Durval Carvalho de Avila Jacintho

Engenheiro Eletricista - Modalidade Eletrônica

Faculdade de Engenharia Elétrica - FEE

Universidade Estadual de Campinas - UNICAMP

Graduação em Dezembro de 1984

ASPECTOS DE ESPECIFICAÇÃO E IMPLEMENTAÇÃO DA ESTRUTURA DE MENSAGENS DE UM SISTEMA DIDÁTICO DO PROTOCOLO MMS

Este exemplar corresponde à redação final da tese defendida por Durval Carvalho de Avila Jacintho e aprovada pela comissão julgadora em 10/11/89

Dissertação apresentada à Faculdade de Engenharia Elétrica da UNICAMP com requisito parcial para obtenção do título de "Mestre em Engenharia Elétrica".

ORIENTADOR : Manuel de Jesus Mendes

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Novembro de 1989

Dedico à Marisa, com carinho,
pela compreensão e incentivo.

ÍNDICE

1. INTRODUÇÃO

A Especificação MMS ("Manufacturing Message Specification") [1] define um protocolo da camada de Aplicação do Modelo OSI [2], projetado para suportar a comunicação de mensagens entre equipamentos programáveis, no ambiente de manufatura integrada por computador (CIM) e Controle de Processos. Este protocolo faz parte do Projeto MAP ("Manufacturing Automation Protocol") [3], criado pela "General Motors" no final da década de 70.

O Projeto MAP tem por objetivo a interligação de sistemas computacionais de automação industrial, independentemente de fabricante e tipo de equipamento [4], e pode ser visto como um subconjunto do Modelo OSI, utilizando-se os protocolos já definidos por este [5]. Caracteriza-se como a aceitação de parte do Modelo OSI em ambiente industrial.

1.1. Objetivo

O objetivo principal desta tese é a especificação da estrutura de mensagens, trocadas na comunicação entre processos, na implementação do MMS. Esta implementação foi desenvolvida para o Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP (SISDI_MAP) [6], no Departamento de Computação e Automação Industrial da FEE-UNICAMP.

1.2. Histórico do MMS

A versão 2.1 do Projeto MAP [7], especificou para a camada de Aplicação, a partir de um subconjunto do Protocolo FTAM [8], um formato padrão na transmissão de mensagens no ambiente de manufatura, o MMFS - "Manufacturing Message Format Standards",

denominado também MFS, (lê-se Memphis), para comunicação entre equipamentos do chão de fábrica como robôs, controladores programáveis, etc [9]. A GM submeteu o MMFS a um comitê da EIA ("Electrical Industries Association") para que este fosse reescrito na sintaxe padrão ASN.1 ("Abstract Syntax Notation One") [10], surgindo assim o Padrão EIA RS-511, designado posteriormente como Protocolo MMS, em fase final de padronização na ISO (em 1987 status de "Draft International Standard"), padrão ISO/DIS 9506.

O Protocolo MMS adquiriu uma grande importância, tornando-se hoje, dentro da comunidade de automação, o componente chave da especificação MAP.

1.3. Características do MMS

A Especificação MMS define a interação de dispositivos programáveis num ambiente de manufatura, com características de operação em tempo real para controle de processos [11].

O Protocolo MMS permite :

- A definição de um padrão que, por sua generalidade, forneça um conjunto de comandos independente dos fabricantes de equipamentos de manufatura;
- Uma descrição completa, estruturada e funcional dos serviços de comunicação;
- A utilização de uma interface de alto nível que segue a terminologia ISO e se situa dentro do Modelo de Referência OSI [2];
- A adequação dos serviços MMS aos requisitos e características de um equipamento específico, através dos padrões próprios deste equipamento, denominados "Companion Standards";
- A verificação de conformidade à especificação dos serviços MMS [1].

O Protocolo MMS integra-se na distribuição funcional da figura 1.1., composta dos seguintes elementos :

- Programas Aplicativos (AP's), que são os usuários finais que se utilizem dos serviços MMS, oferecidos pelo sistema de comunicação;
- Interface de Programas de Aplicação (API) do MMS, definida como a interface entre o AP e o sistema de comunicação [12], que provê portabilidade e transparência no uso, pelos AP's, dos serviços MMS;
- Provedor de serviços MMS [13], que efetivamente executa a máquina de estados do Protocolo;
- Protocolo ACSE [14] que, sendo um serviço comum da camada de Aplicação para diversos ASE's ("Application Service Elements"), provê ao MMS o suporte de comunicação entre as Entidades de Aplicação (AE's), para o estabelecimento e liberação de associações;
- Camada de Apresentação [15], que provê o suporte da comunicação entre as AE's, para a execução dos serviços MMS, dentro de uma associação estabelecida.

Como se pode notar na distribuição funcional esquematizada na figura 1.1., o MMS se enquadra dentro do Modelo OSI, sendo definido como um ASE, e utiliza-se do ACSE e da Camada de Apresentação.

Entretanto o MAP vai além dos sete níveis definidos no OSI, apresentando para o MMS uma API, que pode ser considerada um sub-nível entre o usuário final (AP) e a Camada de Aplicação, provendo serviços e primitivas para a execução do AP.

