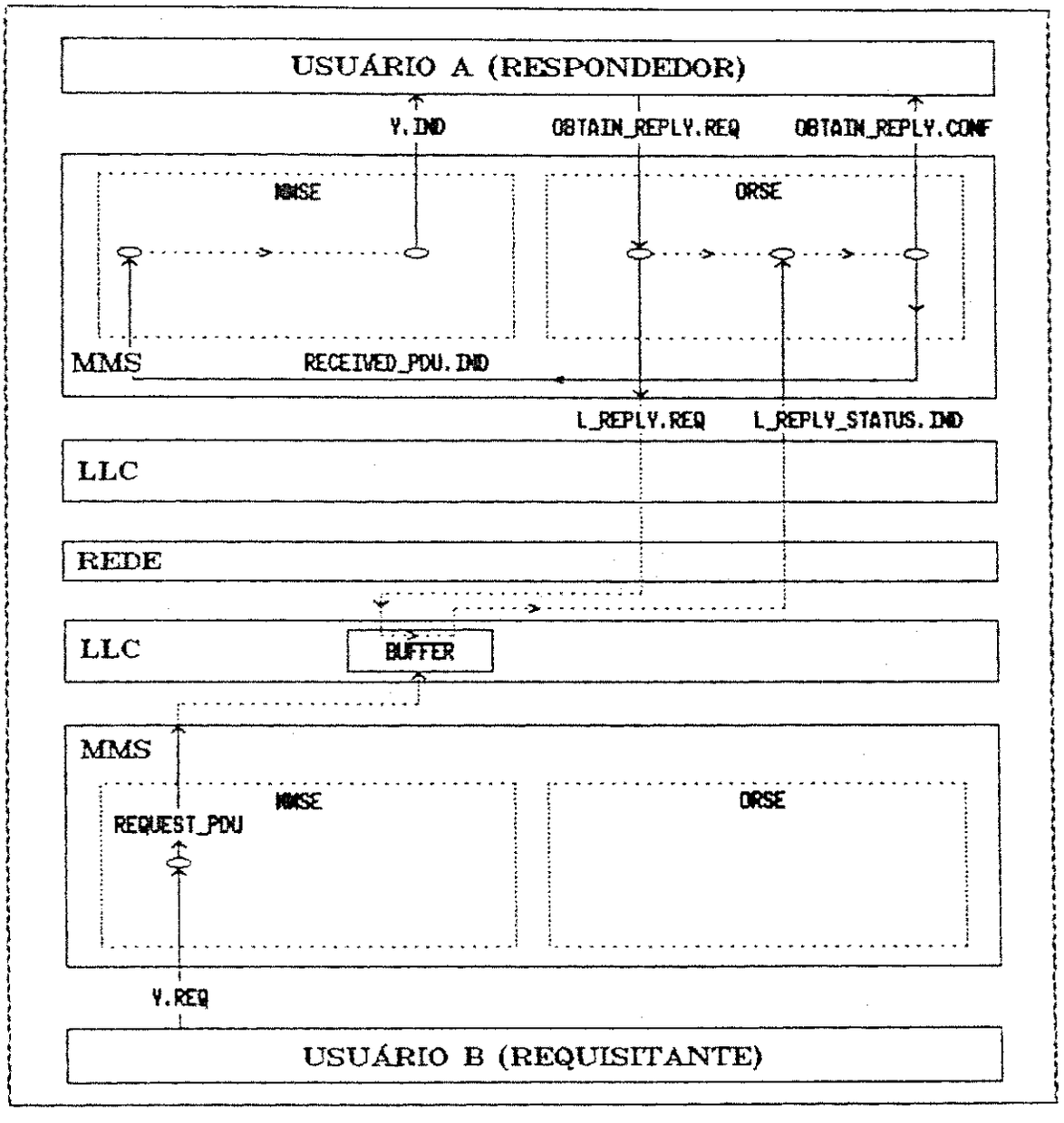


FIGURA 3.6 - DIAGRAMA DE ESTADOS DA COMUNICAÇÃO DE RESPOSTA IMEDIATA (II)

Na Figura 3.6 a estação B deseja obter um serviço da estação A. Como não participa do anel, a estação B só tem permissão para solicitar serviços não confirmados. O

procedimento do ORSEA nesse caso é o mesmo descrito para a Figura 3.5.

Esta implementação suporta os três tipos de comunicação.

3.2.2 CLASSES DE ENDEREÇAMENTO

O Mini-MAP permite ao MMS utilizar três classes de endereçamento: individual, grupo e *reply buffer*. A seleção de cada classe é feita pelo usuário MMS e enviada em primitivas de serviços *request* e *response*. Cada classe de endereçamento indicada pelo usuário gera no MMS uma primitiva LLC diferente.

O **endereçamento individual** corresponde a uma comunicação ponto-a-ponto entre entidades pares de aplicação. Como o MMS pode manter várias associações abertas, em um dado instante pode haver várias comunicações ponto-a-ponto em andamento.

No **endereçamento de grupo** o MMS difunde a mensagem na rede para um determinado grupo de entidades de aplicação. Esta classe de endereçamento não permite a utilização de serviços confirmados e só suporta comunicação sem associação.

A classe ***reply buffer*** corresponde à comunicação de resposta imediata.

Esta implementação suporta as três classes de endereçamento.

3.2.3 SERVIÇOS MMS/ORSE

O Mini-MAP só alterou serviços MMS da unidade de gerenciamento de contexto.

O serviço ***REJECT*** foi substituído pelo serviço ***REJECT_SENT***, que, assim como o ***REJECT***, só gera primitiva de serviço do tipo *indication*. Essa mudança de primitivas de serviços também introduziu alterações nos procedimentos da rejeição de UDPs [GM/MAP,1988] [EIA/MMS,1987].

O serviço ***ABORT*** sofreu algumas alterações a nível de procedimentos.

Nas mudanças efetuadas pelo Mini-MAP inclui-se, também, o acréscimo dos seguintes serviços:

- Message_Status

  Usado pelo MMS para informar ao usuário o estado do LLC (tal como: "OK" ou "sem recursos"). Só gera primitiva de serviço do tipo *indication*.

- Force_Close

  Utilizado pelo usuário MMS para cancelar o envio de primitiva de serviço *response* após um número de tentativas de envio tendo como resposta um status LLC = "sem recursos". Só gera primitiva de serviço *request*.

O ORSE especifica os seguintes serviços:

- Obtain_Reply

  Usada pelo usuário MMS para solicitar um serviço qualquer de uma estação não participante do anel lógico.

Este serviço também permite às estações que não participam do anel lógico requisitar serviços não confirmados. Gera primitivas de serviços *request* e *confirm*.

- Received_PDU

  Usada pelo ORSE para informar ao MMS o recebimento de uma UDP da estação remota. Só gera primitiva *indication*.

As máquinas de estado com o funcionamento do MMS e do ORSE no Mini-MAP são mostradas no Anexo 2.

3.2.4 PROTOCOLO MMS/ORSE

O número de UDPs MMS foi reduzido para quatro: *REQUEST_PDU*, *RESPONSE_PDU*, *ERROR_PDU* e *REJECT_PDU*.

Todos os serviços, com exceção de *REJECT_SENT*, são mapeados nas UDPs ***REQUEST***, ***RESPONSE*** e ***ERROR***.

A primitiva do serviço ***REJECT_SENT*** é mapeada na UDP ***REJECT***.

Como não existe camada de apresentação, a primitiva do serviço ***ABORT*** é mapeada em UDP do tipo ***REQUEST_PDU***.

Não são definidas UDPs para os serviços ORSE e, portanto, seu mapeamento é feito diretamente nas primitivas dos serviços da camada de enlace:

A primitiva do serviço LLC *L_Reply_Request* mapeia a primitiva do serviço ORSE *ObtainReply_Request*.

A primitiva LLC *L_ReplyStatus_Indication* transporta uma LSDU (*Link Service Data Unit*) proveniente da estação remota, contendo uma UDP MMS (*response* ou *request*).

A primitiva ORSE *ReceivedPDU_Indication* é enviada diretamente ao MMS.

3.2.5 A INTERFACE ENTRE MMS E LLC

Na interface LLC/MMS deve haver, no mínimo, um ponto de acesso para cada tipo de comunicação [GM/MAP,1988]. Portanto, o MMS deve especificar pelo menos três pontos de acesso de serviço ao LLC (LSAP = *Link Service Access Point*).

O Mini-MAP prevê, também, a existência de um gerente de estação. Esse gerente indica ao MMS a ocorrência de falhas na estação que impossibilitem a continuidade na comunicação com outras estações. Nesse caso, o MMS avisa ao usuário e encerra todas as associações abertas. As primitivas dos serviços do gerente da estação também devem transitar por um ponto de acesso independente dos demais.

Cabe ao MMS verificar se os serviços com associação e sem associação estão passando pelos LSAPs corretos, e isso é feito apenas na chegada de LSDUs. As máquinas de estado descritas no Mini-MAP não efetuam qualquer verificação da validade do LSAP através do qual o MMS recebe LSDUs para serviços de resposta imediata e primitivas dos serviços do gerente de estação. Mesmo assim, sempre que se

desejar esse tipo de serviço, deve-se especificar o LSAP de destino apropriado.

O fato de só haver conferência entre o tipo de comunicação e o LSAP na chegada de uma LSDU, levou à definição, nesta implementação, da interface mostrada na Figura 3.7. A estrutura lógica da interface consiste de um LSAP no sentido MMS-LLC e quatro LSAPs no sentido LLC-MMS e optou-se por definir a estrutura física da mesma forma. A definição da estrutura física com um único ponto de entrada (sentido LLC-MMS) tornaria necessária a inclusão de um campo de identificação do LSAP referenciado na LSDU.

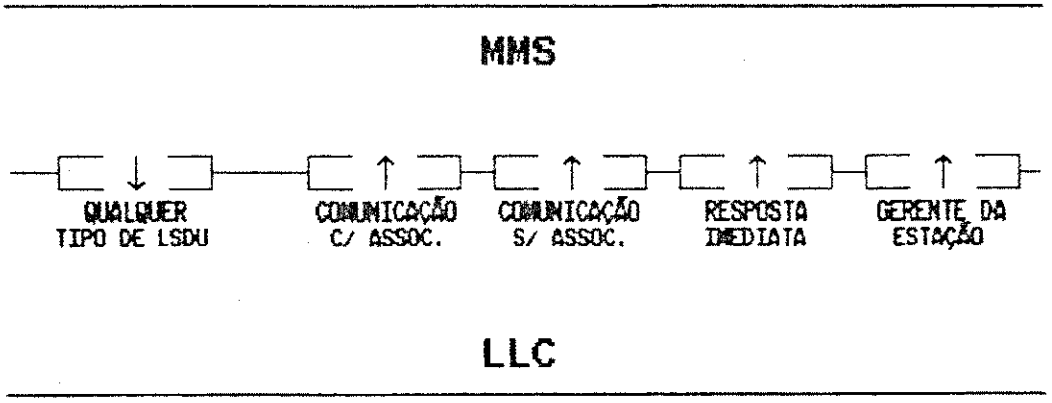


FIGURA 3.7 - INTERFACE MMS/LLC

CAPITULO 4

ESPECIFICAÇÃO DE MENSAGENS

EM

CONTROLADORES LÓGICOS

PROGRAMAVEIS

A implementação de um servidor MMS deve efetuar o mapeamento entre o modelo VMD e a funcionalidade do dispositivo real. Esse mapeamento é descrito em documentos denominados *Companion Standards*. Cada *companion standard* descreve o mapeamento particular de determinado dispositivo.

Neste capítulo é feita uma descrição sucinta da especificação IEC [IEC/PCMS,1989] para controladores lógicos programáveis, abordando os aspectos físicos e funcionais do CLP, bem como o mapeamento entre funções CLP e serviços MMS.

Nesta implementação são definidos os serviços MMS necessários ao suporte das funções CLP até a classe 3. Entretanto, a estrutura do programa permite a extensão para as classes 4 e/ou 5 do CLP com mudanças apenas nas estruturas de dados.

4.1 ESCOPO DA ESPECIFICAÇÃO

O *companion standard* definido pelo IEC aplica-se à comunicação entre CLPs e outros dispositivos que utilizam o padrão MMS [EIA/MMS,1987].

No documento são especificadas as normas de utilização do MMS pelo cliente para requisitar serviços CLP a serem executados pelo servidor.

O cliente pode ser um CLP ou qualquer outro dispositivo. Entretanto, apenas o cliente CLP está no escopo da especificação. O servidor pode ser:

- um CLP que segue as especificações do padrão IEC [IEC/PCMS,1989];
- um CLP que não segue essas especificações;
- um sistema que executa funções do CLP, mas pertence a um dispositivo que não é um CLP.

Todos os três tipos de dispositivo estão no escopo do *companion standard*.

Independentemente dos dispositivos envolvidos, a comunicação MMS que não esteja no contexto do CLP está fora do escopo do *companion standard*. Da mesma forma, qualquer comunicação não MMS entre dispositivos está fora do escopo do documento.

A Figura 4.1 ilustra uma rede onde um CLP servidor comunica-se com três tipos de clientes. Os dispositivos

representados em negrito estão no escopo do *companion standard* IEC [IEC/PCMS,1989].

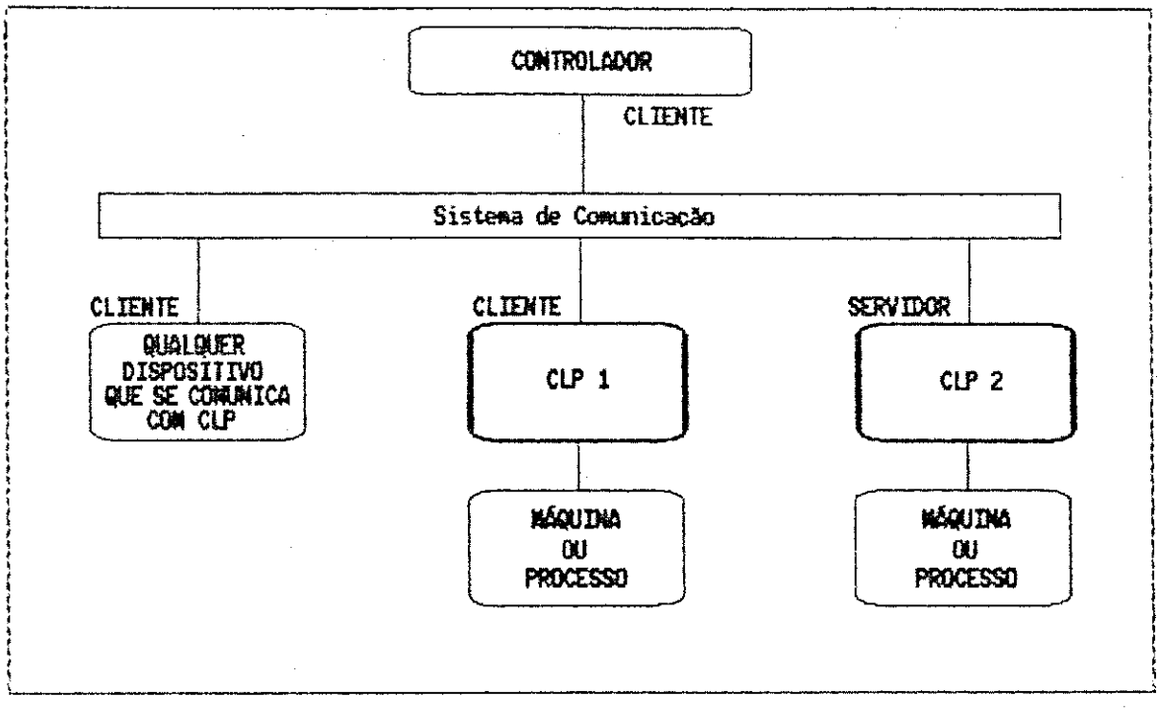


FIGURA 4.1 - O CLP EM UMA REDE INDUSTRIAL

4.2 ARQUITETURA DO CLP IEC

O CLP é um dispositivo aplicado em automação que requisita serviços de outros CLPs e executa funções para outros dispositivos. Essas funções serão descritas mais adiante.

O diagrama da Figura 4.2 mostra a estrutura do CLP IEC e os tipos de subsistemas que podem ser suportados. Dependendo do fabricante ou da aplicação, pode-se ter múltiplos subsistemas do mesmo tipo em um CLP: vários módulos de memória, vários módulos de E/S, várias unidades de processamento, e assim por diante. Os subsistemas do mesmo tipo são ditos similares. Os subsistemas similares são tratados de forma separada, sendo possível atribuir-lhes nomes para a diferenciação de cada um.

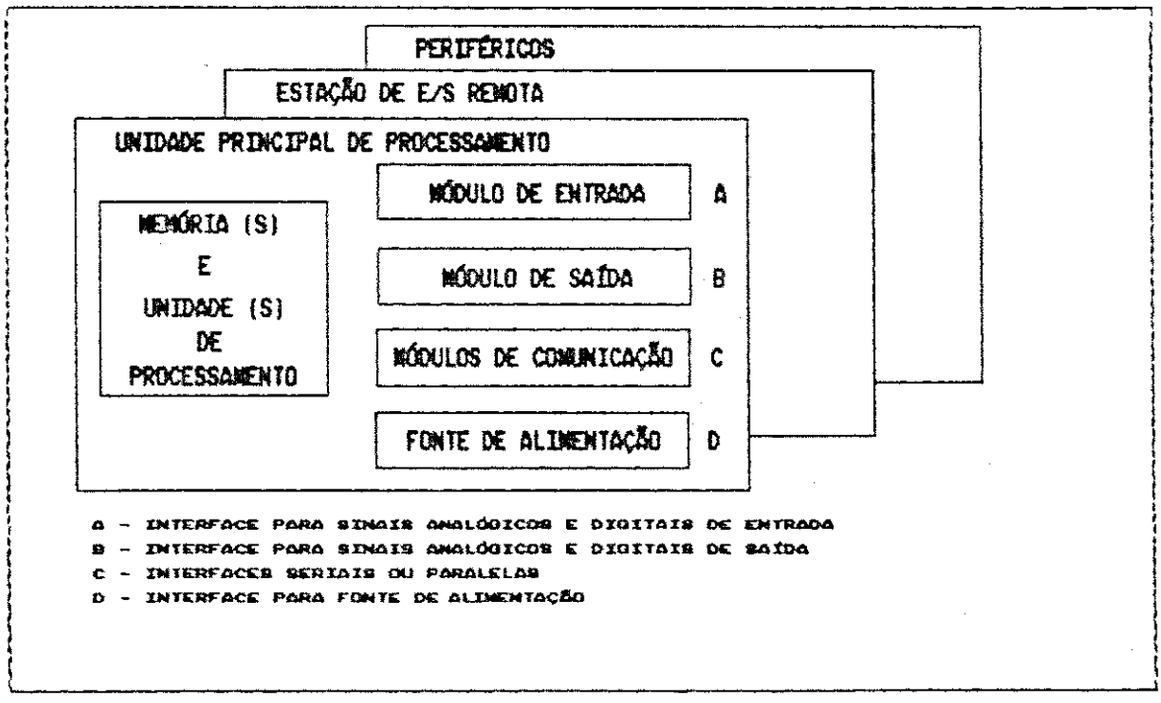


FIGURA 4.2 - ESTRUTURA DO CLP IEC

4.3 FUNCIONALIDADE DO CLP

O CLP efetua diversas funções para o sistema de manufatura. Essas funções são descritas a seguir, com a ressalva de que nem todas são disponíveis em todos os CLPs. São elas:

- Verificação de dispositivo;
- Aquisição de dados;
- Controle;
- Sincronização;
- Alarme;
- Interface com operador;
- Controle e gerenciamento de programa.

4.3.1 VERIFICAÇÃO DE DISPOSITIVO

Permite a um dispositivo verificar se o CLP está apto a executar a função da qual necessita. O CLP servidor envia as informações de verificação para o cliente como resposta a uma requisição feita pelo mesmo, ou espontaneamente, sem prévia solicitação.

O resultado da verificação indica o *status* do CLP e abrange três tópicos:

- informação de estado;
- detecção de falhas;
- isolamento de falhas.

Esses dois últimos tópicos informam a confiabilidade atual do CLP.

A informação de *status* é composta de um sumário de *status* do CLP e de uma lista com o *status* de cada subsistema. O **sumário de *status*** indica a confiabilidade e o estado do CLP de forma geral. A **lista** contém a identificação, a confiabilidade e o estado de cada subsistema, além de informações específicas do fabricante.

Os subsistemas que provêm informações de *status* são: módulos de E/S, unidades de processamento, fontes de alimentação, memória e módulos de comunicação.

O *status* refere-se aos subsistemas de *hardware* e *firmware*, não considerando as informações de configuração e de *software*.

4.3.2 AQUISIÇÃO DE DADOS

Os dados armazenados em um CLP podem ser provenientes de outros dispositivos (dados de processos, máquinas, inventários) ou de resultados de cálculos de programas internos do CLP. O cliente obtém os dados através de *polling*, de forma não solicitada programada ou de forma não solicitada configurada. Essas formas de obtenção de dados são descritas a seguir:

Polled

O cliente lê o valor de uma ou mais variáveis. Esse acesso pode ser controlado pelo CLP.

Aquisição não Solicitada Programada

A unidade de processamento do CLP é programada para enviar mensagens contendo valores de variáveis de processo para o cliente, sem prévia solicitação deste.

Aquisição não Solicitada Configurada

A interface de comunicação do CLP é configurada pelo cliente para enviar mensagens, sem prévia solicitação deste.

4.3.3 CONTROLE

O cliente controla a operação do CLP por dois métodos: controle paramétrico e controle por travamento. A descrição de cada um é dada a seguir:

Paramétrico

Através do programa de aplicação ou de outros mecanismos, o cliente direciona a operação do CLP atribuindo valores às variáveis de controle nele residentes.