Para os implementadores do MMS, a definição dos conceitos de usuário e provedor MMS, citada em [13], se torna importante, uma vez que define claramente a divisão funcional entre os AP's/API's e o Provedor MMS. Os primeiros tratam os serviços e seus parâmetros enquanto o Provedor MMS, situado na Camada de Aplicação, gerencia as máquinas de estado do Protocolo.

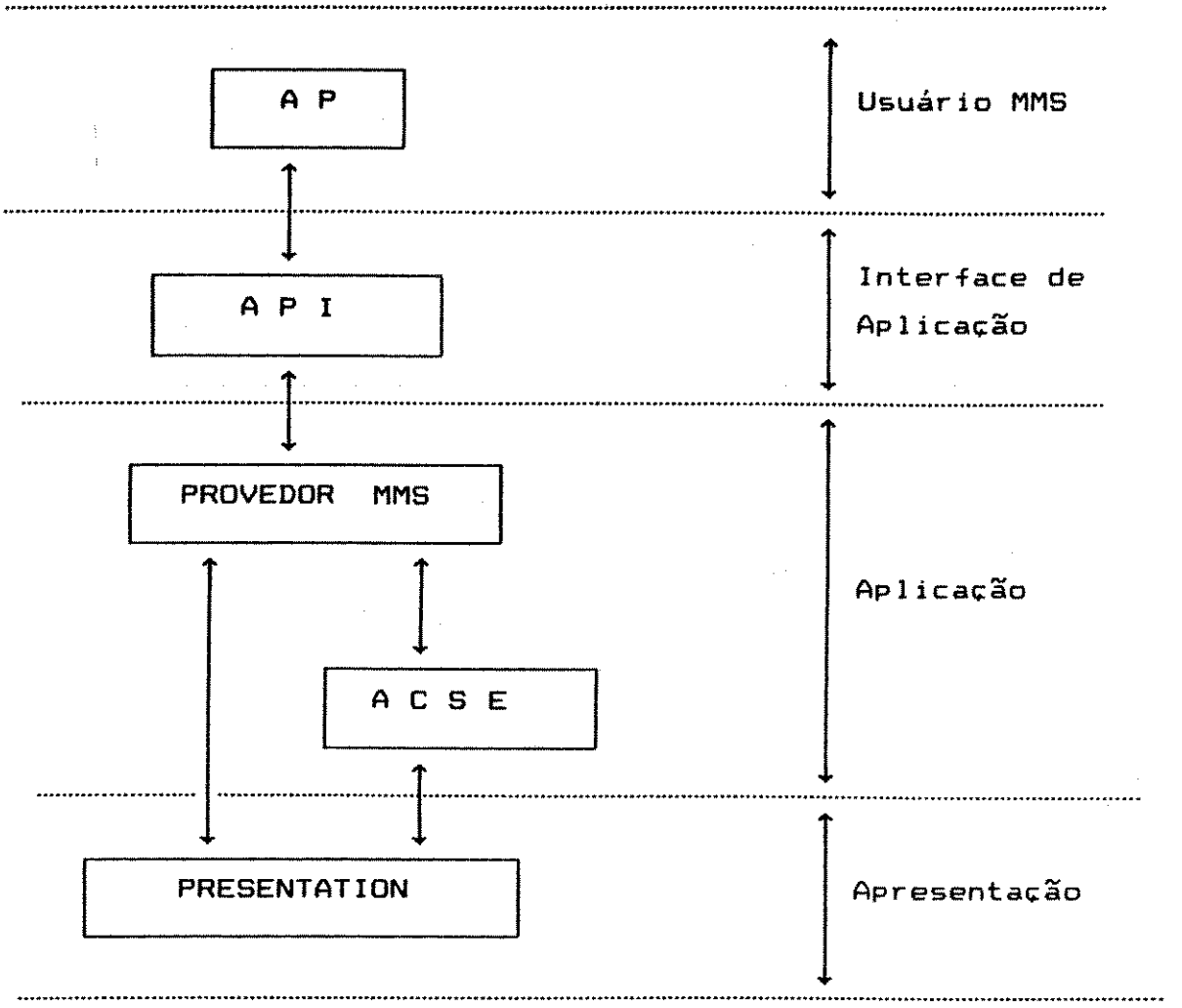

Figura 1.1 : A distribuição funcional do MMS

As setas de interligação na figura 1.1. entre Provedor MMS, ACSE e Apresentação refletem o mecanismo de interação entre estes elementos, no qual o MMS se utiliza dos serviços ACSE apenas para controle das conexões (estabelecimento, liberação e aborto), sendo a transferência de dados processada diretamente com a utilização dos serviços de Apresentação.

1.4. O Sistema Didático SISDI_MAP

O Sistema Didático está dividido funcionalmente em Software Básico e Software Aplicativo. O Software Básico é composto de um núcleo que provê características multi-tarefa e tempo real sobre uma máquina padrão IBM-PC [16]. O núcleo oferece uma interface padrão para acesso a seus serviços, constituindo uma "máquina virtual" para o Software Aplicativo na interface com o Software Básico da máquina hospedeira, conforme mostra a figura 1.2.