Travamento

O cliente solicita ao servidor que execute uma determinada operação e o informe dos resultados. Há dois aspectos relativos a este serviço: a sincronização dos programas de aplicação do cliente e do servidor e a troca de dados entre eles. Essa troca de dados ocorre nos pontos de sincronização entre os programas de aplicação.

Este serviço pode ser usado por um programa de aplicação para disparar a execução de um programa de aplicação remoto.

4.3.4 SINCRONIZAÇÃO

Usuários pares de aplicação podem sincronizar-se através do uso de semáforos. O nome e o conjunto de regras para o uso do semáforo são negociadas entre os usuários.

4.3.5 ALARME

Alarmes são mensagens enviadas do CLP para o cliente quando ocorre determinado evento.

Após a recepção da mensagem de alarme o cliente envia uma mensagem de reconhecimento para o CLP. Enquanto o reconhecimento não chega, o CLP continua indicando a condição de alarme.

Um item opcional para esse serviço é a geração de um sumário de alarme por parte do servidor, após solicitação do cliente. Esse sumário consiste de um relatório com estatísticas de todos os alarmes já reconhecidos pelo cliente.

Uma mensagem de alarme é iniciada de maneira programada ou configurada, conforme descrito a seguir:

Programada

A unidade de processamento do CLP é programada para enviar ao cliente mensagens de alarme contendo os valores de interesse.

Configurada

A interface de comunicação do CLP é configurada para enviar mensagens de alarme para o cliente.

4.3.6 INTERFACE COM OPERADOR

Os dispositivos de interface com operador são usados pelo operador para monitorar e/ou modificar os processos controlados. A operação da interface é controlada através de programação da unidade de processamento do CLP.

A interface pode também ser usada pelo cliente para se comunicar com o operador. Para isso, o cliente deve usar os mecanismos de controle descritos na sub-secção CONTROLE, acima. Esses mecanismos possibilitam ao cliente interagir com o programa de aplicação ou com a estrutura de controle interna do CLP responsável pela interface.

4.3.7 CONTROLE E GERENCIAMENTO DE PROGRAMA

Inclui as seguintes funções:

- relatório do estado do programa de aplicação do CLP;

- controle de execução do programa de aplicação do CLP;
- transferência do programa de aplicação.

O relatório de estado do programa de aplicação inclui informações sobre seu estado físico e seu estado lógico. Os estados físicos podem ser:

- *non-existent*;
- *loading*;
- *complete*;
- *ready*;
- *in-use*;
- *incomplete*.

Os estados lógicos podem ser:

- *idle*;
- *starting*;
- *running*;
- *stopping*;
- *resetting*;
- *stopped*;
- *resuming*;
- *unrunnable*.

O sistema de controle de execução permite ao cliente mudar o estado lógico do programa de aplicação. As-

sim, pode-se iniciar, retomar, reiniciar, terminar, paralisar ou interromper a execução de um programa de aplicação.

O serviço de transferência do programa de aplicação permite ao cliente carregar (*upload*) o conteúdo da memória programável do CLP para arquivamento ou verificação e retornar (*download*) esse conteúdo para o CLP. A memória do CLP deve conter um ou mais dos seguintes itens:

- programas executáveis;
- valores iniciais de variáveis;
- tabela de símbolos;
- configuração do sistema;
- dados específicos do fabricante.

4.4 MAPEAMENTO DAS FUNÇÕES CLP PARA SERVIÇOS MMS

O IEC define cinco classes de conformidade para os serviços de mensagens do CLP [IEC/PCMS,1989], de forma a atender a dispositivos com diferentes graus de complexidade (dos mais simples aos mais complexos). A Tabela 4.1 mostra as classes de conformidade e as funções do CLP providas em cada uma.

TABELA 4.1 - CLASSES DE CONFORMIDADE DO CLP

CLASSE CLP	FUNÇÕES PROVIDAS POR CLASSE
1	VERIFICAÇÃO DE DISPOSITIVO SOLICITADA AQUISIÇÃO DE DADOS TIPO *POLLED* CONTROLE PARAMÉTRICO
2	GERENCIAMENTO DE PROGRAMA
3	VERIFICAÇÃO DE DISPOSITIVO NÃO SOLICITADA AQUISIÇÃO DE DADOS PROGRAMADA CONTROLE POR TRAVAMENTO
4	AQUISIÇÃO DE DADOS CONFIGURADA ALARME PROGRAMADO SEMÁFOROS
5	ALARME CONFIGURADO

As classes estão estruturadas de modo que cada classe englobe os serviços da classe anterior. Assim, a classe 2 inclui, além da função de Gerenciamento de Programa, as funções de Aquisição de Dados Tipo *Polled*, Verificação de Dispositivo Solicitada e Controle Paramétrico (definidas para a classe 1); e assim por diante.

A Tabela 4.2 indica os serviços MMS requeridos para atender a cada tipo de serviço do CLP.

TABELA 4.2 - MAPEAMENTO ENTRE SERVIÇOS CLP E SERVIÇOS MMS

CLASSE	FUNÇÕES PROVIDAS POR CLASSE	SERVIÇOS MMS
1	VERIF. DE DISP. SOLICITADA AQUIS.. DE DADOS TIPO *POLLED* CONTROLE PARAMÉTRICO	Status Read Write
2	GERENCIAMENTO DE PROGRAMA	Initiate Download Sequence, Terminate Download Sequence, Download Segment, Initiate Upload Sequence,Terminate Upload Sequence, Upload Segment, Delete Domain, Get Domain Attributes, Get Name List, Create Program Invocation, Delete Program Invocation, Get Program Invocation Attributes, Start, Stop, Resume, Reset
3	VERIF. DISP. NÃO SOLICITADA AQUIS. DE DADOS PROGRAMADA CONTROLE POR TRAVAMENTO	Status, Unsolicited Status Information Report, Unsolicited Status Get Name List, Data Exchange
4	AQUIS. DE DADOS CONFIGURADA	Get Name List, Unsolicited Status, Read, Event Notification, Report Event Condition Status, Alter Event Condition Monitoring, Acknowledge Event Notification, Report Event Action Status, Report Event Enrollment Status
	ALARME PROGRAMADO	Event Notification, Acknowledge Event Notification, Get Alarm Summary
	SEMAFOROS	Get Name List, Cancel, Take Control, Relinquish Control, Report Semaphore Status, Report Semaphore Entry Status
5	ALARME CONFIGURADO	Get Name List, Define Event Condition, Delete Event Condition, Get Event Condition Attributes, Report Event Condition Status, Alter Event Condition Monitoring, Event Notification, Acknowledge Event Notification, Define Event Action, Delete Event Action, Get Event Action Attributes, Report Event Action Status, Define Event Enrollment, Delete Event Enrollment, Alter Event Enrollment, Get Event Enrollment Attributes, Report Event Enrollment Status, Get Alarm Summary
	AQUIS. DE DADOS CONFIGURADA	Define Event Condition, Delete Event Condition, Get Event Condition Attributes, Define Event Action, Delete Event Action, Get Event Action Attributes, Define Event Enrollment, Delete Event Enrollment, Get Event Enrollment Attributes, Alter Event Enrollment,
	SINCRONIZAÇÃO	Define Semaphore, Delete Semaphore

Na tabela não são mostrados os serviços obrigatórios MMS definidos pelo *companion standard*. Esses serviços devem ser suportados por todos os CLPs que operam em um ambiente MMS. São eles: *Initiate, Conclude, Abort, Reject* e *Identify*.

Como já foi dito anteriormente, o cliente tem acesso à interface com o operador através dos serviços de controle, tanto paramétrico como por travamento.

O serviço MMS *DataExchange* foi definido pelo *companion standard* para mapear comandos CLP de troca de dados (*Send* e *Receive*), usados no controle por travamento.

Em termos de unidades funcionais MMS, o CLP pode suportar:

- Gerenciamento de Contexto (completa);
- Suporte ao VMD (CBBs MMS2 e VMD1);
- Acesso a Variáveis (CBBs VAR2 a VAR6);
- Gerenciamento de Domínio (CBBs DOM1, DOM2, DOM7 e DOM8);
- Gerenciamento de Programa (CBBS PRG1 a PRG5 e PRG7);
- Gerenciamento de Eventos (CBBs EVN1 a EVN3 e EVN5);
- Gerenciamento de Semáforos (SEM1 e SEM3, ambas incompletas).

Os CBBs respectivos de cada unidade estão distribuídos nas classes funcionais do CLP. Nem sempre uma classe suporta um CBB completo, sendo o restante dos serviços do CBB definido na classe seguinte.

As CBBs horizontais MMS definidas no *companion standard* são: NEST, STR1, STR2, VNAM, VADR, VALT. A classe 3 CLP suporta apenas as cinco primeiras CBBs.

Algumas primitivas MMS definem um campo opcional, denominado *Companion Standard Detail*, que deve ser definido por cada *companion standard*. Nesses campos são introduzidos parâmetros específicos do dispositivo representado pelo MMS. No caso do CLP, as primitivas de serviço MMS alteradas são:

- *Unsolicited Status*;
- *Status*;
- *Initiate Download Sequence*;
- *Initiate Upload Sequence*;
- *Get Domain Attribute*;
- *Create Program Invocation*;
- *Start*;
- *Stop*;
- *Resume*;
- *Get Program Invocation Attribute*;
- *Event Notification*.

CAPITULO 5

A IMPLEMENTAÇÃO DO MMS

Esta implementação MMS fornece serviços que permitem a uma aplicação usuária do MMS ter acesso às funções de um CLP servidor classe 3. As unidades funcionais MMS implementadas que atendem aos requisitos de um CLP classe 3 são:

- Gerenciamento de Contexto (CBBs CON1, CON2 e MMS1);
- Acesso a Variáveis (CBBs VAR2 a VAR4);
- Suporte ao VMD (CBBs VMD1 e MMS2);
- Gerenciamento de Domínio (CBBs DOM1, DOM2, DOM7 e DOM8);
- Gerenciamento de Programa (CBBs PRG1 a PRG5 e PRG7).

Além dessas unidades funcionais, a implementação suporta um serviço introduzido pelo *companion standard* do CLP: *DATA_EXCHANGE*.

A implementação suporta também, de acordo com o *companion standard* do CLP, as CBBs horizontais necessárias à classe 3: NEST, STR1, STR2, VNAM e VADR.

Como a implementação é direcionada para o Mini-MAP, as adaptações introduzidas por esta arquitetura no MMS foram seguidas aqui:

- **Máquinas de protocolo implementadas:** MMPM (do SASE MMS) e ORPM (do SASE ORSE);

- **Tipos de comunicação suportados:** com associação, sem associação e resposta imediata;
- **Classes de endereçamento suportadas:** individual, grupo e *reply buffer*;
- **Serviços MMS implementados:** além dos especificados pelas CBBs verticais acima, foram implementados também os serviços *REJECT_SENT*, em substituição a *REJECT*, *MESSAGE_STATUS* e *FORCE_CLOSE*;
- **Serviços ORSE implementados:** *OBTAIN_REPLY* e *RECEIVED_PDU*;
- **Tipos de UDPs MMS:** *REQUEST_PDU*, *RESPONSE_PDU*, *ERROR_PDU* e *REJECT_PDU*:
- **Pontos de acesso de serviço na interface MMS/LLC:** um ponto de acesso no sentido MMS/LLC e quatro pontos de acesso no sentido LLC/MMS. Esses pontos de acesso são usados na recepção de primitivas de serviço da seguinte forma: um ponto de acesso para comunicação com associação, um ponto de acesso para comunicação sem associação, um ponto de acesso para comunicação de resposta imediata e um um ponto de acesso para recepção de primitivas do gerente da estação.

O modelo definido para esta implementação do MMS inclui a especificação dos seguintes elementos:

- processos envolvidos no funcionamento do protocolo;
- tipo de interface com as camadas superior (usuário MMS) e inferior (LLC);

- tipos e número de estruturas de dados necessário à manutenção da máquina de protocolo do MMS (MMPM = *Manufacturing Message Protocol Machine*) e da máquina de protocolo do ORSE (ORPM = *Obtain Reply Protocol Machine*);
- implementação das primitivas e UDPs definidas na sintaxe ASN.1 [ISO/ASN1,1986] na linguagem de programação utilizada (linguagem C).

Na Figura 5.1 pode-se ter uma visão geral de como os processos, as interfaces superior e inferior e as estruturas de dados se relacionam na implementação.

A implementação MMS proposta possui uma definição de estrutura de dados que permite posteriores extensões para o suporte das classes 4 e 5 do CLP.

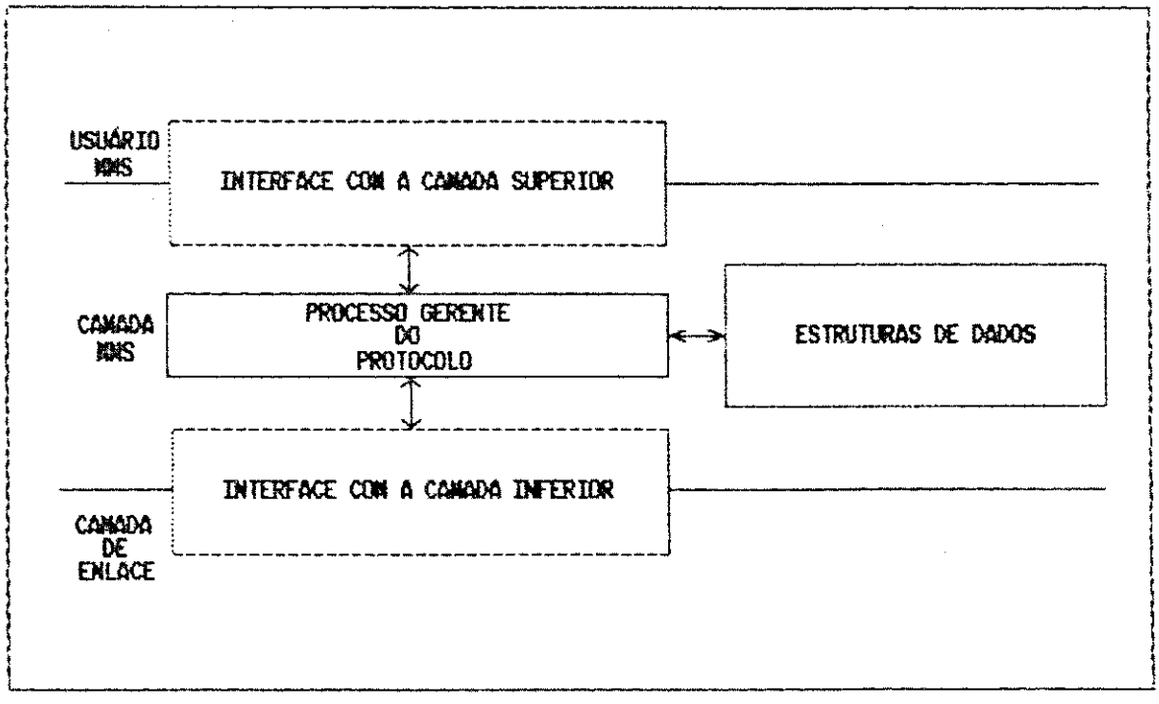


FIGURA 5.1 - INTERAÇÃO ENTRE OS ELEMENTOS DEFINIDOS NA IMPLEMENTAÇÃO

5.1 PROCESSOS

Na nossa solução o protocolo é implementado como uma entidade mono-processo que se encarrega de todas as funções de gerenciamento da camada. Este processo é composto de um módulo geral e dois módulos específicos:

- módulo MMPM, que efetua as funções relativas à máquina de protocolo do MMS;
- módulo ORPM, que efetua as funções referentes à máquina de protocolo do ORSE.

O **módulo geral** encarrega-se de efetuar o controle e o acesso às interfaces superior e inferior, gerar as estruturas de dados, verificar o tipo da mensagem recebida e acionar os dois outros módulos quando necessário.

O **módulo MMPM** trata das primitivas MMS recebidas do usuário local, das primitivas ORSE recebidas do módulo ORPM local e das UDPs MMS recebidas de estações remotas. Esse tratamento inclui:

- Verificação de sintaxe e semântica das UDPs;
- Manutenção e acesso às estruturas de dados da MMPM;
- Início e término normal ou abrupto de associações;
- Invocação de primitivas MMS e LLC;
- Montagem e desmontagem de UDPs;
- Acesso à interface com a camada inferior.

O **módulo ORPM** trata das primitivas ORSE, recebidas do usuário local, e das primitivas LLC, referentes aos

serviços de resposta imediata, recebidas de estações remotas. Esse tratamento inclui:

- Manutenção e acesso às estruturas de dados da ORPM;
- Invocação de primitivas ORSE e LLC;
- Acesso à interface com a camada inferior.

A organização das estruturas de dados permite a esses dois módulos gerenciar informações referentes a vários serviços pendentes numa mesma associação e a várias associações abertas num dado instante.

O funcionamento geral do processo, ilustrado no fluxograma da Figura 5.2, é simples, baseado numa amostragem cíclica das interfaces superior e inferior.

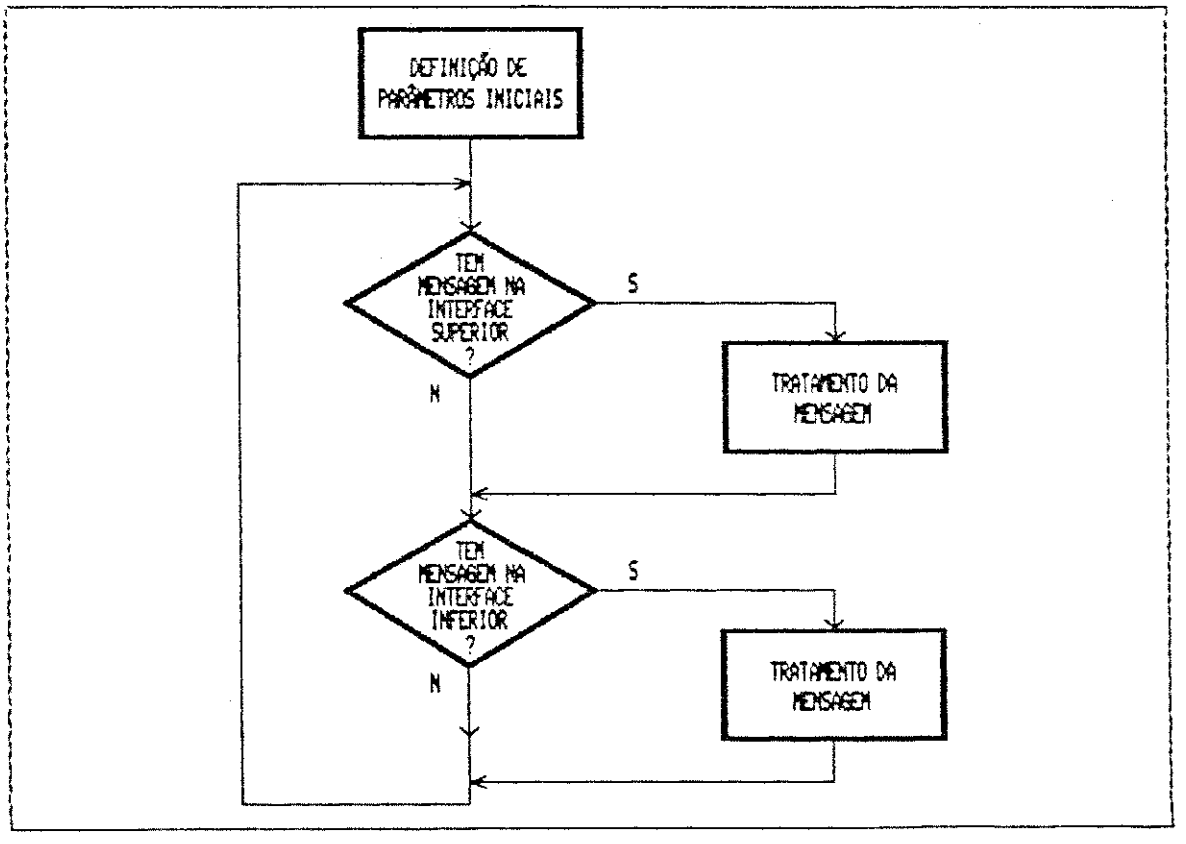


FIGURA 5.2 - FUNCIONAMENTO GERAL DO PROCESSO MMS

Optou-se por esta estrutura tendo em vista que uma estrutura multi-processos poderia degradar o desempenho do protocolo. Essa degradação seria provocada pela utilização dos recursos de memória compartilhada e de filas de mensagens, cujo acesso é lento, para prover a troca de mensagens e parâmetros entre os processos eventualmente envolvidos.

5.2 INTERFACE COM AS CAMADAS ADJACENTES

Essa interface possibilita a troca de mensagens através de filas entre o processo e cada camada adjacente (superior e inferior). A mensagem pode ser uma primitiva MMS, uma primitiva ORSE (ambas recebidas pela interface superior), uma primitiva LLC ou uma primitiva do gerente da estação (ambas recebidas pela interface inferior).

O esquema da Figura 5.3 ilustra a interação do processo MMS com as filas.