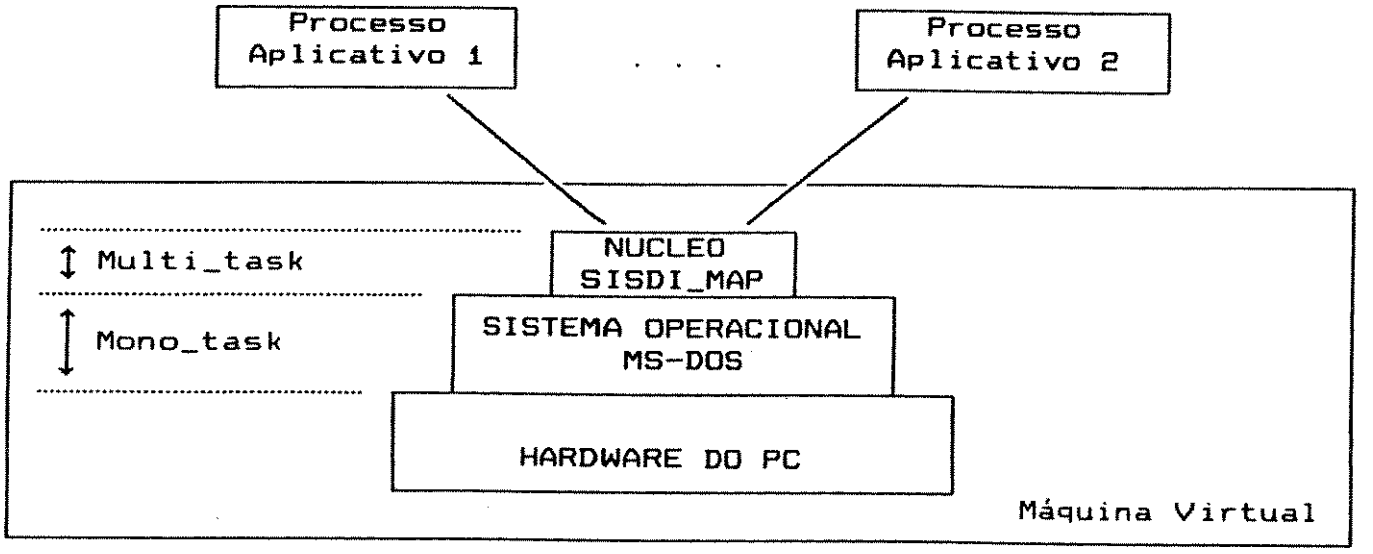

Figura 1.2 : Interface Software Aplicativo-Software Básico no SISDI_MAP

O Software Aplicativo do SISDI_MAP possui uma arquitetura funcional composta atualmente dos seguintes elementos, como mostra a figura 1.3. :

- Interface de Operação de Usuário;
- Programas Aplicativos;
- Interface de Programas Aplicação;
- Protocolo MMS;
- Protocolo ACSE;
- Simulador da Camada de Apresentação.