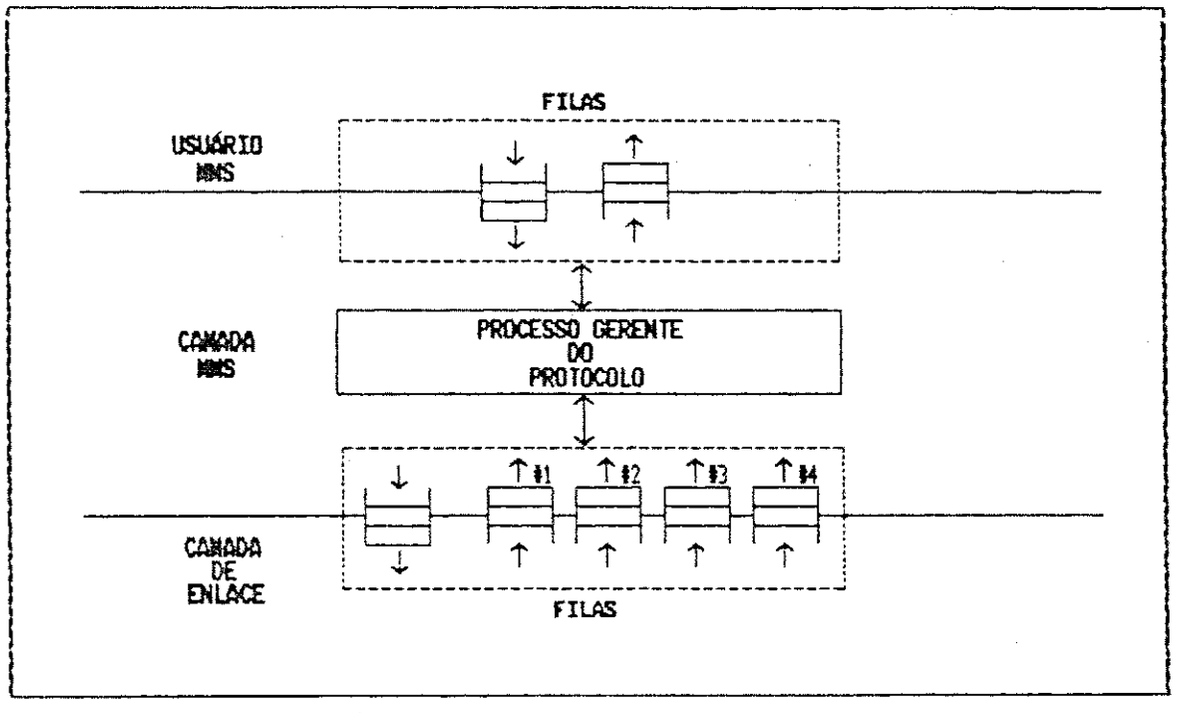


FIGURA 5.3 - FILAS USADAS PARA ACESSO ÀS INTERFACES SUPERIOR E INFERIOR

Na interface superior utiliza-se duas filas de mensagens, descritas a seguir:

Fila de Entrada Superior

Utilizada para receber primitivas MMS (*request* e *response*) e primitivas ORSE (*request*) provenientes do usuário MMS local.

Fila de Saída Superior

Utilizada para enviar primitivas MMS (*indication* e *confirm*) e primitivas ORSE (*confirm*) para o usuário MMS local.

Na interface inferior utiliza-se cinco filas de mensagens, onde cada fila corresponde a um ponto de acesso LLC (LSAP). Essas filas são descritas a seguir:

Fila de Saída Inferior

Utilizada para o envio de primitivas LLC (*request*) contendo UDPs MMS ou primitivas ORSE para a estação remota.

Fila de Entrada Inferior #1 - Comunicação com Associação

Utilizada para recepção de primitivas LLC (*indication*) contendo UDPs MMS enviadas pela estação remota, primitivas essas tratadas pelo módulo MMPM. Esta fila é exclusiva para serviços que utilizem comunicação com associação.

Fila de Entrada Inferior #2 - Comunicação sem Associação

Utilizada para recepção de primitivas LLC (*indication*) contendo UDPs MMS enviadas pela estação remota. Por esta fila só se permite a passagem de serviços que utilizem comunicação sem associação. Esses serviços são tratados pelo módulo MMPM.

Fila de Entrada Inferior #3 - Resposta Imediata

Utilizada para recepção de um tipo de primitiva LLC (*indication*) específico do serviço de resposta imediata. Todas as primitivas LLC recebidas através desta fila são tratadas pelo módulo ORPM.

Fila de Entrada Inferior #4 - Gerente da Estação

Utilizada para recepção de primitivas do gerente da estação, as quais são tratadas pelo módulo MMPM. Por enquanto, só está prevista a recepção de um tipo de primitiva, a que indica ocorrência de falha na estação [GM/MAP,1988].

O acesso de cada módulo às filas superiores e inferiores é feito através da disciplina de serviço FIFO (*First-In-First-Out*).

5.3 ESTRUTURAS DE DADOS

O modelo definido para a implementação das estruturas de dados baseou-se nas propostas encontradas em [HALSALL,1988], [MARTINS,1990] e [LIMA,1990]. Essas propostas sugerem estruturas de dados semelhantes, as quais são especificadas de forma a desencadear ações de saída como conseqüência da ocorrência de um evento. Isso é feito através de pesquisas seqüenciais em tabelas, utilizando um mínimo de código de programação.

Apesar da existência de duas máquinas de protocolo envolvidas na implementação, não foi necessária a definição de estruturas de dados diferentes, exceto a Tabela de Estados e a Lista de Reconhecimentos Pendentes, que foram definidas em separado para cada máquina de protocolo. A Figura 5.4 mostra o relacionamento entre as máquinas de protocolo MMPM e ORPM e as estruturas de dados definidas.

Cada máquina de protocolo controla e tem acesso a três tabelas principais. A MMPM mantém a Tabela de Estados MMPM, a Tabela de Transição de Estados e a Tabela de Funções de Saída. A ORPM mantém a Tabela de Estados ORPM, a Tabela de Transição de Estados e a Tabela de Funções de Saída.

Além dessas tabelas principais, outras estruturas foram definidas: Tabela de Associações Ativas, Tabela de Endereçamento, Lista de Reconhecimentos Pendentes na MMPM e Lista de Reconhecimentos Pendentes na ORPM.

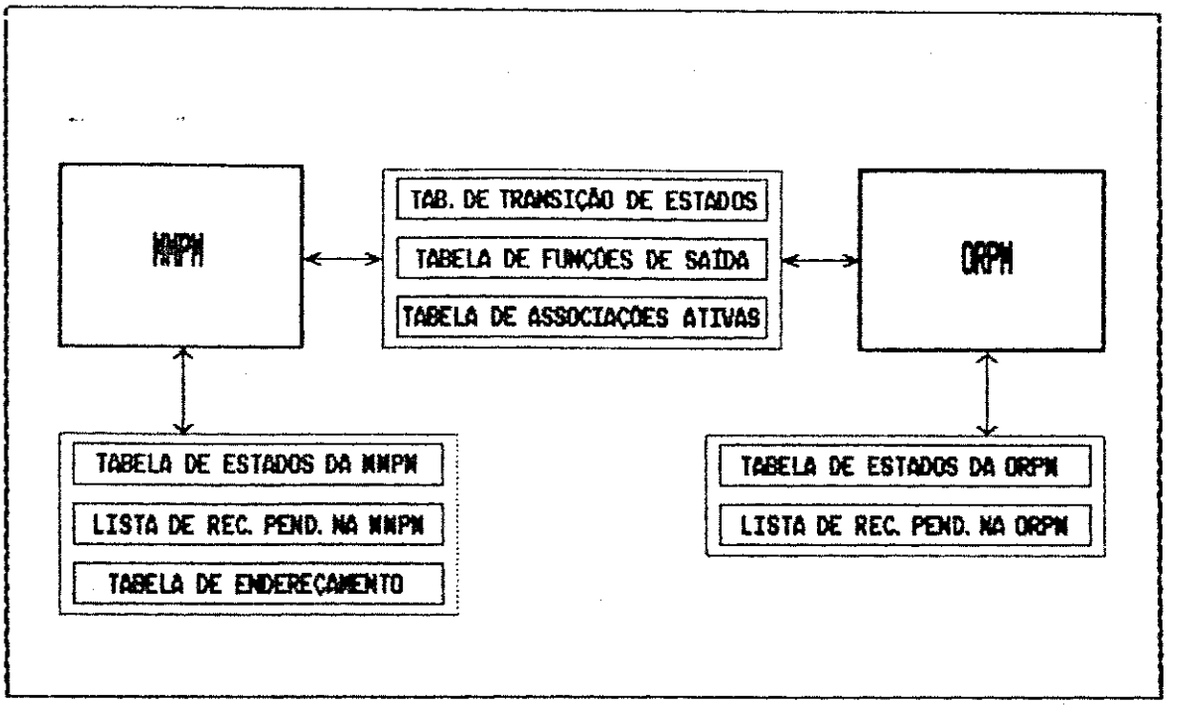


FIGURA 5.4 - ACESSO DAS MAQUINAS DE PROTOCOLO ÀS ESTRUTURAS DE DADOS

É apresentada, a seguir, uma descrição de cada uma.

5.3.1 TABELA DE ESTADOS MMPM

Esta tabela é um arranjo unidimensional indexado por solicitação de serviço. Cada elemento ativo da tabela corresponde a uma solicitação de serviço pendente. Os elementos inativos correspondem às solicitações de serviços pendentes que a MMPM poderá ainda utilizar.

A tabela é criada com uma dimensão igual ao produto do número máximo de associações permitidas pelo número máximo de serviços pendentes por associação. Este último valor é negociado durante o estabelecimento da associação; entretanto, a tabela é criada antes de qualquer comunicação

e, além disso, o valor pode variar de associação para associação. Por isso, foi utilizado o valor *default* mantido pela MMPM, que representa sempre o máximo definido pela implementação. Assim, a tabela poderá ficar com folga nos casos em que for negociado um número máximo de serviços pendentes menor que o valor *default*. Os valores adotados nesta implementação foram:

- número máximo de associações permitidas = 32. Esse valor corresponde ao número máximo de estações no segmento definido pelo Mini-MAP [GM/MAP,1988];
- número máximo de serviços pendentes por associação = 10 (definição local).

Ao receber uma requisição de serviço, a MMPM verifica se o número de serviços permitidos por associação foi ultrapassado. Em caso afirmativo, a requisição será rejeitada, pois não será possível a abertura da máquina de estados, mesmo havendo elementos livres na tabela. Este controle também é feito para comunicações sem associação, para manter um equilíbrio no número de serviços pendentes por comunicação.

A Tabela de Estados armazena todas as informações necessárias ao gerenciamento do contexto MMS. Com essas informações é possível analisar a situação atual do serviço pendente e qual o procedimento a seguir na chegada de um evento. Um evento é qualquer tipo válido de mensagem recebida através das interfaces superior ou inferior: pri-

mitivas MMS, primitivas LLC, primitivas ORSE e primitivas do gerente da estação.

Cada requisição de serviço válida recebida pela MMPM gera uma máquina de estados, que corresponde a um elemento na tabela. A máquina deixará de existir quando a solicitação de serviço pendente for atendida.

A Figura 5.5 mostra a estrutura da Tabela de Estados da MMPM e os dados mantidos por ela.

Como a MMPM pode manter várias associações pendentes e vários serviços pendentes por associação, a pesquisa na tabela é feita com o identificador do serviço (***invoke_id***) e o identificador da associação (o endereço MAC da estação remota). Quando ocorre um evento, a tabela é pesquisada com o valor do ***invoke_id*** e o endereço MAC da estação remota, ambos presentes na primitiva ou UDP, para obtenção do estado atual da solicitação de serviço (variável **estado_atual**). O ***invoke_id*** identifica univocamente uma solicitação de serviço em uma associação; o endereço MAC da estação remota é utilizado para identificar uma associação ou, no caso de comunicação sem associação, a estação origem/destino do serviço. Caso não seja encontrado um elemento ativo (**ativa** = *TRUE*) com o ***invoke_id*** e o endereço MAC remoto (**end_mac_r**) correspondentes, identifica-se o estado atual como sendo CLOSED, ou seja, não existe máquina de estados ativa para o ***invoke_id*** procurado. Com estes dados (evento e estado) efetua-se uma busca na Tabela de Transição de Estados.

```
struct te_mmpm {
  BOOL             ativa;         /* máquina de estados ativa (V/F) */
  enum estados     estado_atual;  /* estado atual da solicitação de serviço */
  ULONG            invoke_id;     /* identificação da solicitação de serviço pendente */
  enum serv_mms    nome_servico;  /* nome do serviço pendente */
  enum status_rsp  status_resp;   /* indica se a primitiva response é positiva ou negativa */
  struct dados_mms primitiva;     /* última primitiva MMS recebida do usuário local */
  unsigned         prioridade;    /* usado pelo LLC */
  unsigned         end_mac_l [3]; /* endereço MAC local - identifica univocamente a estação */
  unsigned         end_mac_r [3]; /* endereço MAC remoto - identifica univocamente a estação */
  unsigned char    ssap;          /* ponto de acesso LLC local */
  unsigned char    dsap;          /* ponto de acesso LLC remoto */
};
```

FIGURA 5.5 - ESTRUTURA DA TABELA DE ESTADOS DA MMPM

5.3.2 TABELA DE ESTADOS ORPM

Como a Tabela de Estados MMPM, esta tabela é um arranjo unidimensional indexado por solicitação de serviço. Cada elemento ativo da tabela corresponde a uma solicitação de serviço de resposta imediata pendente. Os elementos inativos correspondem às solicitações de serviço de resposta imediata que a ORPM pode ainda vir a utilizar.

A dimensão desta tabela é a mesma da Tabela de Estados MMPM. A ORPM, entretanto, não controla a quantidade de serviços ORSE pendentes por associação.

Observando-se o funcionamento da ORPM descrito no capítulo 3 pode-se perceber que este controle é feito pela MMPM: cada primitiva *request* ORSE solicita a recepção de uma UDP MMS *request* ou *response* da estação remota (que não faz parte do anel). Ao receber a UDP MMS esperada, a ORPM a repassa para a MMPM efetuar o devido tratamento. Se

a UDP recebida for *request*, a MMPM verifica se o limite de serviços pendentes por associação não foi ultrapassado. Se a UDP recebida for *response*, a MMPM já terá registrado uma primitiva *request* enviada pelo usuário antes deste solicitar o envio da primitiva *response* através da primitiva ORSE. Logo, o número de primitivas ORSE pendentes por associação nunca será maior que o número máximo permitido de serviços MMS pendentes por associação.

A ORPM controla sua Tabela de Estados da mesma forma que a MMPM: após a ocorrência de um evento (primitiva ORSE), a situação atual do serviço pendente é analisada e uma ação de saída é executada. Se o evento for válido, uma máquina de estados é criada e existirá enquanto a requisição de serviço estiver pendente.

A Figura 5.6 mostra a estrutura da Tabela de Estados da ORPM e os dados mantidos por ela. Como se pode observar, esta tabela não armazena o nome do serviço, o status da primitiva *response* e a última primitiva recebida, presentes na Tabela de Estados da MMPM, porque o único serviço ORSE que gera máquina de estados é *OBTAIN_REPLY.REQ*, o qual não gera primitiva *response*. São incluídos dois novos valores, não presentes na Tabela de Estados da MMPM: *invoke_id* esperado e tipo de UDP esperada. Esses valores são usados para determinar se a UDP recebida da estação remota refere-se ao serviço esperado e é do tipo esperado.

```
struct te_orpm {
  BOOL              ativa;         /* máquina de estados ativa (V/F) */
  enum estados      estado_atual;  /* estado atual da solicitação de serviço */
  ULONG             invoke_id;     /* identificação da solicitação de serviço pendente */
  ULONG             inv_id_esper;  /* invoke_id da primitiva response esperada pelo MMS */
  enum tipo_pdu_mms pdu_esper;     /* tipo de UDP que o MMS espera receber da estação remota */
  unsigned          prioridade;    /* usado pelo LLC */
  unsigned          end_mac_l [3]; /* endereço MAC local - identifica univocamente a estação */
  unsigned          end_mac_r [3]; /* endereço MAC remoto - identifica univocamente a estação */
  unsigned char     ssap;          /* ponto de acesso LLC local */
  unsigned char     dsap;          /* ponto de acesso LLC remoto */
};
```

FIGURA 5.6 - ESTRUTURA DA TABELA DE ESTADOS DA ORPM

O procedimento seguido pela ORPM para identificar o estado atual de uma solicitação de serviço pendente na Tabela de Estados ORPM é o mesmo seguido pela MMPM.

Após identificação do evento e do estado da solicitação de serviço, efetua-se uma busca na Tabela de Transição de Estados.

5.3.3 TABELA DE TRANSIÇÃO DE ESTADOS

Esta tabela é um arranjo unidimensional, onde cada elemento armazena uma intersecção evento/estado válida e os dados correspondentes à intersecção.

A estrutura unidimensional foi adotada como alternativa à estrutura bidimensional, pois esta última teria 110 elementos (11 eventos x 10 estados), com apenas 19 elementos válidos. Com a estrutura unidimensional, a busca seqüencial não compromete de modo muito significativo o tempo de acesso em relação à busca por índice da estrutura

bidimensional, devido à pequena dimensão da tabela (19 índices).

Como se pode observar na Figura 5.7, cada elemento da tabela armazena: evento, estado, um apontador para uma função de teste de predicado e um número inteiro, usado como primeiro índice na Tabela de Funções de Saída.

```
struct t_trans_est
  {
  enum eventos       ev;             /* evento válido */
  enum estados       est;            /* estado válido */
  int                (*test_pred)(); /* apontador para teste de predicado */
  int                indice_1;       /* primeiro índice da Tabela de Funções de Saída */
  struct t_trans_est *prox;          /* próximo nó da lista */
  };
```

FIGURA 5.7 - ESTRUTURA DA TABELA DE TRANSIÇÃO DE ESTADOS

Após a pesquisa na tabela, caso não seja encontrado o nó correspondente ao evento/estado, nenhuma ação de saída é executada, e o MMS passa a aguardar o próximo evento. Por exemplo: recebimento de uma primitiva *response* com a máquina de estados *CLOSED* (*CLOSED* permite apenas o recebimento de uma primitiva *request*).

Se for encontrada na tabela o nó correspondente ao evento/estado, executa-se a função de teste de predicado indicada pelo apontador de função armazenado no nó. Cada teste de predicado retorna um número inteiro. Este número será utilizado como segundo índice da Tabela da Funções de Saída, para obter a função de saída a ser executada (o

primeiro índice é um dos campos do nó da Tabela de Transição de Estados).

Os testes de predicado analisam o contexto atual da associação e efetuam verificações sintáticas e semânticas das mensagens recebidas, dentre as quais:

- Se há elementos livres na Tabela de Estados MMPM ou ORPM para aceitação de novas solicitações de serviços;

- Se há elementos livres na Tabela de Associações Ativas (descrita mais adiante) para estabelecimento de novas associações;

- Se há correspondência entre o ponto de acesso de entrada local e o tipo de comunicação com a estação remota. Por exemplo: se o tipo de comunicação for com associação, verifica-se se o endereço do ponto de acesso de entrada local indicado na primitiva é o endereço do ponto de acesso de entrada local para comunicação com associação;

- Obediência do usuário MMS local e da estação remota aos limites do próprio MMS no que se refere aos parâmetros da primitiva *Initiate_Request*, negociados no estabelecimento da associação;

- *Status* da primitiva *response* recebida do usuário local, para mapeamento da primitiva em uma *RESPONSE_PDU* ou em uma *ERROR_PDU*;

- Tipo de comunicação e classe de endereçamento, para mapeamento da primitiva na UDP e na primitiva LLC corretas;

- *Status* do reconhecimento recebido do LLC remoto com relação à primitiva LLC enviada pelo MMS;

- Se há correspondência entre o contexto atual e o tipo de primitiva LLC, de UDP, ou de serviço MMS (confirmado/não confirmado) recebidos;

- Se há correspondência entre o tipo de UDP esperado pela ORPM e o tipo de UDP recebido;

- Se há correspondência entre o *invoke_id* esperado pela ORPM e o *invoke_id* da UDP recebida;

- Se há coerência na requisição de serviço feita pela estação remota e recebida pela ORPM em relação ao tipo de comunicação (resposta imediata).

O conteúdo da Tabela de Transição de Estados é mostrado na Figura 5.8.

```
struct t_trans_est tte [19] =
 {{OBTAIN_REPLY_REQ       , CLOSED          , t_pred_13, 13},
  {X_REQ                  , CLOSED          , t_pred_2 , 0 },
  {X_RESP                 , NEED_RESP       , t_pred_1 , 1 },
  {FORCE_CLOSE_REQ        , NEED_RESP       , t_pred_16, 2 },
  {DATA_ACK_IND           , CLOSED          , t_pred_9 , 3 },
  {DATA_ACK_IND           , WAIT_CONFM_RESP , t_pred_5 , 5 },
  {DATA_ACK_STATUS_IND    , CLOSED          , t_pred_14, 16},
  {DATA_ACK_STATUS_IND    , WAIT_NULL_ACK   , t_pred_3 , 6 },
  {DATA_ACK_STATUS_IND    , WAIT_CONFM_ACK  , t_pred_4 , 7 },
  {DATA_ACK_STATUS_IND    , WAIT_UNCON_ACK  , t_pred_6 , 8 },
  {DATA_ACK_STATUS_IND    , WAIT_RESP_ACK   , t_pred_8 , 9 },
  {REPLY_UPDATE_STATUS_IND, WAIT_REQ_UPDATE , t_pred_16, 10},
  {REPLY_UPDATE_STATUS_IND, WAIT_RESP_UPDATE, t_pred_16, 10},
  {DATA_IND               , CLOSED          , t_pred_15, 4 },
  {RECEIVED_PDU_IND       , CLOSED          , t_pred_11, 17},
  {RECEIVED_PDU_IND       , WAIT_CONFM_RESP , t_pred_12, 11},
  {REPLY_STATUS_IND       , WAIT_REPLY      , t_pred_10, 12},
  {STATION_FAILURE_IND    , CLOSED          , t_pred_7 , 14},
  {STATION_FAILURE_IND    , WAIT_CONFM_RESP , t_pred_7 , 15}};
```

FIGURA 5.8 - CONTEODO DA TABELA DE TRANSIÇAO DE ESTADOS

A Tabela 5.1 mostra a que máquina de estados (MMPM ou ORPM) correspondem os eventos previstos na tabela de transição de estados, bem como a proveniência desses eventos.

TABELA 5.1 - DISTRIBUIÇAO DOS EVENTOS NAS MAQUINAS DE ESTADOS

MAQUINA DE ESTADOS	EVENTOS	PROVENIENCIA
MMPM	X.req‡, X.resp‡, Force_Close.req	Usuário MMS
	Data_Ack.ind, Data_Ack_Status.ind, Reply_Update_Status.ind, Data.ind	Camada LLC
	Received_PDU.ind	ORPM
	Station_Failure.ind	Gerente da estação
ORPM	Obtain_Reply.req	Usuário MMS
	Reply_Status.ind	Camada LLC

‡ X.req e X.resp indicam qualquer primitiva MMS, exceto Force_Close.req.

5.3.4 TABELA DE FUNÇÕES DE SAIDA

Esta tabela é um arranjo bidimensional, indexado pelo inteiro encontrado na Tabela de Transição de Estados e o inteiro fornecido pela função de teste de predicado. O conteúdo da tabela de função de saída é apresentado na Figura 5.9.

```
void (* tfsaida [18] [7])() =
 {
 fsd_1 , fsd_2 , fsd_8 , fsd_9 , fsd_42, fsd_51, fsd_50,
 fsd_3 , fsd_4 , fsd_5 , fsd_6 , fsd_54, fsd_52, fsd_53,
 fsd_7 , NULL  , NULL  , NULL  , NULL  , NULL  , NULL  ,
 fsd_32, fsd_31, fsd_33, fsd_34, fsd_30, fsd_40, NULL  ,
 fsd_32, fsd_34, fsd_40, NULL  , NULL  , NULL  , NULL  ,
 fsd_21, fsd_22, fsd_23, fsd_20, fsd_24, fsd_56, fsd_57,
 fsd_10, fsd_11, NULL  , NULL  , NULL  , NULL  , NULL  ,
 fsd_12, fsd_13, fsd_14, fsd_55, NULL  , NULL  , NULL  ,
 fsd_26, fsd_27, NULL  , NULL  , NULL  , NULL  , NULL  ,
 fsd_35, fsd_36, fsd_37, fsd_38, fsd_39, NULL  , NULL  ,
 fsd_28, NULL  , NULL  , NULL  , NULL  , NULL  , NULL  ,
 fsd_16, fsd_17, fsd_18, fsd_15, fsd_19, fsd_58, fsd_59,
 fsd_47, fsd_48, fsd_49, NULL  , NULL  , NULL  , NULL  ,
 fsd_46, fsd_50, NULL  , NULL  , NULL  , NULL  , NULL  ,
 fsd_29, NULL  , NULL  , NULL  , NULL  , NULL  , NULL  ,
 fsd_25, NULL  , NULL  , NULL  , NULL  , NULL  , NULL  ,
 fsd_41, NULL  , NULL  , NULL  , NULL  , NULL  , NULL  ,
 fsd_43, fsd_44, fsd_45, NULL  , NULL  , NULL  , NULL  ,
 };
```

FIGURA 5.9 - CONTEUDO DA TABELA DE FUNÇÕES DE SAIDA

Nem todos os elementos do arranjo são elementos válidos na tabela, tendo portanto conteúdo nulo (ou seja, o apontador de função é igual a NULL). Nesse caso, nenhuma ação de saída é executada para o evento de entrada e, como conseqüência, o evento ocorrido é ignorado. Os elementos

válidos contêm um apontador para a função de saída correspondente ao evento/estado/teste de predicado.

Cada função executa uma série de procedimentos de saída, tais como: montagem de UDPs, montagem de primitivas LLC, envio de primitivas, abertura e fechamento de máquinas de estado, atualização de estado na Tabela de Estados e estabelecimento e encerramento de associações.

5.3.5 TABELA DE ASSOCIAÇÕES ATIVAS

Consiste de um arranjo unidimensional, onde cada elemento é uma estrutura que armazena informações referentes a uma associação.

A dimensão da tabela é fixa e igual ao número máximo de estações na rede, que na arquitetura Mini-MAP é limitado a 32 [GM/MAP,1988]. Só será possível o estabelecimento de uma nova associação se houver pelo menos um elemento livre na tabela.

A Tabela de Associações Ativas mantém informações sobre as associações abertas, necessárias mesmo após a execução dos serviços e enquanto a associação existir. Essas informações são mostradas na Figura 5.10.

O objetivo desta tabela é indicar as associações existentes, uma vez que a Tabela de Estados mantém informações sobre as associações apenas enquanto o serviço estiver pendente. Quando o serviço for concluído, a máquina de estados é eliminada da tabela de estados, mas a associação

permanece ativa até que seja encerrada de forma normal (primitiva *Conclude*) ou abrupta (primitiva *Abort*).

```
struct t_ass_at
  {
  BOOL             ativa;        /* associação ativa (V/F) */
  unsigned         end_mac_l [3]; /* endereço MAC local - identifica univocamente a estação */
  unsigned         end_mac_r [3]; /* endereço MAC remoto - identifica univocamente a estação */
  unsigned char    ssap;         /* ponto de acesso LLC local */
  unsigned char    dsap;         /* ponto de acesso LLC remoto */
  unsigned         prioridade;   /* usado pelo LLC */
  P_RES_INI        param_init;   /* parâmetros negociados durante estabelec. de associação */
  };


typedef struct
 {
 int               versao;            /* versao do MMS utilizado */
 unsigned long     tam_mens;          /* tamanho máximo de uma UDP */
 int               spchamante;        /* num. máximo de serviços pendentes no chamador */
 int               spchamado;         /* num. máximo de serviços pendentes no chamado */
 enum   list_tipos niv_estrut;        /* nivel de complexidade dos dados */
 unsigned          cbb_horiz [CBBH];  /* CBBs horizontais suportadas */
 unsigned          cbb_c_vert [CBBV]; /* CBBs verticais suportadas pelo cliente */
 unsigned          cbb_s_vert [CBBV]; /* CBBs suportadas pelo servidor */
 } P_RES_INI;
```

FIGURA 5.10 - ESTRUTURA DA TABELA DE ASSOCIAÇÕES ATIVAS

5.3.6 TABELA DE ENDEREÇAMENTO

A Tabela de Endereçamento é um arranjo unidimensional com apenas quatro elementos, cada um contendo um identificador. Esses identificadores correspondem aos tipos de LSAPs definidos no Capítulo 3.

Esta tabela é utilizada para verificar se o ponto de acesso através do qual a mensagem foi recebida corresponde ao tipo de comunicação desejado.

A Figura 5.11 mostra a estrutura e a conteúdo desta tabela.

```
struct t_end
  {
  enum tipo_comunic tp_com;
  } tab_ender [NUM_SAP] = {COM_ASSOCIACAO,
                           SEM_ASSOCIACAO,
                           RESPOSTA_IMEDIATA,
                           GERENTE_ESTACAO};
```

FIGURA 5.11 - ESTRUTURA E CONTEUDO DA TABELA DE ENDEREÇAMENTO

5.3.7 LISTA DE RECONHECIMENTOS PENDENTES NA MMPM

A MMPM mantém duas estruturas desse tipo: uma é utilizada quando a estação não participa do anel lógico; a outra é utilizada quando a estação participa do anel lógico. Cada estrutura tem como finalidade manter, em uma fila de espera, a identificação de todos os serviços MMS que ainda não receberam reconhecimento do LLC remoto.

O reconhecimento indica que a LSDU foi recebida pelo LLC remoto. A informação de *status* contida no reconhecimento indica se a LSDU enviada pode ou não ser processada pelo LLC remoto. Esse tipo de mensagem identifica apenas a estação com a qual está sendo mantida a comunicação, não contendo qualquer identificação a respeito do serviço a que se refere. Como pode haver mais de um serviço pendente numa mesma associação, tornou-se necessária a criação desta estrutura.

Cada nó da lista contém apenas o índice da Tabela de Estados MMPM, em cujo elemento estão armazenadas as informações correspondentes ao serviço aguardando reconhecimento.

Cada vez que uma UDP MMS é enviada para uma estação remota, um nó é acrescentado à lista. Ao chegar uma mensagem de reconhecimento do LLC, obtém-se o índice da Tabela de Estados do primeiro nó da lista. Verifica-se, então, no elemento correspondente da Tabela de Estados, se o endereço da estação que enviou o reconhecimento é o mesmo endereço remoto indicado na tabela. Se for, o nó é removido e as ações pertinentes ao serviço são executadas. Se não for, repetem-se os procedimentos acima para o próximo nó da lista, até a condição ser satisfeita.

Como se pode observar, apenas a estação remota é identificada. Se houver mais de um serviço pendente na comunicação entre duas estações, o reconhecimento será atribuído ao serviço aguardando reconhecimento há mais tempo, ou seja, o que estiver mais à frente na lista. Esse mecanismo leva em conta o fato de que o MMS assume que os serviços nas camadas inferiores são confiáveis e, portanto, as mensagens de reconhecimento deverão chegar na mesma ordem em que as LSDUs foram enviadas [EIA/MMS,1987].

5.3.8 LISTA DE RECONHECIMENTOS PENDENTES NA ORPM

Esta lista tem a mesma finalidade e a mesma estrutura da Lista de Reconhecimentos Pendentes na MMPM:

armazena o índice da Tabela de Estados ORPM, através do qual se obtém as informações referentes ao serviço pendente.

O funcionamento também é o mesmo: a primeira mensagem de reconhecimento a chegar é atribuída ao serviço aguardando há mais tempo.

5.4 DETALHES DE FUNCIONAMENTO DA IMPLEMENTAÇÃO

Nesta secção será feita uma descrição detalhada do funcionamento da implementação, na qual é exposta a seqüência e o acesso de cada módulo do processo às estruturas de dados e às interfaces superior e inferior.

5.4.1 MÓDULO GERAL

O fluxograma da Figura 5.12 ilustra como o módulo geral obtém e encaminha as mensagens para os módulos MMPM e ORPM.

Ao entrar em operação, o processo MMS:

- cria as Tabelas de Estados MMPM e ORPM;
- cria a Tabela de Associações Ativas;
- cria a Tabela de Transição de Estados;
- cria a Tabela de Funções de Saída;
- cria a Tabela de Endereçamento;
- inicializa os descritores das Listas de Reconhecimentos Pendentes da MMPM e da ORPM.

Para retirar as mensagens das filas aloca-se um *buffer* com tamanho em bytes igual ao número de bytes ocupado pela estrutura da mensagem.

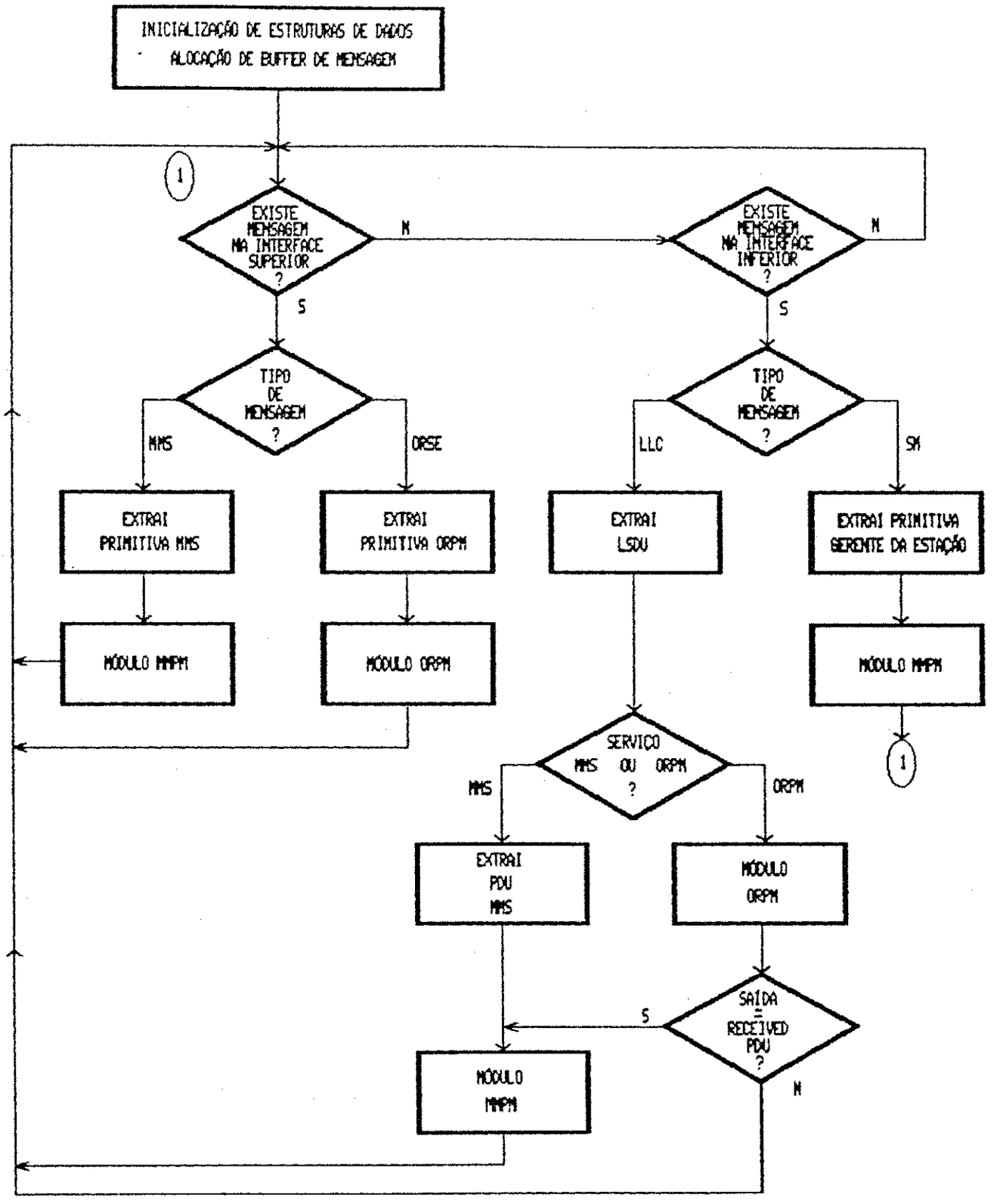


FIGURA 5.12 - FUNCIONAMENTO DO MÓDULO GERAL

Como já foi dito no Capítulo 3, a interface inferior possui quatro pontos de acesso. Na amostragem desta interface verifica-se a existência de mensagem em cada ponto de acesso. A mensagem encontrada é tratada e o próximo ponto de acesso é verificado. Este procedimento se repete até o último ponto de acesso e, então, é feita nova varredura na interface superior.

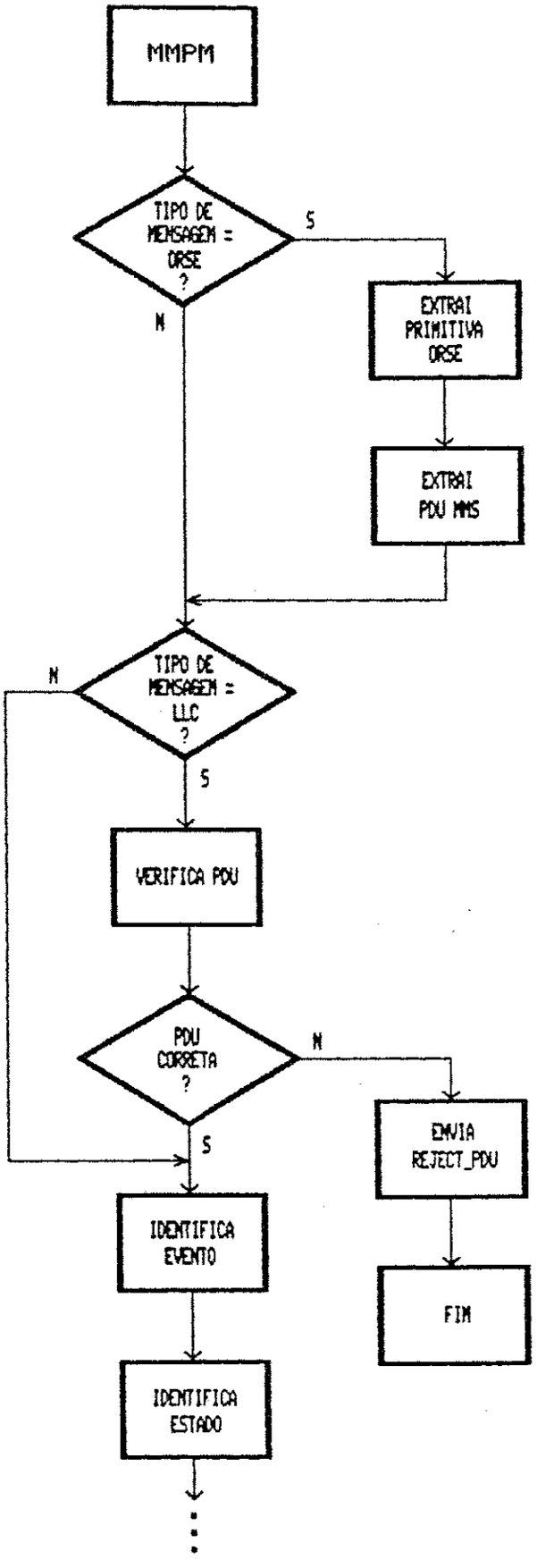
O tratamento da mensagem é interrompido em três casos: o processo não conseguiu retirar a mensagem da fila; a MMPM não conseguiu alocar espaço para incluir um nó na Lista de Reconhecimentos Pendentes ou a ORPM não conseguiu alocar espaço para incluir um nó na Lista de Reconhecimentos Pendentes. Caso ocorra um desses erros, uma mensagem de erro será mostrada na tela e o processo passará a ler as filas de entrada seguintes para obtenção de nova mensagem.

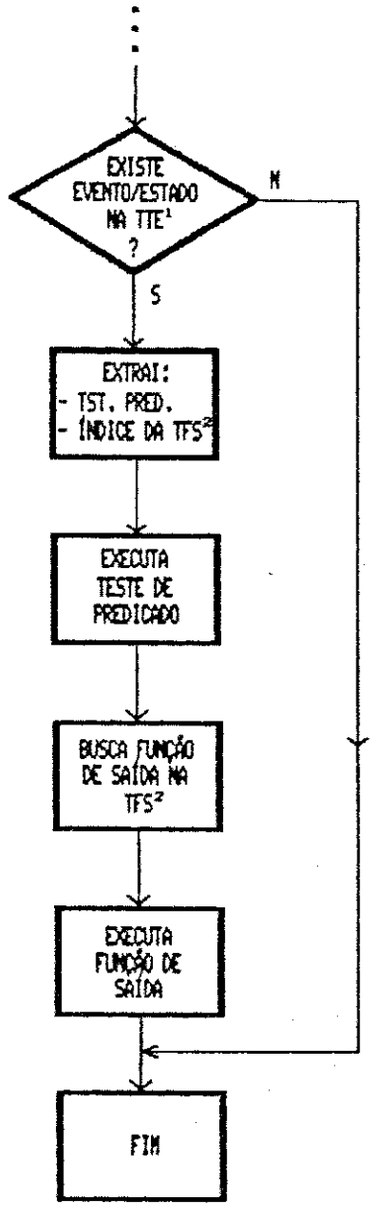
Caso o tipo da mensagem recebida seja inválido, a mensagem é ignorada e o usuário local é comunicado.

5.4.2 MÓDULO MMPM

O fluxograma da Figura 5.13 mostra como a MMPM trata as mensagens retiradas das filas pelo módulo geral.

Se o tipo de mensagem recebido for **ORSR**, o módulo interpreta que o evento ocorrido foi uma primitiva *ReceivedPDU.Indication* e extrai da primitiva a UDP MMS enviada pela estação remota.


MMPM
TIPO DE MENSAGEM = ORSE ?
S
N
EXTRAI PRIMITIVA ORSE
EXTRAI PDU MMS
TIPO DE MENSAGEM = LLC ?
N
S
VERIFICA PDU
PDU CORRETA ?
N
S
ENVIA REJECT_PDU
FIM
IDENTIFICA EVENTO
IDENTIFICA ESTADO



[1]TTE = TABELA DE TRANSIÇÃO DE ESTADOS

[2]TFS = TABELA DE FUNÇÕES DE SAÍDA

FIGURA 5.13 - FUNCIONAMENTO DA MMPM

Se o tipo da mensagem recebida for LLC, o módulo efetua algumas verificações preliminares na UDP MMS recebida. Caso seja encontrada alguma irregularidade, o usuário MMS local é comunicado via primitiva *RejectSent.Indication*

e o usuário MMS remoto via *REJECT_PDU*. Outras verificações na UDP MMS são efetuadas nos testes de predicado.

Há três situações de erro nas quais nenhuma função de saída é acionada e a execução do módulo MMPM é encerrada: a MMPM não encontra na Tabela de Transição de Estados o nó correspondente ao evento/estado procurados; o teste de predicado retorna um valor menor do que zero; a MMPM não consegue identificar o evento ocorrido. Nos dois primeiros casos a mensagem é descartada, e no último caso o usuário local é comunicado. Em todos os casos, o processo passa a ler as próximas filas de entrada para obter uma mensagem.