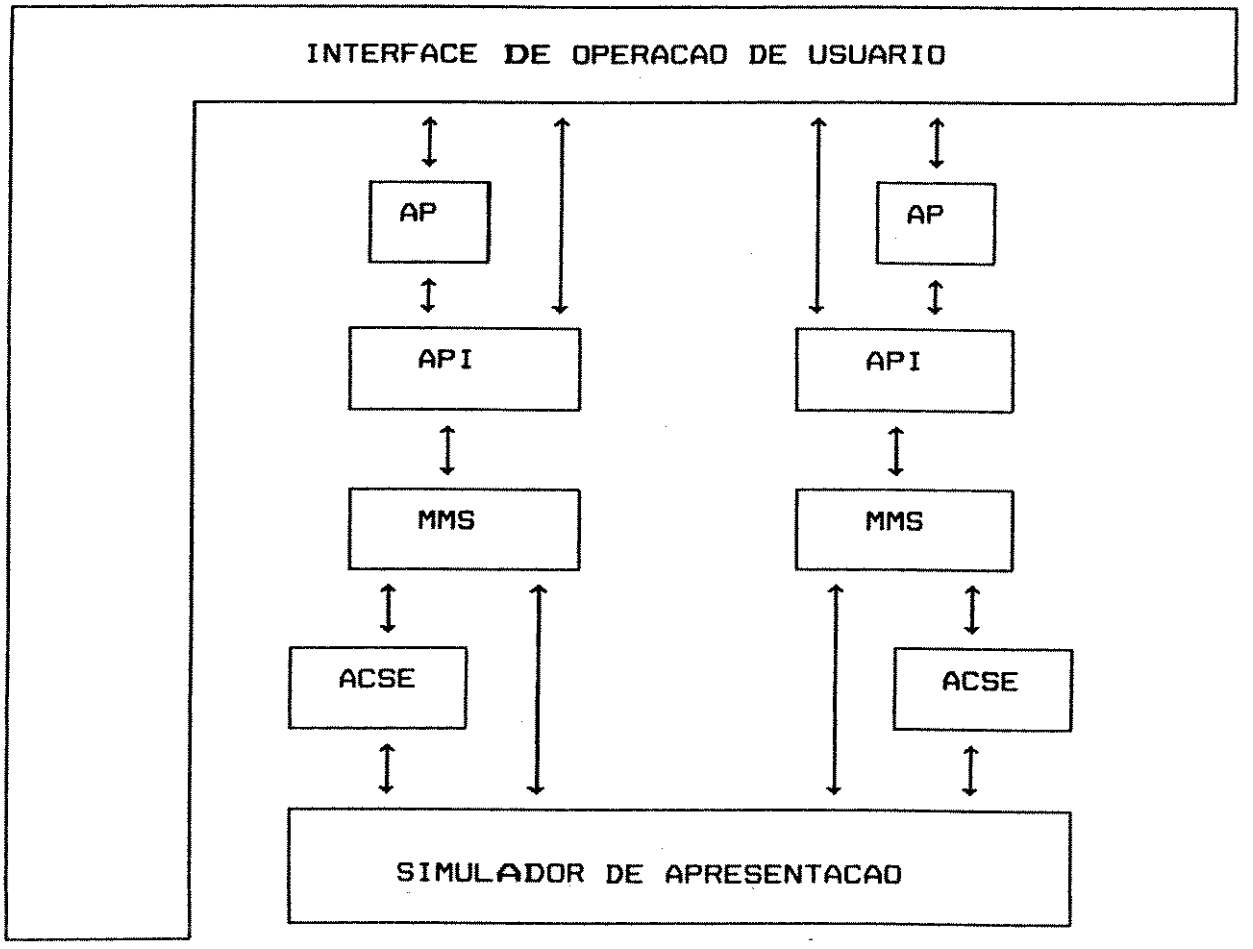

Figura 1.3 : Arquitetura Funcional do SISDI_MAP

A Interface de Operação de Usuário destina-se a prover uma interface de comunicação entre o operador do Sistema e o Protocolo MMS, através de janelas, "menus" e gráficos, de modo a fornecer informações ao usuário sobre a execução de cada serviço solicitado através dos protocolos [17]. O usuário pode executar uma seqüência de comandos MMS de modo interativo, interfaceando-se diretamente com a API, ou de modo automático, transferindo o controle da execução ao AP, como mostra a figura 1.3.

A Interface de Aplicação (API) provê uma biblioteca de funções padronizadas, utilizada pelos AP's, facilitando a programação destes, e fornecendo uma interface amigável ao MMS [18].

O MMS é responsável pela execução das funções do Provedor MMS, cuja implementação detalhada, apresentando as máquinas de estado de protocolo, encontra-se descrita em [19].

O ACSE é responsável pela execução das funções do Provedor ACSE, detalhado em [20].

O Simulador da Camada de Apresentação é responsável pelo suporte da comunicação fim-a-fim entre os protocolos deste nível através de simulação das primitivas de serviço de Apresentação e demais camadas do modelo OSI.

Esta tese especifica a implementação da estrutura de mensagens na comunicação entre a API, os Protocolos MMS e ACSE, o Simulador de Apresentação e a Interface de Operação. O Capítulo 2 descreve a "Especificação dos Serviços e Protocolo MMS", padrão ISO 9506, que define o MMS, fornecendo conceitos para a implementação desta estrutura.

No Capítulo 3 o modelo de implementação é apresentado, considerando-se os recursos providos pelo núcleo utilizado no SISDI_MAP. Esta capítulo detalha a estrutura das mensagens e também descreve a Simulação do Provedor de serviços de Apresentação, e sua operacionalidade no SISDI_MAP.

O Capítulo 4 apresenta um exemplo de execução dos serviços MMS, mostrando a utilização das mensagens pelos processos do SISDI_MAP.

No Capítulo 5 estão as conclusões obtidas no desenvolvimento desta tese, e o futuro do SISDI_MAP; e no Capítulo 6 estão as referências bibliográficas utilizadas.

O Capítulo 7 inclui dois anexos contendo as listagens da programação desenvolvida, completadas com comentários para maior clareza do seus conteúdos.

2. ESPECIFICAÇÃO DOS SERVIÇOS E DO PROTOCOLO MMS

O Documento ISO/DIS 9506 "Especificação de Mensagens da Manufatura" [1], define na parte 1 os serviços MMS e na parte 2 o Protocolo MMS, constituindo-se como a principal fonte de especificação para a implementação da estrutura de mensagens no SISDI_MAP.

A Especificação MMS define a interação entre os equipamentos que se comunicam no ambiente de manufatura, para solicitação e fornecimento de serviços, dentro do modelo Cliente-Servidor. Cada equipamento atuando como Servidor MMS, deve ser "visto" pelo Cliente conforme as características e funcionalidades deste equipamento. Assim, para formalizar esta interação, a Especificação MMS utiliza-se de um modelo abstrato de representação de um equipamento real, que se comporta como Servidor MMS, definindo o conceito de Dispositivo Virtual da Manufatura, o VMD ("Virtual Manufacturing Device").