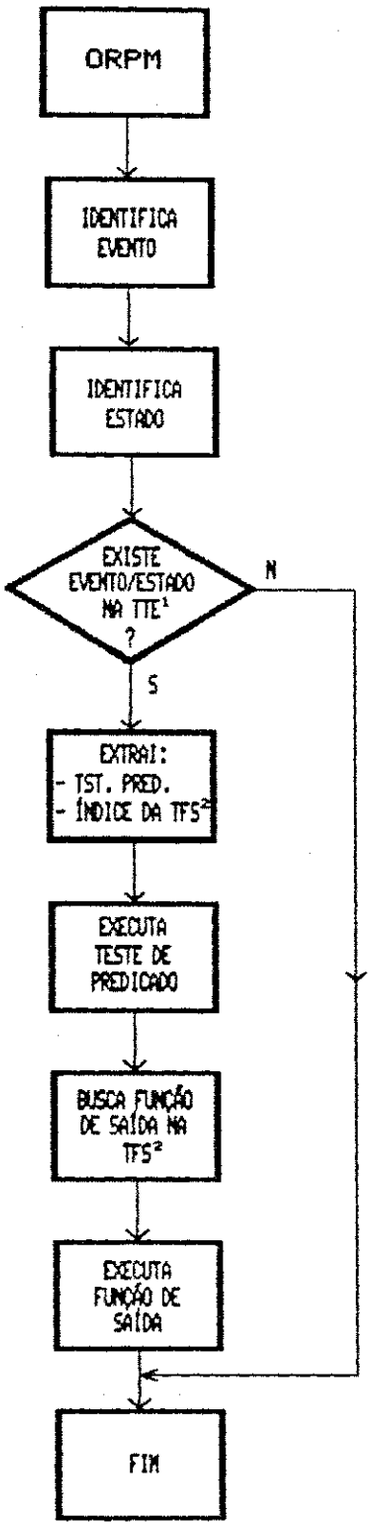
5.4.3 MÓDULO ORPM

O fluxograma da Figura 5.14 mostra o mecanismo de funcionamento da ORPM.

Como se pode observar, a partir da identificação do evento e do estado, a ORPM funciona de forma semelhante à MMPM.

As condições para o término da execução do módulo ORPM sem acionar funções de saída são as mesmas descritas para o módulo MMPM.

Se uma função de saída gerar uma primitiva *ReceivedPDU.Indication* o módulo ORPM aciona o módulo MMPM com esta primitiva e encerra a execução de forma normal.



[1]TTE = TABELA DE TRANSIÇÃO DE ESTADOS

[2]TFS = TABELA DE FUNÇÕES DE SAÍDA

FIGURA 5.14 - FUNCIONAMENTO DA ORPM

5.5 ESTRUTURA DAS PRIMITIVAS E UDPS NA LINGUAGEM C

A partir da definição ASN.1 encontrada em [GM/MAP,1988] e [EIA/MMS,1987] e usando o mapeamento descrito no Anexo 3, definiu-se a estrutura das primitivas MMS, ORSE e do gerente da estação e das UDPs MMS na linguagem C. A estrutura usada das primitivas LLC é definida em [TURNELL,1990].

As primitivas MMS consistem de parâmetros comuns e de parâmetros específicos de cada serviço. Os parâmetros comuns identificam o tipo de primitiva (*request*, *indication*, *response* ou *confirm*), a classe de endereçamento, os endereços MAC/LLC locais e de destino, o tipo de serviço e a identificação da solicitação de serviço. A Figura 5.15 ilustra a estrutura geral das primitivas MMS.

```
struct cab_mms {
  enum   tipo_prim    tipo;        /* tipo de primit.: request, indication, response ou confirm */
  enum   lk_class_end classe_end;  /* classe de endereçam.: individual, grupo ou reply buffer */
  struct endereco     ender;       /* estrutura com os endereços MAC e LLC locais e remotos */
};

struct dados_mms {
  enum  serv_mms servico;          /* nome do serviço MMS: initiate, conclude, etc. */
  ULONG          invoke_id;        /* identificação da solicitação de serviço */
  union {
    union  request    par_req;     /* parâmetros de uma primitiva request */
    union  indication par_ind;     /* parâmetros de uma primitiva indication */
    struct response   par_rsp;     /* parâmetros de uma primitiva response */
    struct confirm    par_cfm;     /* parâmetros de uma primitiva confirm */
  } param_prim;
};

struct prim_mms {
  struct cab_mms   cabecalho;
  struct dados_mms conteudo;
};
```

FIGURA 5.15 - ESTRUTURA DAS PRIMITIVAS MMS

A estrutura das primitivas ORSE é semelhante à estrutura das primitivas MMS: consiste de parâmetros comuns e parâmetros específicos de cada primitiva. Os parâmetros comuns identificam o tipo de primitiva (*request*, *indication* ou *confirm*), os endereços MAC/LLC locais e remotos, o tipo de serviço e a identificação de solicitação de serviço pendente. A estrutura geral das primitivas ORSE é mostrada na Figura 5.16.

```
struct cab_orse {
  enum   tipo_prim tipo;              /* tipo de primitiva: request, indication ou confirm */
  struct endereco  ender;             /* estrutura com os endereços MAC e LLC locais e remotos */
};

struct dados_orse {
  enum   serv_orpm servico;           /* nome do serviço ORSE: obtainReply ou receivedPDU */
  ULONG            invoke_id;         /* identificação da solicitação de serviço */
  union {
    ARGUM_RECPDU     receivedPDU_ind; /* parâmetros de receivedPDU */
    ARGUM_OBTREPLY   obtainReply_req; /* parâmetros de obtainReply.request */
    RESULT_OBTREPLY  obtainReply_cfm; /* parâmetros de obtainReply.confirm */
  } param;
};

struct prim_ors {
  struct cab_orse   cabecalho;
  struct dados_orse conteudo;
};
```

FIGURA 5.16 - ESTRUTURA DAS PRIMITIVAS ORSE

As primitivas do gerente da estação contêm o tipo de serviço e os endereços MAC/LLC locais e remotos, conforme mostra a Figura 5.17.

```
struct pria_stm {
  enum   serv_sm  servico;
  struct endereco ender;
};
```

FIGURA 5.17 - ESTRUTURA DAS PRIMITIVAS DO GERENTE DA ESTAÇÃO

As UDPs MMS são compostas de dois parâmetros: o tipo da UDP e os argumentos da primitiva. Esta estrutura é mostrada na Figura 5.18.

```
struct pdu_ams {
  enum   tipo_pdu_ams tipo_pdu; /* tipo da UDP: request, response, etc. */
  struct dados_ams    pri;      /* arguentos da primitiva */
};
```

FIGURA 5.18 - ESTRUTURA DAS UDPS MMS

As primitivas LLC são compostas de um campo de cabeçalho e um campo de conteúdo. Dos parâmetros contidos no campo de cabeçalho o MMS utiliza: o tipo da primitiva LLC, os endereços MAC local e remoto, o tamanho da UDP MMS que será enviada e o *status* do reconhecimento enviado pelo LLC remoto. Dos parâmetros do campo de conteúdo o MMS utiliza: os endereços LLC local e remoto e o campo de dados, que contém a UDP MMS. Os demais parâmetros são utilizados pelo LLC para controle interno da camada, e não dizem respeito ao MMS. A Figura 5.19 ilustra a estrutura das primitivas LLC.

```
struct cab_lsdu {
  enum     tipo_prim_llc tipo_prim;     /* tipo de prim. LLC: l_data_ack_ind, l_data_ind, etc. */
  enum     status        status;        /* status do reconhecim. remoto: OK, NO_RESOURCES, etc. */
  enum     situacao_lsdu situacao;      /* uso do LLC */
  unsigned               prioridade;    /* prioridade do serviço LLC */
  unsigned               testcode;      /* uso do LLC */
  unsigned               tam_pdu_mms;   /* tamanho da UDP MMS a ser enviada (em octetos) */
  unsigned               end_mac_l [3]; /* endereço MAC local */
  unsigned               end_mac_r [3]; /* endereço MAC remoto */
  struct   pdu_llc       *prox;         /* uso do LLC */
  struct   pdu_llc       *response;     /* uso do LLC */
};

struct pdu_llc {
  unsigned char  dsap;                  /* endereço LLC remoto */
  unsigned char  ssap;                  /* endereço LLC local */
  unsigned char  controle;              /* uso do LLC */
  char           *dados;                /* UDP MMS */
};

struct prim_llc {
  struct cab_lsdu cabecalho;
  struct pdu_llc  conteudo;
};
```

FIGURA 5.19 - ESTRUTURA DAS PRIMITIVAS LLC

5.6 VOLUME FINAL DE CÓDIGO

Esta implementação consumiu cerca de cinco mil linhas de código fonte em linguagem C, divididas entre os seguintes arquivos:

- mms_prim.h - definição da estrutura das primitivas.
- mms_stru.h - definição das estruturas de dados.
- mms_inic.h - conteúdo das estruturas de dados.
- mms_pdu.h - definição da estrutura da UDP MMS e da primitiva LLC.
- mms_par.h - definição de macros, tipos *enum* e tipos em C adaptados da sintaxe ASN.1.
- mms_fdef.h - definição de todas as funções utilizadas na implementação.
- mms_glob.h - definição das variáveis externas.
- main.c - programa principal (módulos geral, MMPM e ORPM).
- mms_fsai.c - contém todas as funções de saída.
- mms_tprd.c - contém todos os testes de predicado.
- mms_fun.c - contém funções secundárias utilizadas pelos demais arquivos.

Os arquivos de biblioteca (extensão *.h*) totalizaram aproximadamente 1.700 linhas, enquanto os arquivos fontes (extensão *.c*) totalizaram aproximadamente 3.300 linhas.

CAPITULO 6

TESTES DA IMPLEMENTAÇÃO

Como já foi exposto anteriormente, esta implementação foi concebida para ser executada em um Processador Preferencial, utilizando um Executivo desenvolvido na UFPb. Devido à maior disponibilidade de estações do tipo *SPARC* (arquitetura SUN 4c, baseada no microprocessador *SPARC*), comparada com a do PP, optou-se por efetuar os testes nas estações SUN [SUN,1990]. Foi utilizado o sistema operacional *UNIX System V*, que oferece recursos semelhantes aos que seriam utilizados no Executivo. Para o teste não foi necessário alterar a lógica do programa.

Para que a implementação fosse testada foi necessário o desenvolvimento de programas de teste que simulassem as camadas vizinhas ao MMS no Mini-MAP. Assim, criou-se:

- um programa de teste que simula o usuário MMS enviando e recebendo primitivas para/do MMS;
- um programa de teste que simula o funcionamento da camada de enlace recebendo primitivas do MMS e enviando-as à outra estação.

Os testes descritos neste capítulo têm como objetivo:

- validar algumas das primitivas MMS definidas na implementação;
- verificar o funcionamento interno do MMS.

Na validação das primitivas observou-se a coerência dos parâmetros passados entre as entidades pares MMS.

Na verificação do funcionamento interno do MMS foi observado o comportamento da implementação no que diz respeito aos itens:

- identificação do evento ocorrido;
- identificação do estado atual da solicitação de serviço;
- acesso correto às estruturas de dados;
- execução dos procedimentos de saída esperados, incluindo: abertura/fechamento de máquinas de estados, estabelecimento/encerramento de associações, atualização de estado, montagem e envio de primitivas MMS e LLC, etc.

As primitivas testadas foram *Initiate*, *Conclude* e *Read*. Como o serviço *Read* é confirmado e os procedimentos para as primitivas desse tipo são os mesmos, pode-se considerar que os testes foram feitos para serviços confirmados.

Os procedimentos de teste basearam-se na execução de dois processos MMS em uma mesma estação. Esses dois processos simularam a comunicação entre duas estações

físicas, cada um interagindo com os programas testadores superior e inferior.

Para um melhor entendimento da plataforma de testes utilizada, o capítulo inclui a descrição dos recursos de *software* utilizados durante a execução dos testes, além da descrição plataforma de testes e da apresentação dos resultados obtidos.

6.1 RECURSOS UTILIZADOS

A implementação e a plataforma de testes do MMS requerem a utilização de alguns recursos avançados do sistema operacional. Esses recursos são oferecidos pelo sistema operacional UNIX e permitem a duplicação de um processo e a execução simultânea de vários programas, bem como a comunicação entre eles através da utilização de filas de mensagens. Esses recursos são descritos a seguir.

6.1.1 DUPLICAÇÃO DE PROCESSOS E EXECUÇÃO SIMULTANEA DE PROGRAMAS

As chamadas ao supervisor para duplicar um processo e executar um programa são: ***fork ()*** e a família ***exec***, respectivamente.

A função ***fork ()*** cria um processo, chamado **processo filho**, a partir do processo atual, chamado **processo pai**. Esses dois processos são independentes, mas possuem o mesmo conteúdo. A função é tipicamente usada para criar múltiplas cópias de qualquer programa que deva executar procedimentos diferentes como parte de sua operação normal. Após a execução da função, o processo pai continua a executar normalmente; o processo filho inicia a execução no ponto imediatamente após o *fork ()*.

O *fork ()* retorna um identificador para cada processo. O identificador (*process ID*) do filho é sempre igual a zero; o identificador do pai (*parent ID*) é sempre

um número inteiro positivo. Esses valores são usados para determinar quais ações devem ser tomadas pelo pai e quais devem ser tomadas pelo filho. Caso ocorra um erro na execução do *fork ()*, o valor retornado será -1 e o processo filho não será criado.

A família ***exec*** é composta das funções ***execl***, ***execv***, ***execle***, ***execve*** e ***execvp***, cuja sintaxe é mostrada na Figura 6.1.

```
int execl (path, arg0, arg1, ..., argn, (char *)0)
char *path, *arg0, *arg1, ..., *argn;

int execv (path, argv)
char *path, *argv [];

int execle (path, arg0, arg1, ..., argn, (char *)0, envp)
char *path, *arg0, *arg1, ..., *argn, *envp [];

int execve (path, argv, envp)
char *path, *argv [], *envp [];

int execlp (file, arg0, arg1, ..., argn, (char *)0)
char *file, *arg0, *arg1, ..., *argn;

int execv (file, argv)
char *file, *argv [];
```

FIGURA 6.1 - SINTAXE DAS FUNÇÕES DA FAMILIA EXEC

O parâmetro ***path*** é o caminho completo do comando ou programa a ser executado.

Os parâmetros ***arg0***, ***arg1***, ..., ***argn*** formam uma lista de argumentos a serem passados para o novo processo. Pelo menos o parâmetro *arg0* deve estar presente, e seu conteúdo é sempre o último elemento de *path*. O último elemento da lista é 0.

O parâmetro ***file*** é o nome do arquivo com o programa ou comando a ser executado; nesse caso, o caminho é dado pela variável PATH do sistema.

O parâmetro ***argv*** é uma lista de argumentos para o novo processo; *argv* deve ter pelo menos um componente: o nome do arquivo indicado em *path* (omitindo o caminho). O último elemento da lista é um apontador nulo.

O parâmetro ***envp*** indica a configuração do ambiente para o novo processo.

Em programas na linguagem C deve-se usar as funções ***execl*** e ***execv***. A função ***execl*** é usada na execução de programas com número conhecido de argumentos; ***execv*** é utilizada quando não se conhece o número de argumentos do programa. As demais funções da família são usadas em rotinas de baixo nível.

Cada função ***exec*** transforma um programa executável em um novo processo. O programa executável é um arquivo binário composto das seguintes secções:

Cabeçalho

Define o tamanho das demais secções do arquivo e identifica o ponto de início da execução do processo.

Texto

Contém as instruções do programa a ser executado.

Dados

Contém todos os dados com valores iniciais.

BSS

Contém os dados sem valores iniciais. Quando o programa é transformado em processo, o UNIX atribui zeros a todas as variáveis no BSS.

Relocação

Define como o link-editor deve modificar o arquivo de programa, quando este é ligado a outros arquivos de programa. Esta secção é removida quando o link-editor termina a construção do arquivo binário executável.

Tabela de Símbolos

Relaciona símbolos a locações no processo. Pode ser removida durante a compilação do programa.

O processo é a execução do programa e contém as seguintes secções:

Texto

É uma imagem da secção de texto do arquivo de programa e possui tamanho fixo.

Dados

É obtida pela união das secções de dados e BSS do arquivo de programa, podendo crescer ou diminuir de tamanho conforme as necessidades.

Pilha

É criada pelo UNIX quando o processo é construído. Mantém os seguintes dados: variáveis locais de sub-rotinas, argumentos do programa e ambiente do processo.

Da mesma forma que a secção de dados, a secção de pilha aumenta ou diminui conforme as necessidades.

Bloco de Usuário

É um subconjunto de informações que o sistema mantém sobre o processo. Essas informações ficam residentes no espaço de endereçamento do núcleo e não são diretamente endereçáveis pelo processo. O bloco de usuário possui tamanho fixo.

A Figura 6.2 mostra como as funções ***exec*** mapeiam as secções de um arquivo de programa nas secções de um processo.

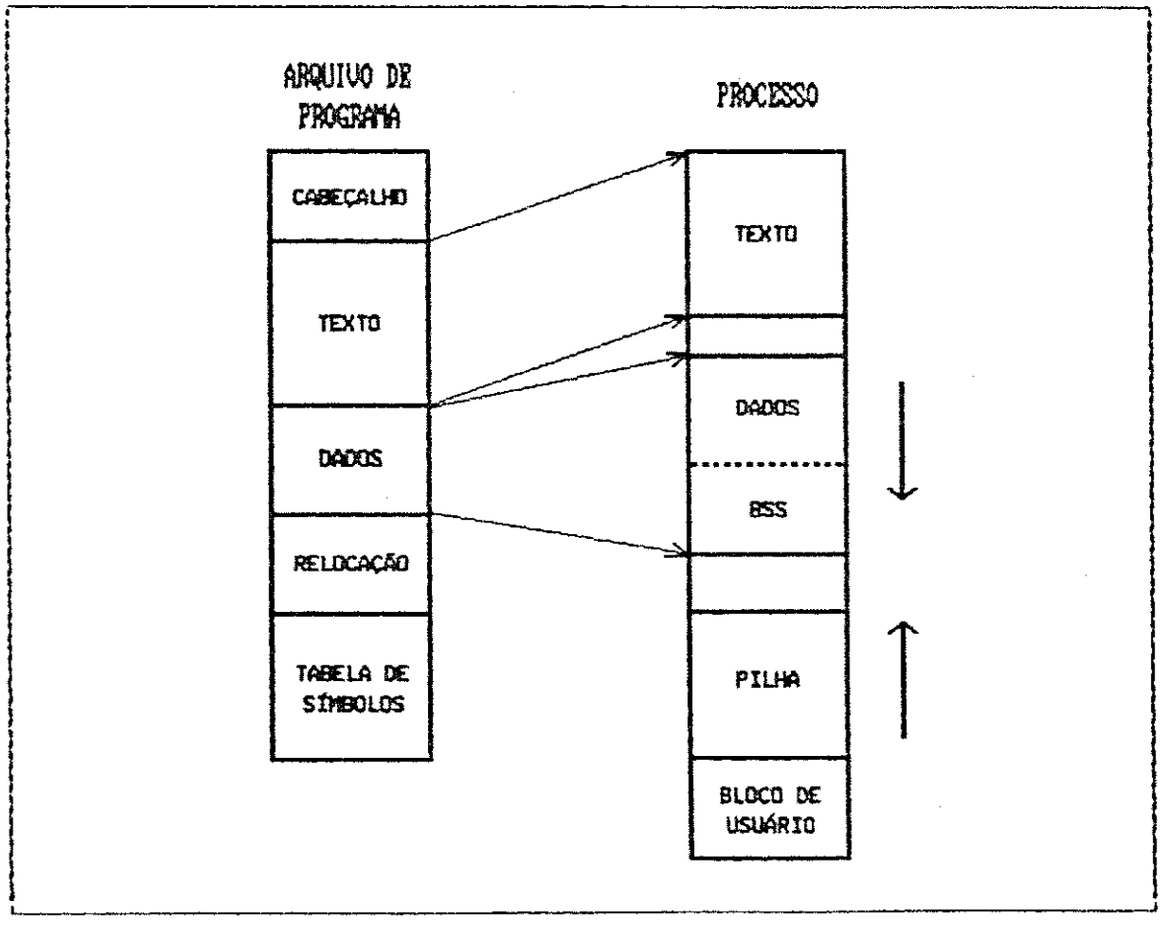


FIGURA 6.2 - MAPEAMENTO DE UM ARQUIVO DE PROGRAMA EM UM PROCESSO

A região vazia entre as secções de dados e texto do processo na Figura 6.2 permite qualquer ajuste exigido por restrições de *hardware* do gerenciamento de memória. A região vazia entre as secções de dados e pilha define a área livre usada para expandir ambas as secções. As setas na figura indicam o sentido de crescimento de cada secção.

6.1.2 COMUNICAÇÃO INTERPROCESSOS

O sistema operacional UNIX oferece diversos mecanismos de comunicação interprocessos. Nesta implementação utilizou-se apenas o recurso de **filas de mensagens**.

As filas são utilizadas quando as mensagens a serem trocadas são de tipos diferentes, e as tarefas a serem executadas pelos processos envolvidos dependem do tipo de mensagem recebida ou enviada. Cada fila de mensagem tem um dono, um grupo e um conjunto de permissões que definem o tipo de acesso permitido ao dono, ao grupo e aos demais usuários. O tipo de acesso pode ser de leitura ou escrita.

As funções UNIX para controle e acesso às filas de mensagens são: ***msgget***, ***msgctl***, ***msgsnd*** e ***msgrcv***. Para utilizar essas funções, o programa deve incluir as seguintes bibliotecas: *sys/types.h*, *sys/ipc.h* e *sys/msg.h*.

A função ***msgget*** cria uma fila de mensagens e retorna o identificador associado a ela. Esse identificador será utilizado nas demais funções de controle e acesso. A sintaxe da função ***msgget*** é mostrada na Figura 6.3.