2.1. A Especificação do VMD

A implementação de um Servidor MMS deve prover o mapeamento do Modelo do VMD com os aspectos funcionais do Equipamento real de manufatura. A orientação na escolha de um determinado mapeamento pode ser encontrada nos Padrões "Companion Standards", que descrevem aspectos de aplicação particulares para cada tipo de máquina [21].

Um VMD existe dentro do AP do Servidor MMS, constituindo um conjunto de recursos e funções associadas com o equipamento real da manufatura, que estão disponíveis ao Cliente MMS. Um AP pode definir zero um mais VMD's. Se nenhum VMD é definido, este não pode atuar como Sevidor MMS.

Cada VMD pode conter zero ou mais AE's. Cada AE dentro de um VMD representa um conjunto de capacidades de comunicação

utilizadas pelo AP. Quando um VMD contêm mais do que uma AE, ele contêm mais do que um protocolo de comunicação, como por exemplo um VMD que suporta separadamente o Protocolo de Terminal Virtual e o Protocolo MMS.

Um dado VMD é endereçado por uma ou mais AE's, cada uma das quais é endereçada através de um único Endereço de Apresentação, que identifica um ou mais PSAP's. A figura 2.1. mostra o relacionamento entre um AP atuando como um Servidor MMS, seus VMD's e PSAP's usados para acessá-lo.

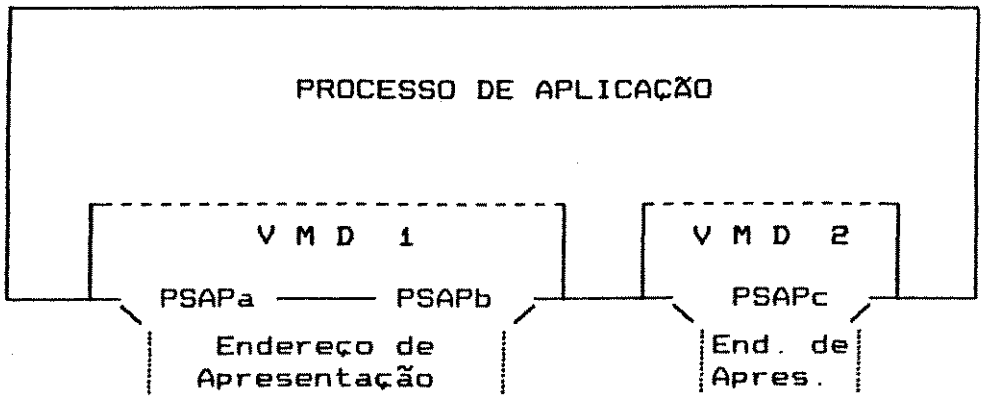

Figura 2.1 : O Processo de Aplicação Servidor MMS

2.1.1. A Estrutura do VMD

A estrutura de um VMD é dividida em (Ver figura 2.2.) :

- Função Executiva : Esta parte gerencia :
  - O acesso do Cliente MMS aos recursos do VMD;
  - Os aspectos de operação local do VMD que são visíveis para os seus Clientes;

- As capacidades e recursos do sistema como memória, processadores e portas de I/O.

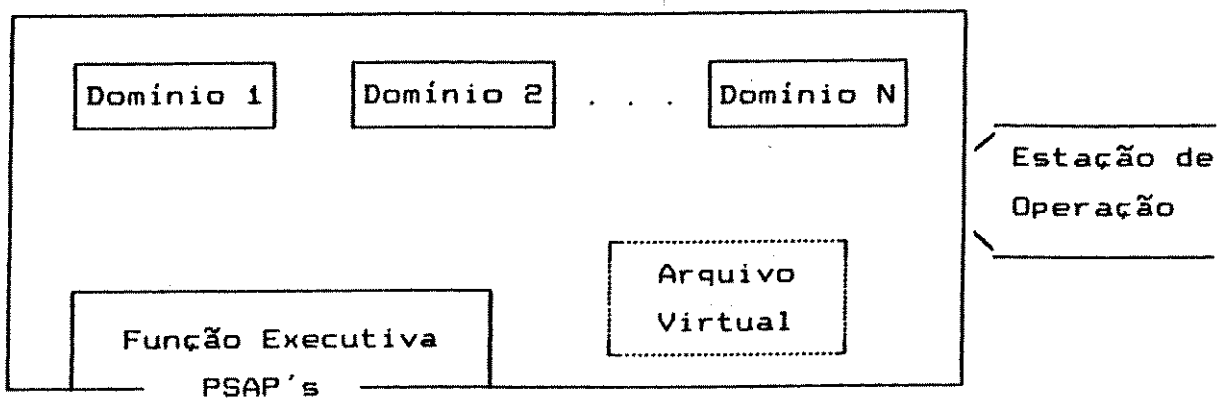

Figura 2.2 : A Estrutura do VMD

- Domínio : Representa o conjunto de recursos reais do VMD. Estes recursos podem ser recursos Hardware do equipamento (dispositivos de I/O, memória), ou recursos de Software (programas). A alocação de recursos de um VMD pode ser :

  - Estática : O Domínio é pré-definido dentro do Servidor MMS, e não pode ser deletado;
  - Dinâmica : O domínio é introduzido e removido do VMD através de serviços MMS.