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>

int msgget (key, msgflg)
key_t key;
int msgflg;
```

FIGURA 6.3 - SINTAXE DA FUNÇÃO *MSGGET*

O parâmetro ***key*** é o nome da fila de mensagem. Esse nome é definido pelo usuário.

O parâmetro ***msgflg*** é uma combinação dos comandos: *IPC_CREAT* e *IPC_EXCL*, e um número de permissão denominado *mode*. O comando *IPC_CREAT* solicita a criação de uma fila com o nome ***key***. Uma vez criada, a fila terá a permissão de acesso definida por *mode*. Caso a fila já exista, *IPC_CREAT* será ignorado. O comando *IPC_EXCL*, quando usado em combinação com *IPC_CREAT*, força a função a retornar um erro quando a fila já existe. O número *mode* define que tipo de permissão será dada aos usuários da fila (dono, grupo e outros): somente leitura, somente escrita ou leitura e escrita.

A função ***msgctl*** efetua operações de controle de mensagens. Sua sintaxe é mostrada na Figura 6.4.

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>

int msgctl (msqid, cmd, buf)
int msqid, cmd;
struct msqid_ds *buf;
```

FIGURA 6.4 - SINTAXE DA FUNÇÃO *MSGCTL*

O parâmetro ***msqid*** é o identificador da fila retornado por *msgget*.

O parâmetro ***cmd*** é um dos seguintes comandos: *IPC_STAT*, *IPC_SET* ou *IPC_RMID*. O comando *IPC_STAT* coloca o valor corrente de cada membro da estrutura de dados associada com *msqid* na estrutura apontada por *buf*. Essa estrutura (*msquid_ds*) contém, entre outros elementos, a identificação do dono e do grupo, as permissões, o número de mensagens correntemente na fila e o número máximo de bytes permitidos na fila. O comando *IPC_SET* atribui à estrutura apontada por *buf*: a identificação do dono e do grupo, as permissões da fila associada a *msqid* e o número máximo de bytes na fila. O comando *IPC_RMID* remove a fila de mensagens especificada por *msqid* e destrói a estrutura de dados associada a ela.

A função ***msgsnd*** é utilizada para enviar mensagens para a fila identificada por *msqid*. A sintaxe de *msgsnd* é mostrada na Figura 6.5.

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>

int msgsnd (msqid, msgp, msgsz, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz, msgflg;
```

FIGURA 6.5 - SINTAXE DA FUNÇÃO MSGSND

O parâmetro ***msgp*** aponta para uma estrutura contendo a mensagem. A estrutura contém os seguintes elementos:

```
struct msgbuf
  {
  long  mtype;       /* tipo da mensagem */
  char  mtext [1];   /* texto da mensagem */
  }
```

O campo ***mtype*** é um inteiro maior que zero, que pode ser usado na seleção da mensagem recebida pelo processo receptor. O campo ***mtext*** é um *array* contendo a mensagem. As filas de mensagens podem ser utilizadas no envio de mensagens de diversos tipos: inteiros, arranjos, estruturas, etc. Se o tipo original da mensagem não for um arranjo de caracteres, pode-se fazer uma operação de *cast* na variável que contém a mensagem e transformá-la em um arranjo.

O parâmetro ***msgsz*** é o número de bytes ocupados pela mensagem. Neste número não está incluído o tamanho do campo *mtype*.

O parâmetro ***msgflg*** pode assumir os valores: IPC_NOWAIT ou 0. Se for especificado o comando IPC_NOWAIT, a função retorna -1 caso não seja possível o envio da mensagem. Se for especificado o valor 0, a função suspenderá a execução até a ocorrência de uma das seguintes condições:

- O fator que causou a suspensão não existe mais; nesse caso, a mensagem é enviada normalmente;

- O identificador *msqid* é removido do sistema; nesse caso, a função retorna -1;

- O processo recebeu um sinal que deve ser verificado; a mensagem não é enviada e o processo reassume a execução.

A função ***msgrcv*** lê uma mensagem da fila especificada por *msqid* e a coloca na estrutura apontada por *msgp*. A sintaxe de *msgrcv* é mostrada na Figura 6.6.

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>

int msgrcv (msqid, msgp, msgsz, msgtyp, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz;
int msgflg;
```

FIGURA 6.6 - SINTAXE DA FUNÇÃO MSGRCV

O parâmetro ***msgsz*** indica o tamanho máximo do texto da mensagem a ser recebida (*mtext*) em bytes.

O parâmetro ***msgtyp*** indica o tipo da mensagem a ser lida. Se *msgtyp* for igual a zero, a função retorna a primeira mensagem da fila; se *msgtyp* for maior que zero, a função retorna a primeira mensagem da fila do tipo *msgtyp*; se *msgtyp* negativo, a função retorna a primeira mensagem da fila com tipo menor ou igual ao valor absoluto de *msgtyp*.

O parâmetro ***msgflg*** pode ter os valores: IPC_NOWAIT, MSG_NOERROR ou 0. Se o comando IPC_NOWAIT for

especificado, a função *msgrcv* retorna imediatamente com o valor -1 caso não haja mensagem do tipo definido por *msgtyp* na fila. O comando MSG_NOERROR faz com que qualquer mensagem maior que *msgsz* seja truncada para *msgsz* bytes, sem retornar mensagem de erro. Se nenhum dos dois comandos for especificado (*msgflg* = 0) e *msgrcv* não encontra na fila a mensagem do tipo indicado, a função suspende a execução até que ocorra uma das seguintes condições:

- A mensagem do tipo esperado é encontrada na fila;
- O identificador *msqid* é removido do sistema; nesse caso, a função retorna -1;
- O processo recebeu um sinal que deve ser verificado; nesse caso, nenhuma mensagem é recebida e o processo retoma a execução.

Se *msgflg* é igual a zero e o número de bytes em *msgtext* é maior que *msgsz*, a função retorna -1 e a mensagem não é recebida.

6.2 PLATAFORMA DE TESTES

A plataforma de testes inclui dois tipos de programa, como mostra a Figura 6.7: um programa de teste na interface superior e um na interface inferior. Nos pontos **1, 2, 3 e 4** é feita uma amostragem dos parâmetros das primitivas e UDPs enviados pelo MMS; esses parâmetros são exibidos em tela e comparados aos parâmetros gerados pelo programa testador superior.

O programa de teste superior possui duas versões diferentes executando em uma mesma estação: uma versão age sempre como estação iniciadora, testando uma instância do MMS através do envio de primitivas *request* e recebimento de primitivas *confirm*; a outra age sempre como respondedora, testando uma instância do MMS através do recebimento de primitivas *indication* e envio de primitivas *response*.

A interface inferior tem como objetivo enviar as mensagens recebidas do MMS para uma fila de comunicação com o processo par e vice-versa. As primitivas LLC *request* recebidas do outro processo são mapeadas em primitivas LLC *indication*. Há também duas versões deste programa: uma para a estação iniciadora e outra para a respondedora.

A comunicação entre as duas "estações" é feita através de duas filas de mensagens. A fila **FIR** (Fila no sentido Iniciador/Respondedor) é utilizada para a transferência de mensagens do iniciador para o respondedor. A fila **FRI** (Fila Respondedor/Iniciador) é utilizada para a transferência de mensagens do respondedor para o iniciador.

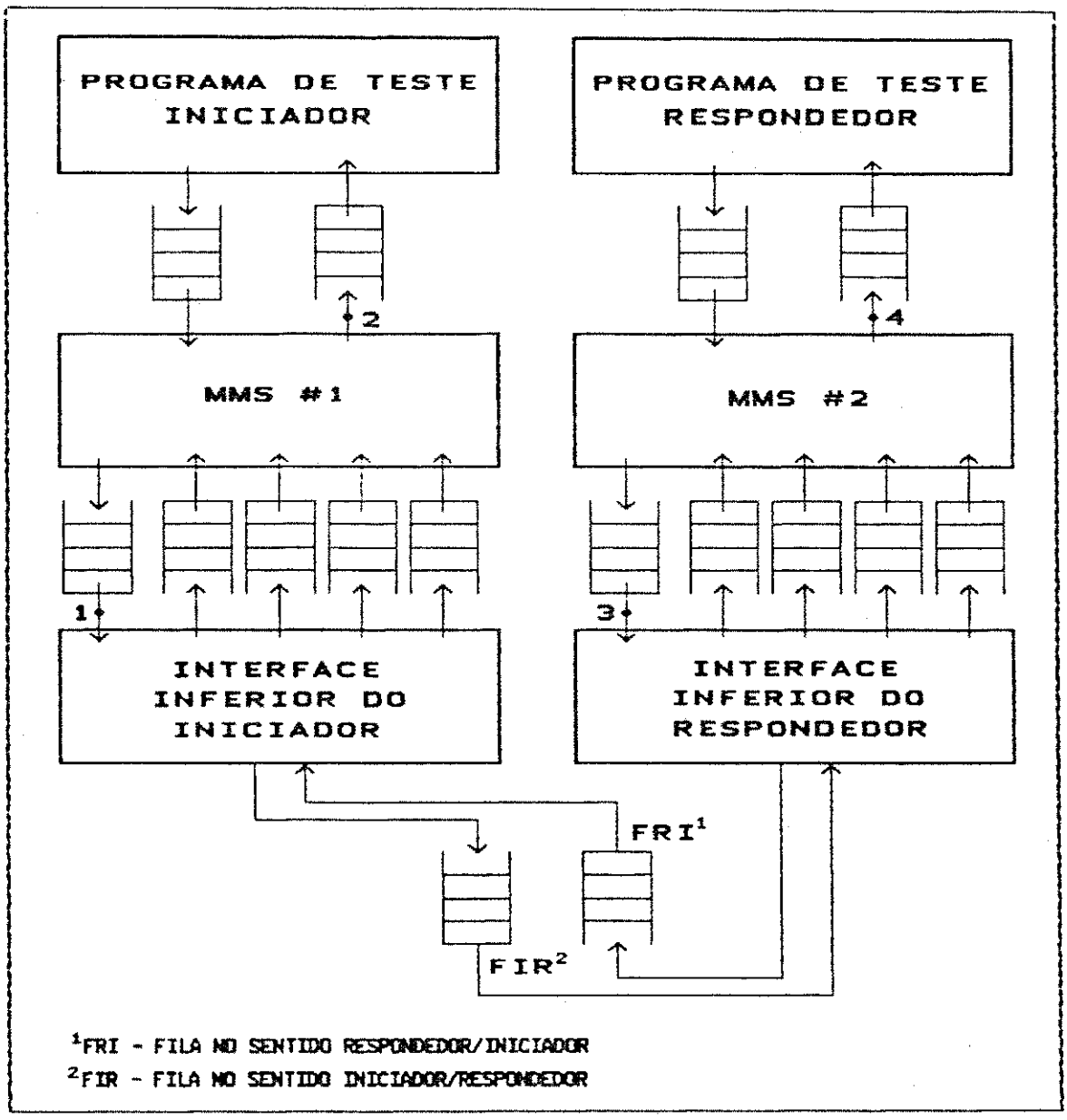
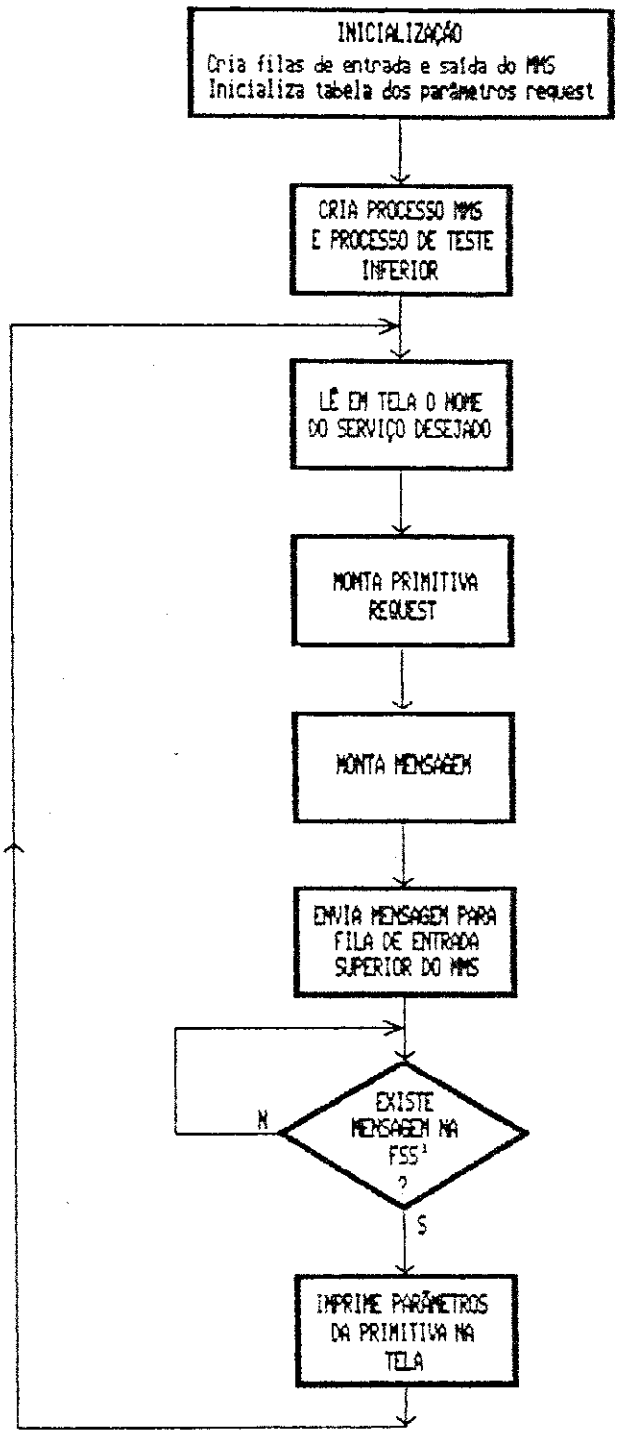


FIGURA 6.7 - MODELO DA PLATAFORMA DE TESTES

O fluxograma da Figura 6.8 ilustra o funcionamento do programa de teste iniciador. Este programa é o responsável pela criação de todas as filas de mensagens das interfaces superior e inferior do MMS, além de gerar os processos referentes ao MMS e à interface inferior.



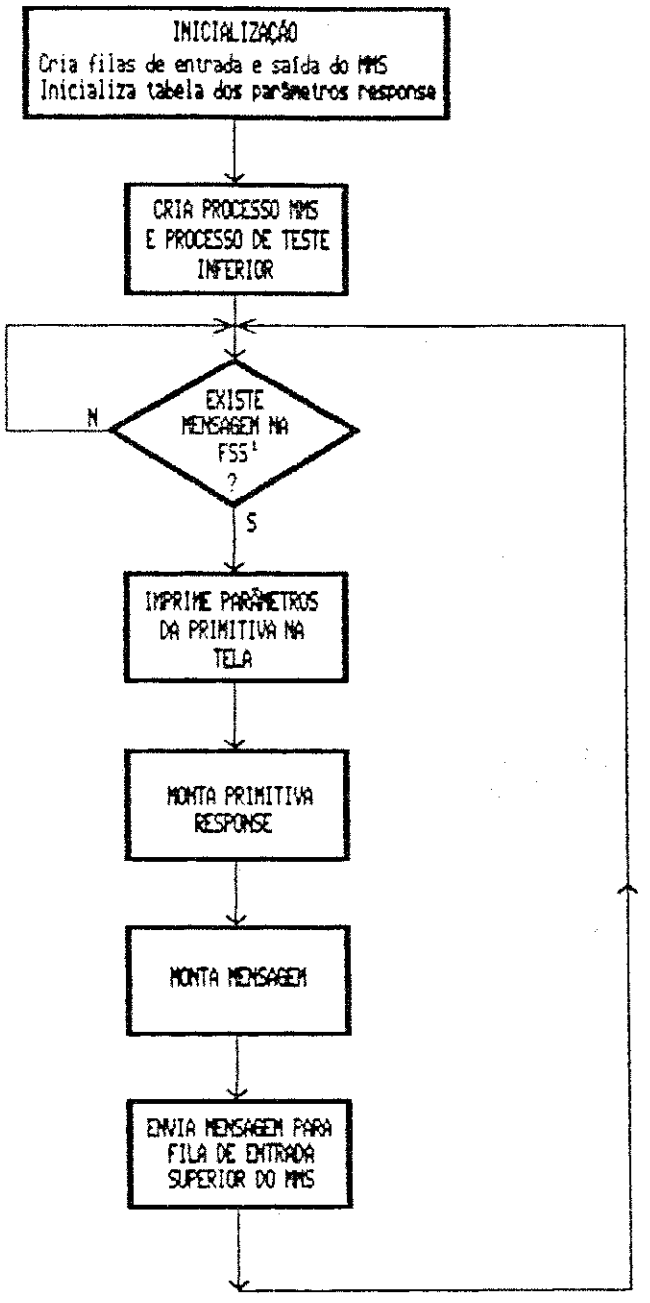
[1]FSS - FILA DE SAÍDA SUPERIOR DO MMS

FIGURA 6.8 - FUNCIONAMENTO DO PROGRAMA DE TESTE INICIADOR

Após ler via teclado o tipo de primitiva a ser enviado, o processo iniciador monta os parâmetros e envia a primitiva para a fila de entrada superior do MMS, passando, em seguida, a aguardar a chegada da resposta (primitiva *confirm*). A fila de saída superior do MMS é, então, lida continuamente até a chegada de mensagem. Após exibir em tela os parâmetros recebidos, o processo volta a ler o tipo de primitiva a ser enviado.

O fluxograma da Figura 6.9 ilustra o funcionamento do programa de teste respondedor.

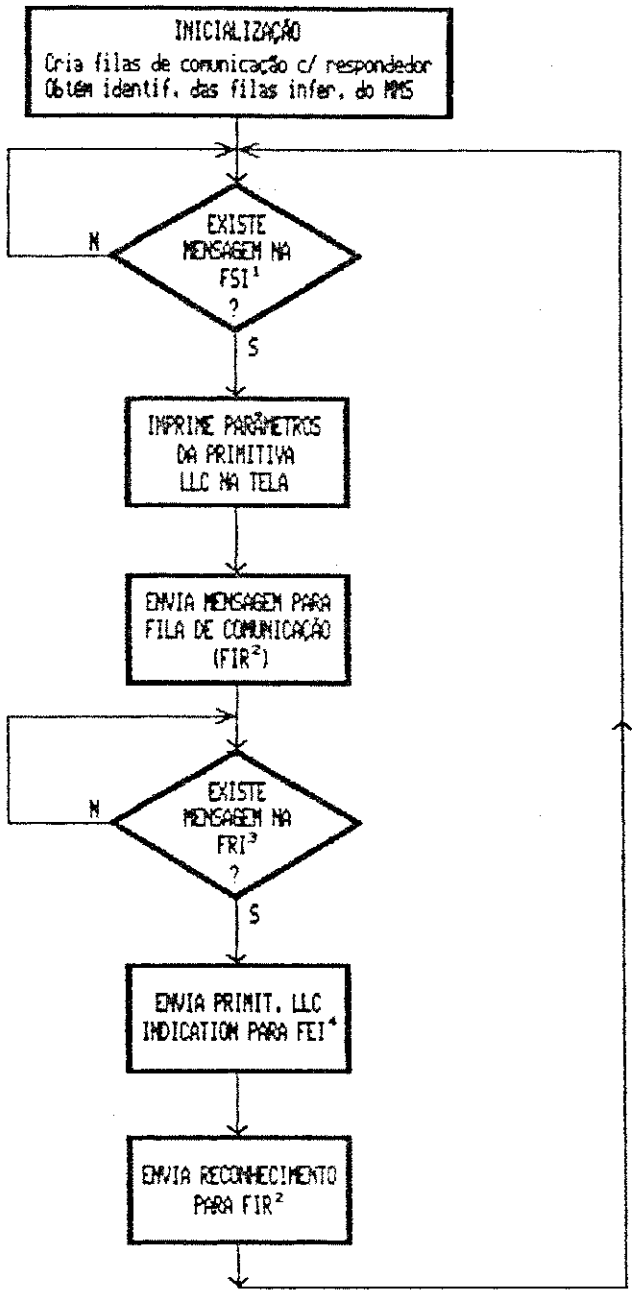
Da mesma forma que o programa de teste iniciador, este programa é o responsável pela criação de todas as filas de mensagens e da geração dos processos referentes ao MMS e à interface inferior. O processo respondedor lê continuamente a fila de saída superior do MMS até a chegada de uma mensagem contendo uma primitiva *indication*. Quando a primitiva é recebida, seus parâmetros são exibidos em tela e o processo efetua a montagem da primitiva de resposta (*response*), enviando-a para a fila de entrada superior do MMS. Em seguida, o processo passa a aguardar a chegada de nova primitiva (*indication*).



[1]FSS - FILA DE SAÍDA SUPERIOR DO MMS

FIGURA 6.9 - FUNCIONAMENTO DO PROGRAMA DE TESTE RESPONDEDOR

O programa de interface inferior do iniciador cria as filas FIR e FRI. O funcionamento do processo é ilustrado no fluxograma da Figura 6.10.



[1]FSI - FILA DE SAÍDA INFERIOR DO MMS

[2]FIR - FILA DE COMUNICAÇÃO NO SENTIDO INICIADOR/RESPONDEDOR

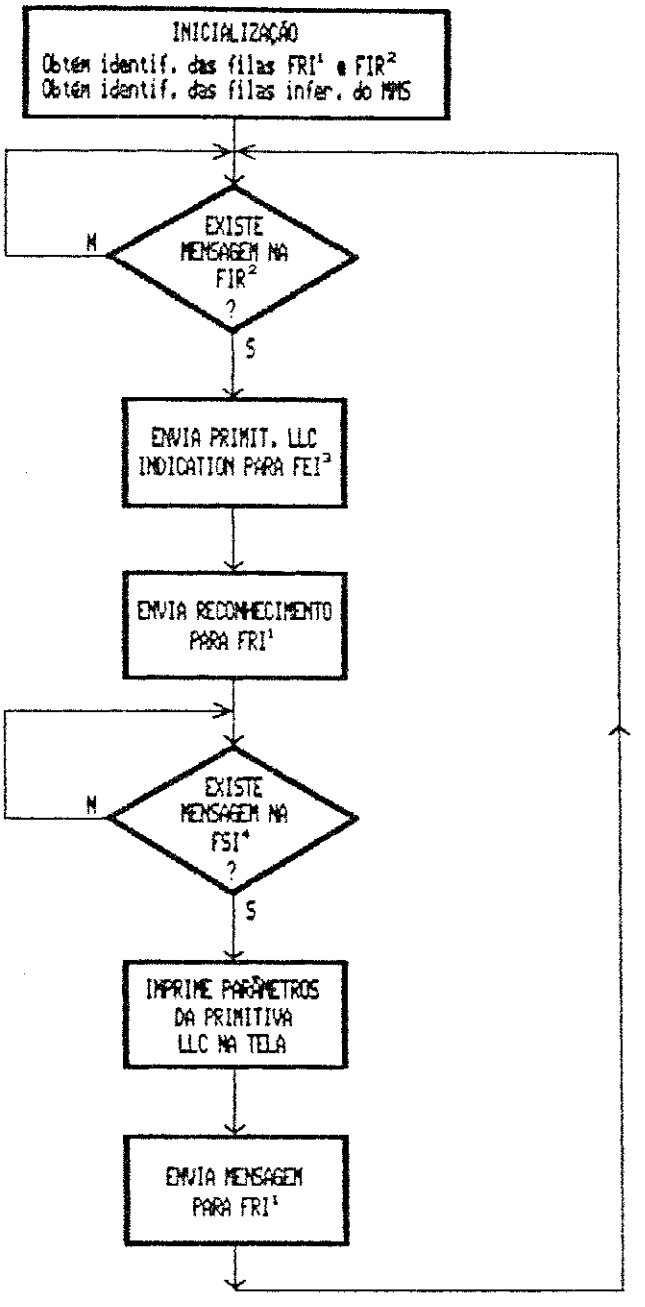
[3]FRI - FILA DE COMUNICAÇÃO NO SENTIDO RESPONDEDOR/INICIADOR

[4]FEI - FILA DE ENTRADA INFERIOR DO MMS

FIGURA 6.10 - FUNCIONAMENTO DA INTERFACE INFERIOR DO INICIADOR

Como se pode observar na Figura 6.10, a fila de saída inferior do MMS é lida continuamente até a chegada de uma mensagem; quando isso ocorre, os parâmetros da PDU MMS contida na primitiva LLC são exibidos em tela e a mensagem é colocada na FIR. O processo passa, então, a aguardar a resposta, lendo a fila FRI. As primitivas LLC *request* lidas dessa fila são alteradas para *indication* e enviadas para a fila de entrada inferior do MMS; as primitivas LLC *request* lidas da fila de saída inferior do MMS não sofrem alteração, sendo enviadas diretamente para a fila FIR.

O funcionamento do programa de interface inferior do respondedor é ilustrado no fluxograma da Figura 6.11. Este processo também é cíclico: ele aguarda a chegada de mensagens através da fila FIR. As primitivas LLC *request* recebidas são alteradas para *indication* e enviadas para a fila de entrada inferior do MMS. Em seguida, o processo passa a rastrear a fila de saída inferior do MMS até a chegada da resposta, que é enviada para a fila FRI sem alterações.



[1]FRI - FILA DE COMUNICAÇÃO NO SENTIDO RESPONDEDOR/INICIADOR

[2]FIR - FILA DE COMUNICAÇÃO NO SENTIDO INICIADOR/RESPONDEDOR

[3]FEI - FILA DE ENTRADA INFERIOR DO MMS

[4]FSI - FILA DE SAÍDA INFERIOR DO MMS

FIGURA 6.11 - FUNCIONAMENTO DA INTERFACE INFERIOR DO RESPONDEDOR

6.3 RESULTADOS OBTIDOS

Os resultados obtidos dos testes foram considerados satisfatórios, em relação aos objetivos descritos no início deste capítulo.

Durante os testes, acompanhou-se tanto o funcionamento interno do MMS como a transferência das primitivas de serviços entre os processos envolvidos. Observou-se que a implementação funcionou conforme o esperado e as primitivas observadas foram transportadas entre os processos mantendo-se a integridade e coerência dos parâmetros gerados pelos programas de teste iniciador e respondedor.

A seqüência de testes incluiu a abertura de uma associação, a troca de primitivas das unidades de Acesso a Variáveis e Gerenciamento de Programa, e o fechamento da associação, de forma negociada.

Observou-se apenas o comportamento do MMS para uma única associação pendente, não para várias, pois seria necessária a execução de mais instâncias do MMS, do programa de teste superior e do programa de teste inferior, levando a uma sobrecarga do sistema.

CAPITULO 7

CONSIDERAÇÕES FINAIS

Pode-se destacar dois aspectos para discussão com relação à implementação desenvolvida: processos e estruturas de dados.

Na estrutura com apenas um processo, todo o gerenciamento da camada é centralizado. Esse gerenciamento inclui: controle de associações, criação e acesso às estruturas de dados, alocação de *buffers*, leitura e escrita em filas de mensagens, etc. A vantagem dessa estrutura é a capacidade de controlar um grande número de serviços pendentes, distribuídos nas associações (nesse caso, criaram-se condições para manter até 320 serviços nas Tabelas de Estados MMPM e ORPM). Entretanto, pelo fato de só haver um processo, não é possível tratar em paralelo várias mensagens, e isso pode comprometer o tempo de resposta do protocolo às solicitações de serviço.

Uma alternativa para esse problema é a distribuição das tarefas em mais de um processo. Um processo principal encarrega-se da obtenção das mensagens nas interfaces de entrada superior e inferior, criação das estruturas de dados e alocação de *buffers*. Para cada serviço solicitado ao MMS, o processo principal gera um outro processo, responsável pelo tratamento das mensagens (incluindo o acesso às estruturas de dados) e envio de mensagens para as filas de saída superior e inferior. Através desse meca-

nismo, o protocolo poderia tratar vários serviços ao mesmo tempo, bastando para isso gerar um processo para cada serviço. Entretanto, seria necessária a utilização de mecanismos de comunicação interprocessos para permitir a troca de mensagens entre o processo principal e cada processo de tratamento de serviços. Esses mecanismos, representados pelas filas de mensagens, memória compartilhada, semáforos, etc, são fatores que sobrecarregam o sistema e, portanto, afetam o desempenho do protocolo. Além disso, o número de serviços que pode ser mantido pelo protocolo reduz-se para apenas algumas dezenas [ISO/FTAM,1990], pois o sistema tem um limite de eficiência no gerenciamento de processos em paralelo: cada processo de tratamento de serviços gerado representa um aumento no número de alocações e acesso à memória.

A princípio, não é possível definir qual das estruturas é mais eficiente, sendo necessário, para isso, um trabalho de avaliação de cada uma.

A definição das estruturas de dados é feita de forma a permitir a inclusão de novos serviços MMS, que dêem suporte às funções de um CLP classes 4 e 5. Com exceção da primitiva *CANCEL*, toda primitiva a ser incluída exige apenas a introdução de sua estrutura no arquivo de definição das primitivas MMS e a alteração do(s) bit(s) correspondente(s) nos arranjos de CBBs vertical e horizontal, que estão armazenados como valores *default* do MMS. A primitiva *CANCEL* exige também a inclusão de nós nas Tabelas de Tran-

sição de Estados e Funções de Saída e a codificação dos testes de predicado e funções de saída correspondentes.

Os recursos avançados de sistema operacional exigidos pela implementação são: filas de mensagens e criação de processos em paralelo. Portanto, o sistema operacional utilizado para execução deste MMS deve prover esses recursos, como é o caso do Executivo [TURNELL,1990].

O *hardware* utilizado poderá limitar principalmente:

- O número máximo de serviços pendentes por associação e o número máximo de associações ativas ao mesmo tempo, pois a dimensão das Tabelas de Estados MMPM e ORPM é obtida pela multiplicação desses números; assim, uma pequena variação em um dos números causa uma variação considerável na dimensão da tabela.

- As alocações de memória para armazenar as mensagens recebidas, as primitivas e as UDPs.

No ambiente de teste utilizado alocou-se espaço para o gerenciamento de 10 serviços pendentes por associação, com um máximo de 32 associações ativas. O limite de associações ativas foi definido com base no número máximo de estações (32) no segmento, especificado pelo Mini-MAP [GM/MAP,1988]. O limite de serviços pendentes por associação foi estabelecido como um valor suficiente para esta implementação e pode ser variado conforme a aplicação em que for utilizado. Entretanto, um aumento excessivo deste

valor pode aumentar muito o tamanho das Tabelas de Estados (ORPM e MMPM) e, conseqüentemente, aumentar consideravelmente o tempo de busca em cada tabela.

Como já foi dito no Capítulo 7, o protocolo comportou-se como esperado para as situações de teste a que foi submetido. Evidentemente, a validação completa de um protocolo é uma tarefa complexa e demorada e, por isso, restringiu-se a abrangência dos testes a serviços considerados relevantes: serviços de gerenciamento de contexto e serviços confirmados.

Um servidor MMS possui dois níveis de funcionalidade: o primeiro nível diz respeito ao dispositivo virtual (VMD); o segundo nível diz respeito ao dispositivo real. A implementação completa de um servidor deve incluir, além do VMD, o mapeamento entre o VMD e o dispositivo real (nesse caso, um CLP) representado pelo VMD.

A implementação do primeiro nível consiste da especificação dos serviços e do protocolo MMS de acordo com [EIA/MMS,1987], direcionados para um CLP seguindo o documento [IEC/PCMS,1989]. A implementação do segundo nível consiste do mapeamento entre esses serviços e os serviços de um CLP real. Como existem diversos CLPs de fabricantes diferentes (e, portanto, com características específicas diferentes), é necessária a seleção do dispositivo de um determinado fabricante e o estudo de suas funções e seu mecanismo. Tudo isso torna extenso o trabalho de implementação e por essa razão optou-se por desenvolvê-lo em duas fases: a primeira fase, correspondente ao primeiro nível de

funcionalidade, foi desenvolvida nesta implementação; a implementação do segundo nível faz parte de um outro trabalho a ser desenvolvido no âmbito do GRC/UFPB (segunda fase).

ANEXO 1

OS SERVIÇOS MMS

Neste anexo descreve-se todos os serviços MMS por unidade funcional, de acordo com [EIA/MMS,1987].

GERENCIAMENTO DE CONTEXTO

- INITIATE

  Iniciar um diálogo através da definição do contexto MMS e negociar opções e limites a serem usados nesse diálogo.

- CONCLUDE

  Terminar um diálogo de forma normal.

- ABORT

  Terminar um diálogo de forma anormal.

- CANCEL

  Cancelar um serviço pendente.

- REJECT

  Informar a ocorrência de erro de protocolo.

SUPORTE AO VMD

- STATUS

  Obter o estado de um VMD remoto.

- UNSOLICITED STATUS

  Receber espontaneamente o estado de um VMD remoto.

- GET NAME LIST

  Obter uma lista de nomes de objetos definidos no VMD.

- IDENTIFY

  Obter informações de identificação do equipamento remoto.

- RENAME

  Mudar o identificador de um objeto.

ACESSO A VARIAVEIS

- READ

  O cliente lê o conteúdo de uma ou mais variáveis definidas no VMD.

- WRITE

  O cliente muda o conteúdo de uma ou mais variáveis definidas no VMD.

- INFORMATION REPORT

  O servidor envia ao cliente informação do valor de uma ou mais variáveis.

- DEFINE SCATTERED ACCESS

  Cria no VMD um objeto de acesso do tipo **variáveis dispersas**, as quais podem ser nomeadas, não nomeadas ou dispersas, em qualquer combinação.

- DEFINE NAMED TYPE

  Armazena a especificação de tipo com nome para uso em uma subseqüente definição, no VMD, de objetos do tipo **variável nomeada** e/ou **tipo com nome.**

- DEFINE NAMED VARIABLE

  Cria no VMD um objeto de acesso do tipo **variável com nome.**

- DEFINE NAMED VARIABLE LIST

  Cria no VMD um objeto de acesso do tipo **lista de variáveis com nome**, contendo variáveis nomeadas, não nomeadas ou dispersas, em qualquer combinação.

- GET NAMED TYPE ATTRIBUTES

  Solicita os atributos de um objeto de acesso **tipo com nome.**

- GET VARIABLE ACCESS ATTRIBUTES

  Solicita ao VMD os atributos de objetos do tipo **variáveis nomeadas** ou **variáveis não nomeadas**, ou a descrição de tipo derivado de um objeto **variáveis dispersas.**

- GET NAMED VARIABLE LIST ATTRIBUTES

  Solicita os atributos de um objeto de acesso do tipo **lista de variáveis com nome.**

- GET SCATTERED ACCESS ATTRIBUTES

  Solicita ao VMD atributos de **variáveis dispersas.**

- DELETE VARIABLE ACCESS

  Suprime um ou mais objetos de acesso do tipo **variáveis nomeadas** ou **variáveis dispersas**.

- DELETE NAMED VARIABLE LIST

  Suprime uma ou mais listas de variáveis com nome.

- DELETE NAMED TYPE

  Suprime um ou mais tipos com nome.

GERENCIAMENTO DE EVENTOS

- DEFINE EVENT CONDITION

  Cria uma condição de evento no VMD.

- DEFINE EVENT ACTION

  Cria uma ação de evento no VMD.

- DEFINE EVENT ENROLLMENT

  Adiciona um cliente solicitante ou um cliente subordinado à lista de usuários que receberão notificação de evento resultante de uma condição de evento especificada.

- DELETE EVENT ACTION

  Suprime uma ou mais ações de eventos definidas no VMD.

- DELETE EVENT CONDITION

  Suprime uma ou mais condições de eventos definidas no VMD.

- DELETE EVENT ENROLLMENT

  Suprime um ou mais registros de eventos no VMD.

- GET EVENT ACTION ATTRIBUTES

  Solicita os atributos de uma ação de evento definida no VMD.

- GET EVENT CONDITION ATTRIBUTES

  Solicita atributos de uma condição de evento do VMD.

- GET EVENT ENROLLMENTS

  Solicita uma lista de registros de notificações de evento que satisfaçam a determinadas condições.

- GET ALARM SUMMARY

  Solicita do VMD um sumário com o estado atual de determinadas condições de evento. O estado pode ser: *disabled*, *idle* ou *active*.

- GET ALARM ENROLLMENT SUMMARY

  Solicita do VMD um sumário com o estado atual de determinados registros de evento. O estado pode ser: *disabled*, *idle* ou *active*.

- REPORT EVENT ACTION STATUS

  Obtém o número de registros de eventos que especificam uma ação de evento definida no VMD.

- REPORT EVENT CONDITION STATUS

  Obtém informações do estado de condição de evento do VMD.

- REPORT EVENT ENROLLMENT STATUS

  Obtém estado de registro de notificação de evento do VMD.

- ALTER EVENT CONDITION MONITORING

  Altera os atributos de uma condição de evento do tipo *monitored.*

- ALTER EVENT ENROLLMENT

  Troca atributos de um registro de eventos referente a uma condição de evento do tipo *monitored.*

- TRIGGER EVENT

  Inicia um evento tipo *network triggered.*

- EVENT NOTIFICATION

  Recebe notificação de um VMD da ocorrência de transição de estado associada com uma condição de evento.

- ACKNOWLEDGE EVENT NOTIFICATION

  Notifica ao VMD que seu usuário reconheceu uma notificação de evento recebida do VMD.

- ATTACH TO EVENT CONDITION MODIFIER

  Solicita a um VMD que atrase a execução de um serviço solicitado até a ocorrência de uma condição de evento especificada.

GERENCIAMENTO DE SEMAFOROS

- DEFINE SEMAPHORE

  Criar semáforo do tipo *token* no MMS respondedor. O usuário respondedor é aquele que envia uma primitiva de serviço *response* (após o recebimento de uma primitiva de serviço *indication*).

- REPORT POOL SEMAPHORE STATUS

  Obter nome e estado de *tokens* nomeados controlados por um semáforo do tipo *pool*.

- REPORT SEMAPHORE ENTRY STATUS

  Obter estado detalhado da lista de solicitantes de um semáforo.

- REPORT SEMAPHORE STATUS

  Obter os estados resumidos de um semáforo.

- DELETE SEMAPHORE

  Suprimir um semáforo.

- ATTACH TO SEMAPHORE STATUS

  Atrasar um serviço solicitado no respondedor até que o controle de um semáforo seja garantido por este serviço.

- TAKE CONTROL

  Obter o controle de um semáforo.

- RELINQUISH CONTROL

  Liberar o controle de um semáforo.

GERENCIAMENTO DE INVOCAÇÃO DE PROGRAMA

- CREATE PROGRAM INVOCATION

  Juntar domínios em uma invocação de programa executando no servidor MMS.

- DELETE PROGRAM INVOCATION

  Suprimir uma invocação de programa existente no servidor.

- START

  Mudar o estado de uma invocação de programa de *idle* para *running* (ou seja, iniciar uma execução).

- STOP

  Mudar o estado de uma invocação de programa de *running* para *stopped* (ou seja, suspender a execução).

- RESUME

  Mudar o estado de uma invocação de programa de *stopped* para *running* (ou seja, reiniciar uma execução previamente suspensa).

- RESET

  Mudar o estado de uma invocação de programa de *stopped* ou *running* para *idle* (ou seja, colocar um programa num estado que permita o início de uma execução).

- KILL

  Mudar o estado de uma invocação de programa para *unrunnable* (ou seja, **matar** uma execução).

- GET PROGRAM INVOCATION ATTRIBUTE

  Solicitar os atributos associados com uma invocação de programa especifica.

COMUNICAÇÃO ENTRE OPERADORES

- INPUT

  Executa operações de entrada.

- OUTPUT

  Executa operações de saída.

GERENCIAMENTO DE "JORNAL"

- READ JOURNAL

  Ler um ou mais registros de um "jornal".

- WRITE JOURNAL

  Escrever um ou mais registros em um "jornal".

- INITIALIZE JOURNAL

  Definir o valor inicial de um ou mais registros de um "jornal" já existente como logicamente vazio.

- REPORT JOURNAL STATUS

  Solicitar número de registros de um "jornal".

GERENCIAMENTO DE DOMINIO

- INITIATE DOWNLOAD SEQUENCE

  O cliente instrui o servidor MMS para criar um domínio com nome e iniciar o processo de *download*.

- DOWNLOAD SEGMENT

  O servidor MMS transfere partes de informações de *download* do domínio para o cliente.

- TERMINATE DOWNLOAD SEQUENCE

  O servidor indica ao cliente que terminou o *download*.

- INITIATE UPLOAD SEQUENCE

  O cliente solicita ao servidor que inicie o processo de *upload* de um domínio específico.

- UPLOAD SEGMENT

  O cliente solicita ao servidor a transferência de partes de informações de *upload* de um determinado domínio.

- TERMINATE UPLOAD SEQUENCE

  O cliente solicita ao servidor que termine um *upload*.

- REQUEST DOMAIN DOWNLOAD

  O servidor MMS solicita a um servidor de arquivos subordinado que inicie a função de *download*.

- REQUEST DOMAIN UPLOAD

  O servidor MMS solicita a um servidor de arquivos subordinado que inicie a função de *upload*.

- LOAD DOMAIN CONTENT

  O cliente solicita ao servidor que carregue, em um domínio, um arquivo de seu próprio sistema de arquivos ou de um servidor de arquivos subordinado ao servidor MMS.

- STORE DOMAIN CONTENT

  O cliente solicita ao servidor que armazene o conteúdo de um domínio em um sistema de arquivos.

- DELETE DOMAIN

  O cliente solicita ao servidor MMS que suprima um domínio específico e coloque seus recursos disponíveis.

- GET DOMAIN ATTRIBUTE

  O cliente solicita ao servidor a lista de atributos de um domínio específico.

- OBTAIN FILE

  O cliente solicita ao servidor que obtenha, de um servidor de arquivos subordinado ou do próprio cliente, um arquivo específico para seu sistema de arquivos local.

GERENCIAMENTO DE ARQUIVOS

- FILE OPEN

  Abrir um arquivo para ser lido.

- FILE READ

  Transferir todo ou parte do conteúdo de um arquivo já aberto do servidor para o cliente.

- FILE CLOSE

  Fechar arquivo já aberto, liberando recursos associados.

- FILE RENAME

  Mudar o nome de um arquivo.

- FILE DELETE

  Suprimir um arquivo.

- FILE DIRECTORY

  Obter os nomes e atributos de um arquivo ou conjunto de arquivos do sistema de arquivos do servidor.