  O seguinte exemplo mostra a função do Domínio no VMD : Uma Aplicação de Controle Numérico pode definir vários Domínios, um com recursos associados à parte de execução de programas, outro associado às funções de manuseio de materiais, outro domínio, estático, associado com procedimentos pré-definidos de "calibração" do equipamento.

- Arquivos Virtuais : é um objeto opcional no modelo. Consiste de um conjunto de arquivos rotulados, que quando presentes, atuam como um substituto para o armazenamento de dados e programas. Os arquivos num sistema podem ser usados por alguns dos serviços de Domínio e de Gerenciamento de Programas.
- Estação de Operação : Permite a operação local do VMD, e pode ser acessada pelo Cliente MMS através de primitivas de serviço específicas (Ver item 2.2.).

2.2. Os Serviços MMS

Os serviços MMS são aqueles providos pelo Provedor MMS, e podem ser usados por outros elementos de serviço do Nível de Aplicação ou por outros elementos dos Processos de Aplicação. A Especificação de serviços faz uso das diversas definições do Modelo OSI [2], da Estrutura da Camada de Aplicação [22] e da Definição de Serviços do ACSE [14].

A Especificação MMS define os serviços em Classes Funcionais, como mostra a Tabela 2.1.

Para execução de todos os serviços MMS definidos, é necessário o estabelecimento de um contexto entre os usuários MMS comunicantes. Este estabelecimento se faz através dos serviços da Classe Funcional de Gerenciamento de Contexto. A Tabela 2.2. apresenta os serviços de contexto e os demais serviços das Classes Funcionais do MMS, que podem ser executados dentro de um contexto MMS.

Uma implementação específica do MMS pode não necessariamente utiliza todos os serviços definidos. Assim, para que os serviços possam ser agrupados, além das Classes Funcionais, definiu-se os Blocos Construtivos de Conformidade (CBB's "Conformance Building Blocks"). Os CBB's podem ser "CBB's Verticais", que correspondem a sub-conjuntos apropriados de serviços na mesma Classe Funcional,

ou "CBB's Horizontais", que especificam serviços que estão relacionados a parâmetros que podem aparecem em mais de um serviço (em geral de Classe Funcional diferente). A Tabela 2.2. mostra, na coluna "Classe de Conformidade", os CBB's Verticais associados a cada serviço. Todos os serviços MMS apresentados possuem primitivas do tipo "request", "indication", "response" e "confirm", exceto os que estão assinalados com (*) e (**) na Tabela 2.2.

Classe Funcional	Descrição dos Serviços
Gerenciamento de Contexto	Iniciam, encerram e abortam a comunicação entre os usuários MMS, cancelam um serviço e notificam a ocorrência de uma PDU rejeitada
Suporte de VMD	Informam o estado de um VMD, o nome dos vários objetos definidos no VMD, a identificação do usuário MMS respondedor e requerem do respondedor a mudança do nome do identificador de um objeto
Gerenciamento de Domínio	Operam sobre domínios, executando funções como o carregamento de um programa em um equipamento ("download") e a obtenção de um programa carregado num equipamento ("upload"), armazenam conteúdos de domínios, deletam e solicitam atributos de domínios
Gerenciamento de Invocação de Programas	Criam, obtêm atributos, deletam uma Invocação de Programa e operam sobre ela através de funções como iniciação, parada, continuação e reiniciação. é um sub-conjunto simplificado do protocolo JTM ("Job Transfer and Manipulation") [23]
Acesso a Variáveis	Definem e deletam nomes de tipos, variáveis, lista de variáveis; obtêm informações sobre atributos destes elementos, lêm e escrevem sobre estes elementos de dados
Gerenciamento de Semáforos	Sincronizam, controlam e coordenam recursos compartilhados entre usuários MMS, através do controle e liberação de um semáforo, da definição e deleção dos semáforos, e informações dos tipos de semáforos e seus estados

[continua na página seguinte]

	[continuação]
Comunicação com a Estação de Operação	Obtêm dados ou entram com dados de ou para uma estação de operação do VMD
Gerenciamento de Eventos	Definem e gerenciam eventos num VMD e obtêm notificações de eventos ocorridos, condicionados às situações de tempo real. Os serviços incluem criar, alterar, deletar e obter informações sobre Condições de Evento e Ações de Evento
Gerenciamento de Jornal	Registram e recuperam informações ordenadas (Arquivo LOG), incluindo eventos, variáveis relativas aos eventos e textos explicativos ou comentários. Os serviços incluem leitura, escrita, iniciação e informação sobre o estado do Jornal
Gerenciamento de Arquivos	Provêm funções necessárias para leitura de arquivos contendo programas de controle e dados dos sistemas armazenadores de arquivos (nos equipamento de controle e servidores de arquivo), e para gerenciar os sistemas de arquivos. Estes serviços são opcionais e representam um sub-conjunto do Protocolo FTAM [8], que deu origem ao MMS.

Tabela 2.1 : Classes Funcionais dos Serviços MMS

Classe Funcional	Classe de Conformidade	Serviço
Gerenciamento de Contexto	CON1, CON2	Initiate
	CON1, CON2	Conclude
	MMS1	Abort (*)
	CON3	Cancel
	MMS1	Reject (**)
Suporte de VMD	VMD1, MMS2	Status
	VMD1	Unsolicited_status (*)
	MMS2	Get_name_list
	MMS2	Identify
	VMD2	Rename
Gerenciamento de Domínio	DOM1, DOM3	Initiate_down_load_sequence
	DOM1, DOM3	Down_load_sequence
	DOM1, DOM3	Terminate_down_load_sequence
	DOM2, DOM4	Initiate_upload_sequence
	DOM2, DOM4	Upload_segment
	DOM2, DOM4	Terminate_upload_sequence
	DOM3	Request_domain_download
	DOM4	Request_domain_upload
	DOM5	Load_domain_content
	DOM6	Store_domain_content
	DOM7	Delete_domain
	DOM8	Get_domain_attribute
Gerenciamento de Invocação de Programa	PRG1	Create_program_invocation
	PRG1	Delete_program_invocation
	PRG2	Start
	PRG3	Stop
	PRG4	Resume
	PRG5	Reset
	PRG6	Kill
	PRG7	Get_program_invocation_attribute
Acesso a Variáveis	VAR9	Define_named_type
	VAR6	Define_named_variable
	VAR5	Define_named_variable_list
	VAR7	Define_scattered_access
	VAR9	Delete_named_type
	VAR5	Delete_named_variable_list
	VAR6	Delete_variable_access
	VAR8	Get_named_type_attributes
	VAR5	Get_named_variable_list_attributes
	VAR7	Get_scattered_access_attributes
	VAR1	Get_variable_access_atrributes
	VAR4	Information_report (*)
	VAR2	Read
	VAR3	Write

[continua na página seguinte]

[continuação]

Gerenciamento de Semáforo	SEM1,SEM2,SEM3	Take_control
	SEM1,SEM2,SEM3	Relinguish_Control
	SEM3	Define_semaphore
	SEM3	Delete_semaphore
	SEM1,SEM2,SEM3	Report_semaphore_status
	SEM1,SEM2,SEM3	Report_pool_semaphore_status
	SEM1,SEM2,SEM3	Report_semaphore_entry_status
	SEM2	Attach_to_semaphore_modifier
Comunicação com Operador	OCS1	Input
	OCS2	Output
Gerenciamento de Eventos	EVN3	Acknoledge_event_notification
	EVN2	Alter_event_condition_monitoring
	EVN3	Alter_event_enrollment
	EVN4	Attach_to_event_condition_modifier
	EVN3	Define_event_action
	EVN2	Define_event_condition
	EVN3	Define_event_enrollment
	EVN3	Delete_event_action
	EVN2	Delete_event_condition
	EVN3	Delete_event_enrollment
	EVN2, EVN3	Event_notification (*)
	EVN5	Get_alarm_summary
	EVN6	Get_alarm_enrollment_summary
	EVN3	Get_event_action_attributes
	EVN1	Get_event_condition_attributes
	EVN3	Get_event_enrollments
	EVN3	Report_event_action_status
	EVN1	Report_event_condition_status
	EVN3	Report_event_enrollment_status
	EVN4	Trigger_event
Gerenciamento de Journal	JOU2	Read_journal
	JOU1	Write_journal
	JOU2	Initialized_journal
	JOU2	Report_journal_status
Gerenciamento de Arquivos	FIL2	File_open
	FIL2	File_read
	FIL2	File_close
	FIL3	File_rename
	FIL3	File_delete
	FIL3	File_directory

(*) Somente Primitivas request e indication
(**) Somente Primitiva indication

Tabela 2.2 : Serviços MMS e seus CBB's Verticais

2.2.1. A Estrutura dos Serviços

O padrão MMS associa, a cada serviço, uma tabela contendo colunas para qualificação do seus parâmetros nas primitivas "Request", "Indication", "Response" e "Confirm". Cada tabela é definida com os seguintes elementos :

- "Conformance" (Conformidade) : Especifica o "Conformance Building Block" (CBB) a que se refere o serviço definido na tabela. Esta classificação é utilizada na iniciação do contexto, onde os usuários comunicantes negociam os conjuntos de serviços que serão utilizados;
- "Argument" (Argumento) : Define o conjunto de parâmetros e subparâmetros que estão presentes mas primitivas de "request" e "indication". Cada parâmetro pode ser :
  - Obrigatório (M - "Mandatory");
  - Opcional (U - "User Option");
  - Condicional (C - "Conditional") : depende de outro parâmetro;
  - Seletivo (S - "Selected") : escolhido entre um ou mais;
  - Definido por outros padrões (COMP - "Companion").
- "Result (+)" : Define o conjunto de parâmetros e subparâmetros que estão presentes nas primitivas de "response" e "confirm". Cada parâmetro pode ser do tipo M, U, C, S ou COMP, definidos no Argumento. Este resultado positivo ocorre quando o serviço é executado com sucesso;
- "Result (-)" :Define o tipo de erro ocorrido, caso o serviço termine em fracasso. Este campo define parâmetros obrigatórios, mas é do tipo Seletivo, pois ocorre no lugar do "Result (+)".