ANEXO 2

MAQUINAS DE ESTADOS

MMPM E ORPM

ESTADO ATUAL	TESTES DE PREDICADO	AÇÕES DE SAIDA	NOVO ESTADO
CLOSED	evento = X.REQ x = INITIATE	L_DATA_ACK_STATUS.request (pdu := NULL);	WAIT_NULL_ACK
CLOSED	evento = X.REQ x <> INITIATE x = serviço confirmado	L_DATA_ACK.request (pdu := REQUEST_PDU, x, parâmetros x.req);	WAIT_CONFM_ACK
CLOSED	evento = X.REQ x = serviço não confirmado classe ender. = INDIVIDUAL	L_DATA_ACK.request (pdu := REQUEST_PDU, x, parâmetros x.req);	WAIT_UNCON_ACK
CLOSED	evento = X.REQ x = serviço não confirmado classe ender. = REPLY_BUFFER	L_DATA_ACK.request (pdu := REQUEST_PDU, x, parâmetros x.req);	WAIT_REQ_UPDATE
CLOSED	evento = X.REQ x = serviço não confirmado classe ender. = GROUP	L_DATA_ACK.request (pdu := REQUEST_PDU, x, parâmetros x.req);	CLOSED
NEED_RESP	evento = X.RESP classe ender. = INDIVIDUAL *status response* = OK	L_DATA_ACK.request (pdu := RESPONSE_PDU, x, parâmetros x.resp);	WAIT_RESP_ACK
NEED_RESP	evento = X.RESP classe ender. = INDIVIDUAL *status response* <> OK	L_DATA_ACK.request (pdu := ERROR_PDU, x, parâmetros x.resp);	WAIT_RESP_ACK

NEED_RESP	evento = X.RESP classe ender. = REPLY_BUFFER *status response* = OK	L_REPLY_UPDATE.request (pdu := RESPONSE_PDU, x, parâmetros x.resp);	WAIT_RESP_UPDATE
NEED_RESP	evento = X.RESP classe ender. = REPLY_BUFFER *status response* <> OK	L_REPLY_UPDATE.request (pdu := ERROR_PDU, x, parâmetros x.resp);	WAIT_RESP_UPDATE
NEED_RESP	evento = FORCE_CLOSE_REQ		CLOSED
CLOSED	evento = DATA_ACK_IND tipo pdu = REQUEST_PDU lsap c/ associação = FALSE associação aberta = TRUE x <> INITIATE x = serviço confirmado	X.indication (parâmetros x.req);	NEED_RESP
CLOSED	evento = DATA_ACK_IND ou DATA_IND ou RECEIVED_PDU_IND tipo pdu = REQUEST_PDU lsap c/ associação = FALSE associação aberta = TRUE x = serviço não confirmado	X.indication (parâmetros de x.req);	CLOSED
CLOSED	evento = DATA_ACK_IND tipo pdu = REQUEST_PDU lsap c/ associação = TRUE associação aberta = FALSE x = INITIATE	X.indication (parâmetros initiate.req);	NEED_RESP
CLOSED	evento = DATA_ACK_IND tipo pdu = REQUEST_PDU lsap c/ associação = FALSE associação aberta = TRUE x = INITIATE	REJECT_SENT.indication (razão:=ERRONEOUS_INITIATE); L_DATA_ACK.request (pdu := REJECT_PDU, ERRONEOUS_INITIATE);	CLOSED

CLOSED	evento = DATA_ACK_IND tipo pdu = REQUEST_PDU lsap c/ associação = TRUE associação aberta = FALSE x <> INITIATE	REJECT_SENT.indication (razão := NO_ASSOCIATION); L_DATA_ACK.request (pdu := REJECT_PDU, NO_ASSOCIATION);	CLOSED
CLOSED	evento = DATA_ACK_IND ou DATA_IND ou RECEIVED_PDU_IND tipo pdu = pdu inválida	REJECT_SENT.indication (razão); L_DATA_ACK.request (pdu := REJECT_PDU, razão);	CLOSED
WAIT_CONFM_RESP	evento = DATA_ACK_IND ou RECEIVED_PDU_IND tipo pdu = RESPONSE_PDU x <> INITIATE x <> CONCLUDE	X.confirm (result := SUCCESS, parâmetros x.resp);	CLOSED
WAIT_CONFM_RESP	evento = DATA_ACK_IND ou RECEIVED_PDU_IND tipo pdu = RESPONSE_PDU x = INITIATE	X.confirm (result := SUCCESS, param. initiate.resp); abre associação	CLOSED
WAIT_CONFM_RESP	evento = DATA_ACK_IND ou RECEIVED_PDU_IND tipo pdu = RESPONSE_PDU x = CONCLUDE	X.confirm (result := SUCCESS, param. conclude.resp); fecha associação	CLOSED
WAIT_CONFM_RESP	evento = DATA_ACK_IND tipo pdu = ERROR_PDU ou REJECT_PDU	X.confirm (result := ERROR, parâmetros primit.);	CLOSED
WAIT_NULL_ACK	evento = DATA_ACK_STATUS_IND status llc = OK	L_DATA_ACK.request (pdu := REQUEST_PDU, initiate, param. initiate);	WAIT_CONFM_ACK

WAIT_NULL_ACK	evento = DATA_ACK_STATUS_IND *status* llc <> OK	MESSAGE_STATUS.indication (*status* := *status* llc); INITIATE.confirm (result := LINK_ERROR, NULL);	CLOSED
WAIT_CONFM_ACK	evento = DATA_ACK_STATUS_IND *status* llc = OK	MESSAGE_STATUS.indication (*status* := OK);	WAIT_CONFM_RESP
WAIT_CONFM_ACK	evento = DATA_ACK_STATUS_IND *status* llc = UN ou *status* llc <> UN associação aberta = FALSE	MESSAGE_STATUS.indication (*status* := *status* llc); X.confirm (result := LINK_ERROR, NULL);	CLOSED
WAIT_CONFM_ACK	evento = DATA_ACK_STATUS_IND *status* llc <> OK *status* llc <> UN associação aberta = TRUE	MESSAGE_STATUS.indication (*status* := *status* llc); X.confirm (result := LINK_ERROR, NULL); fecha associação ABORT.indication (razão := LINK_ERROR);	CLOSED
WAIT_UNCON_ACK	evento = DATA_ACK_STATUS_IND *status* llc = OK ou *status* llc = UN ou *status* llc <> UN *status* llc <> OK associação aberta = FALSE	MESSAGE_STATUS.indication (*status* := *status* llc);	CLOSED
WAIT_UNCON_ACK	evento = DATA_ACK_STATUS_IND *status* llc <> OK *status* llc <> UN associação aberta = TRUE	MESSAGE_STATUS.indication (*status* := *status* llc); fecha associação ABORT.indication (razão := LINK_ERROR);	CLOSED

WAIT_RESP_ACK	evento = DATA_ACK_STATUS_IND *status* llc = UN	MESSAGE_STATUS.indication (*status* := UN);	NEED_RESP
WAIT_RESP_ACK	evento = DATA_ACK_STATUS_IND *status* llc = OK x = INITIATE *status response* = OK	MESSAGE_STATUS.indication (*status* := *status* llc); abre associação	CLOSED
WAIT_RESP_ACK	evento = DATA_ACK_STATUS_IND *status* llc = OK x = CONCLUDE *status response* = OK	MESSAGE_STATUS.indication (*status* := *status* llc); fecha associação	CLOSED
WAIT_RESP_ACK	evento = DATA_ACK_STATUS_IND *status* llc <> OK associação aberta = FALSE ou *status* llc = OK x = INITIATE *status response* <> OK ou *status* llc = OK x = CONCLUDE *status response* <> OK ou *status* llc = OK x <> INITIATE x <> CONCLUDE	MESSAGE_STATUS.indication (*status* := *status* llc);	CLOSED
WAIT_REQ_UPDATE	evento = REPLY_UPDATE_STATUS_IND	MESSAGE_STATUS.indication (*status* := *status* llc);	CLOSED
WAIT_RESP_UPDATE	evento = REPLY_UPDATE_STATUS_IND	MESSAGE_STATUS.indication (*status* := *status* llc);	CLOSED

WAIT_CONFM_RESP	evento = STATION_FAILURE_IND associação aberta = TRUE	fecha associação X.confirm (result := REMOTE_STATION_FAILURE, NULL); ABORT.indication (razão := REMOTE_STATION_FAILURE);	CLOSED
CLOSED	evento = STATION_FAILURE_IND associação aberta = TRUE	fecha associação ABORT.indication (razão := REMOTE_STATION_FAILURE);	CLOSED
CLOSED	evento = OBTAIN_REPLY_REQ	L_REPLY.request (pdu := NULL);	WAIT_REPLY
WAIT_REPLY	evento = REPLY_STATUS_IND *status* llc = OK pdu esperada = REQUEST_PDU pdu recebida = REQUEST_PDU x = serviço não confirmado	OBTAIN_REPLY.confirm (status := OK); RECEIVED_PDU.indication (pdu)	CLOSED
WAIT_REPLY	evento = REPLY_STATUS_IND *status* llc = OK pdu esperada = REQUEST_PDU pdu recebida = REQUEST_PDU x = serviço confirmado ou pdu recebida <> REQUEST_PDU	OBTAIN_REPLY.confirm (status := WRONG_PDU);	CLOSED
WAIT_REPLY	evento = REPLY_STATUS_IND *status* llc = OK pdu esperada = RESPONSE_PDU pdu recebida = RESPONSE_PDU *invoke_id* recebido = *invoke_id* esperado	OBTAIN_REPLY.confirm (status := OK); RECEIVED_PDU.indication (pdu)	CLOSED

WAIT_REPLY	evento = REPLY_STATUS_IND *status* llc = OK pdu esperada = RESPONSE_PDU pdu recebida = RESPONSE_PDU *invoke_id* recebido <> *invoke_id* esperado ou pdu recebida <> RESPONSE_PDU	OBTAIN_REPLY.confirm (status := WRONG_PDU);	CLOSED
WAIT_REPLY	evento = REPLY_STATUS_IND *status* llc <> OK	OBTAIN_REPLY.confirm (status := *status* llc);	CLOSED

ANEXO 3

MAPEAMENTO
DAS PRIMITIVAS E UDPS MMS
DEFINIDAS EM ASN.1

As primitivas e UDPs MMS foram definidas através de um mapeamento manual da notação ASN.1 [27] [28], para uma codificação em linguagem C. A definição da estrutura das primitivas e UDPs em notação ASN.1 encontra-se no documento MMS.

Os tipos principais definidos na sintaxe ASN.1 e utilizados na especificação do MMS são: *INTEGER*, *BOOLEAN*, *BIT STRING*, *OCTET STRING*, *SEQUENCE*, *SEQUENCE OF*, *CHOICE* e *VisibleString*. Além destes, o MMS define outros tipos, dos quais os principais são: *Identifier*, *Unsigned8*, *Unsigned16*, *Unsigned32*, *Integer8*, *Integer16* e *Integer32*.

O tipo ***INTEGER*** descreve um número inteiro e corresponde ao tipo *int* da linguagem C.

O tipo ***VisibleString*** indica uma variável composta de uma seqüência de caracteres, que podem ser: letras maiúsculas e minúsculas, números, sublinhado e outros símbolos. Na linguagem C, esse tipo é mapeado para um tipo arranjo de *char*.

O tipo ***BOOLEAN*** descreve variáveis binárias. Como a linguagem C não define um tipo que mapeie diretamente o tipo *BOOLEAN*, ele foi construído a partir do tipo *enum*, da seguinte forma:

```
typedef enum {FALSE, TRUE} BOOL;
```

O tipo ***BIT STRING*** descreve uma variável formada por uma seqüência de bits. A linguagem C define o tipo *bit-field*, que pode ser usado para mapear o *BITSTRING* e, como o MMS limita o tamanho máximo de uma variável deste tipo a 128 bits, a construção correspondente em linguagem C tem a seguinte forma:

```
typedef struct
  {
  unsigned bit: 128;
  } BSTRING;
```

O tipo ***OCTET STRING*** descreve uma variável constituída de uma seqüência de octetos. A definição em ASN.1 não especifica o número máximo de bytes em um *OCTET STRING* e, por isso, definiu-se o tipo em C com um limite de 64 bytes. O tipo resultante na linguagem C consiste de um arranjo de *bit-fields* com 64 elementos:

```
struct
 {
 unsigned bit: 8;
 } oct [64];

typedef oct [64] OCTSTRING;
```

O tipo ***SEQUENCE*** descreve uma estrutura constituída por um ou mais elementos de diversos tipos, inclusive *SEQUENCE*. O tipo correspondente em C é o *struct*. O exemplo da figura A.1 ilustra a construção em C codificada a partir de uma construção ASN.1 na definição da primitiva MMS *DownloadSegment.Response*.

```
Definição ASN.1:

     DownloadSegment-Response ::= SEQUENCE {
       loadData       [0] IMPLICIT OCTET STRING,
       moreFollows    [1] IMPLICIT BOOLEAN DEFAULT TRUE
     }

Definição em C:

     typedef struct {
       OCTSTRING    *info;
       BOOL         continua;
     } P_RES_SEG;
```

FIGURA A.1 - MAPEAMENTO DE SEQUENCE

O tipo ***SEQUENCE OF*** *<type>* indica que a variável é composta por vários elementos do tipo *type*, onde *type* pode ser qualquer tipo ASN.1. A linguagem C não possui um tipo que corresponda a *SEQUENCE OF*. Assim, o mapeamento é feito para uma lista encadeada, conforme o exemplo mostrado na figura A.2 para a primitiva MMS *GetNameList.Response*.

```
Definição ASN.1:

     GetNameList-Response ::= SEQUENCE {
       listOfIdentifier   [0] IMPLICIT SEQUENCE OF Identifier,
       moreFollows        [1] IMPLICIT BOOLEAN
     }

Definição em C:

     struct id_list {
       IDENTIFIER       identif;
       struct id_list   *prox;
     };

     typedef struct {
       struct id_list   *lista_identif;
       BOOL             continua;
     } P_RES_GNL;
```

FIGURA A.2 - MAPEAMENTO DE SEQUENCE OF

O tipo *CHOICE* descreve uma estrutura contendo um ou vários elementos de diversos tipos, inclusive *CHOICE*. Uma variável assim definida contém, num dado instante, apenas o valor de um dos tipos constituintes da estrutura. A construção correspondente em C consiste em uma *struct* contendo dois elementos: o primeiro indica qual o tipo que a variável assumirá; o segundo é uma *union*, contendo as opções definidas em *CHOICE*. O exemplo da figura A.3 ilustra o mapeamento do parâmetro *address*, utilizado nas primitivas da unidade funcional de acesso a variáveis.

```
Definição ASN.1:

      Address ::= CHOICE {
        numericAddress      [0] IMPLICIT Unsigned32,
        symbolicAddess      [1] IMPLICIT VisibleString,
        unconstrainedAddess [2] IMPLICIT OCTET STRING
      }


Definição em C:

      struct estrut_ender {
        enum tipo_endereco ender; /* indica se é end. numérico, simbólico ou informal */
        union {
          unsigned long end_numerico;
          char          end_simbolico [16];
          OCTSTRING     *end_informal;
        } ender_selec;
      };
```

FIGURA A.3 - MAPEAMENTO DE CHOICE

O tipo *Identifier* descreve uma variável do tipo *VisibleString* com um limite de 16 bytes. A definição deste tipo na linguagem C é a seguinte:

```
typedef char IDENTIFIER [16];
```

Os tipos ***Unsigned8***, ***Unsigned16*** e ***Unsigned32*** descrevem variáveis do tipo ASN.1 *INTEGER*, contendo números inteiros sem sinal. O tipo *Unsigned8* é um inteiro de 8 bits; o tipo *Unsigned16* é um inteiro de 16 bits e o tipo *Unsigned32* é um inteiro de 32 bits. Na linguagem C define-se os seguintes tipos correspondentes: *unsigned*, *unsigned* e *unsigned long*, respectivamente.

Os tipos ***Integer8***, ***Integer16*** e ***Integer32*** são do tipo ASN.1 *INTEGER* e referem-se a números inteiros de 8, 16 e 32 bits, respectivamente. Os tipos em C correspondentes são, respectivamente: *int*, *int* e *long* e seu mapeamento nos inteiros depende do compilador e do tipo de máquina utilizada.

REFERENCIAS BIBLIOGRAFICAS

[CHINTAMANENI,1988]

CHINTAMANENI, P. R. et al.; "On Fault Tolerance in Manufacturing Systems". *IEEE Network*, v. 2, n. 3, pp. 32-39, maio, 1988.

[DAIGLE,1988]

DAIGLE, J.N. et al.; "Communications for Manufacturing: an Overview". *IEEE Network*, v. 2, n. 3, pp. 6-13, maio, 1988.

[EIA/MMS,1987]

EIA, "Manufacturing Message Specification - Part 1: Service Specification; Part 2: Protocol Specification", Project 1393A, Draft 6, maio, 1987.

[EIA/NCMS,1987]

EIA, "Proposed Numerical Control Semantics for the Manufacturing Message Systems: Service and Protocol Standards", EIA-511 (DP-1393A) WD 4, abr., 1987.

[GM/MAP,1988]

GENERAL MOTORS, "MAP Specification - Version 3.0", ago., 1988.

[HALSALL,1988]

HALSALL, Fred - *Data Communications, Computer Networks and OSI*, Addison-Wesley, 1988.

[IEC/PCMS,1989]

IEC, "Programmable Controller Message Specification", SC 65A/WG 6/TF 7, jun., 1989.

[IEEE,1982]

IEEE PROJECT 802. *Draft IEEE Standard 802.4*: Token Passing Bus Access Method And Physical Layer Specifications. Draft D, dez., 1982.

[IEEE,1983]

IEEE PROJECT 802. *Draft IEEE Standard 802.2*: Logical Link Control. Draft D, nov., 1982, Módulo 128 Sequence Number Change, jul., 1983.

[ISO/ASN1,1986]

ISO, "Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)", ISO DIS 8824, ago., 1986.

ISO, "Open Systems Interconnection - Abstract Syntax Notation One (ASN.1) Basic Encoding Rules", ISO DIS 8825, ago., 1986.

[ISO/OSI,1984a]

ISO, "Information Processing Systems - Open Systems Interconnection - Basic Reference Model", ISO DIS 7498, 1984.

[ISO/OSI,1984b]

ISO, "Open Systems Interconnection - Network Service Definition-Ad1: Connectionless-mode Transmission", ISO 8348 DAD1 (CLNS), 1984."

ISO, "Open Systems Interconnection - Protocol for Providing the Connectionless-mode Network Service-Ad1: Provision of the Underlying Service Assumed by ISO 8473", ISO 8473 AD1, 1984.

[ISO/OSI,1984c]

ISO, "Open Systems Interconnection - End Systems to Intermediate Systems Exchange Protocol for use in Conjunction with ISO 8473", ISO 9542, 1984.

[ISO/OSI,1984d]

ISO, "Open Systems Interconnection - Transport Service Definitions", ISO 8072, 1984.

ISO, "Open Systems Interconnection - Connection Oriented Transport Protocol Specification", ISO 8073, 1984.

[ISO/OSI,1984e]

ISO, "Open Systems Interconnection - Basic Connection Oriented Session Service Definition", ISO 8326, 1984.

ISO, "Open Systems Interconnection - Basic Connection Oriented Session Protocol Definition", ISO 8327, 1984.

[ISO/OSI,1986a]

ISO, "Open Systems Interconnection - Connection Oriented Presentation Service Definition", ISO 8822, jun., 1986.

ISO, "Open Systems Interconnection - Connection Oriented Presentation Protocol Specification", ISO 8823, jun., 1986.

[ISO/OSI,1986b]

ISO, "Open Systems Interconnection - Application Layer Structure", ISO/TC97/SC21 N1494, 1986.

[ISO/OSI,1986c]

ISO, "Open Systems Interconnection - Service Definition for Common Application Service Elements - Part 2: Association Control", ISO DIS 8649, jun., 1986.

ISO, "Open Systems Interconnection - Protocol Definition for Common Application Service Elements - Part 2: Association Control", ISO DIS 8650, jun., 1986.

[ISO/OSI,1986d]

ISO, "Open Systems Interconnection - Directory Services and Protocols Specification", ISO DIS 9594/1-8, 1986.

[ISO/OSI,1986e]

ISO, "Open Systems Interconnection - Management Information Service Definition, Part 1: Overview; Part 2: Common Management Information Service Definition", ISO DP 9595, out., 1986.

ISO, "Open Systems Interconnection - Management Information Protocol Definition, Part 1: Overview; Part 2: Common Management Information Protocol", ISO DP 9596, out., 1986.

[ISO/OSI,1987]

ISO, "Open Systems Interconnection - File Transfer and Management, Part 1: General Introduction; Part 2: The Virtual Filestore; Part 3: The File Service Definition; Part 4: The File Protocol Specification", ISO DIS 8571 - 1-4, 1987.

[ISO/RCMS,1989]

ISO, "Robot Companion Standard to MMS", TC 184/SC 2/WG 6, fev., 1989.

[KUSIAK,1988]

KUSIAK, A. and HERAGU, S.S.; "Computer Integrated Manufacturing: a Structural Perspective". *IEEE Network*, v. 2, n. 3, pp. 14-22, maio, 1988.

[LIMA,1990]

LIMA, C.P.A. - "Um Modelo para Implementação de um Serviço e Protocolo de Transferência de Arquivos em Rede". Dissertação de Mestrado em Informática, UFPb, Campina Grande - PB, maio, 1990.

[MARTINS,1990]

MARTINS, Joberto S.B. and TURNELL, David J.; "Implementing Multi-Layer Protocol Architectures on a Front-End Processor". In: IV Latim American Congress on Automatics. México, 1990.

[McGUFFIN,1988]

Mc GUFFIN, L. J. et al.; "MAP/TOP in CIM Distributed Computing". *IEEE Network*, v. 2, n. 3, pp. 23-31, maio, 1988.

[MENDES,1989]

MENDES, M.J. - "Comunicação Fabril e o Projeto MAP/TOP"; IV EBAI, Argentina, jan., 1989.

[SPROESSER,1989]

SPROESSER, F.; "A Flexibilidade do CIM". *INFO*, n. 77, p. 32, jun., 1989.

[SUN,1990]

SUN - Manuais de sistema operacional, linguagem C e operação, 1990.

[TELEBRAS,1987]

TELEBRAS, Documentação Técnica do Projeto PP (Processador Preferencial), 1987.

[TURNELL,1990]

TURNELL, D. - "Desenvolvimento de uma Estação de Rede MAP para a Automação Industrial". Dissertação de Mestrado em Informática, UFPB, Campina Grande - PB, set., 1990.