O código "(-)" colocado nas tabelas, a seguir aos códigos M, U, C ou S, indica que o parâmetro é semanticamente igual ao parâmetro na primitiva do serviço imediatamente à sua esquerda na tabela.

Na figura 2.3. mostra-se um exemplo de uma tabela para o serviço "Initiate", do Gerenciamento de Contexto. Nesta tabela todos os parâmetros da primitiva "Confirm" (coluna Cnf), com exceção de "Client Vertical CBB Called" e "Server Vertical CBB Called", são semanticamente iguais aos parâmetros da primitiva "Response" (coluna Rsp), pois possuem o código "(-)".

Conformance: CON1, CON2 Parameter Name	Req	Ind	Rsp	Cnf	CBB
Argument	M	M			
List of Version Numbers	M	M			
Proposed Max Message Size	U	U			
Proposed Max Serv Outstanding Calling	M	M			
Proposed Max Serv Outstanding Called	M	M			
Proposed Data Structure Nesting Level	U	U			
Proposed Horizontal CBB	M	M			
Client Vertical CBB Calling	M	M			
Server Vertical CBB Calling	M	M			
Result(+)			S	S(-)	
Negotiated Version Number			M	M(-)	
Negotiated Max Message Size			U	U(-)	
Negotiated Max Serv Outstanding Calling			M	M(-)	
Negotiated Max Serv Outstanding Called			M	M(-)	
Negotiated Data Structure Nesting Level			U	U(-)	
Negotiated Horizontal CBB			M	M(-)	
Client Vertical CBB Called			M	M	
Server Vertical CBB Called			M	M	
Result(-)			S	S(-)	
Error Type			M	M(-)	

Figura 2.3 : Estrutura do serviço "Initiate"

Os parâmetros da figura 2.3. "List Of Version Numbers" e "Negotiated Version Number" representam, respectivamente, uma lista de versões do Protocolo que podem ser selecionadas para o contexto em estabelecimento, proposta pelo usuário MMS chamador nas primitivas "request/indication", e o valor da lista que foi selecionado pelo usuário MMS chamado nas primitivas "response/confirm". Os parâmetros prefixados na figura 2.3. com "Proposed", na primitivas "request/indication", são analisados pelo usuário MMS chamado, e respondidos com valores menores ou iguais a estes, nos parâmetros prefixados com "Negotiated", nas primitivas "response/confirm". São negociados valores como o tamanho máximo de mensagens ("Max Mesage Size"), o número máximo de serviços pendentes nos usuários MMS chamador e chamado ("Max Serv Outstanding Calling e Called", respectivamente), o número máximo de níveis de aninhamento na estruturação dos dados ("Data Strucuture Nesting Level"), os CBB's Horizontais e os CBB's Verticais dos usuários MMS chamador e chamado, atuando como Cliente e Servidor ("Client/Server Vertical CBB Calling/Called").

Como se pode notar na figura 2.3., cada tabela possui ainda uma coluna, que para cada parâmetro acima, mostra o CBB ao qual o parâmetro está relacionado, isto é, se existir uma dependência do parâmetro com outro serviço, esta coluna mostra o CBB correspondente : o parâmetro estará disponível e pode ser usado, se e somente se o CBB associado foi negociado no contexto MMS estabelecido. Um exemplo de dependência de um parâmetro a um CBB, é o caso do parâmetro "Monitor" do serviço "Create_Program_Invocation" (do grupo de Gerenciamento de Chamadade Programas), qûe para ser utilizado (este parâmetro é do tipo opcional), requer que os CBB's EVN2 e PRG7 tenham sido negociados. Os CBB's EVN2 e PRG7 estão associados aos serviços de "Condição de Eventos" (do grupo de Gerenciamento de Eventos) e ao serviço "Get_Program_Invocation_Attributes" (do grupo de Gerenciamento de Chamada de Programas), respectivamente.

2.3. O Protocolo MMS

O Protocolo MMS é estruturado de forma a permitir que subconjuntos de protocolos possam ser definidos, tornando o MMS capaz de uma grande variedade de aplicações.

2.3.1. O Campo de aplicação do Protocolo MMS

O Protocolo MMS especifica :

- Procedimentos únicos para a transferência de dados e informações de controle de uma AE para a AE-par no contexto MMS;
- A seleção dos serviços à serem utilizados pela AE enquanto comunicando-se no contexto MMS. Esta seleção é feita através dos CBB's negociados;
- A estrutura de Unidades de Dados de Protocolo (PDU's) MMS, utilizadas para a transferência de dados e controle de informação.

Os procedimentos especificados no Protocolo MMS são definidos em termos de :

- Interação entre AE-pares através da troca de PDU's MMS;
- Interação entre um Provedor MMS e o Usuário MMS (via API) no mesmo sistema, através da troca de primitivas MMS;
- Interação entre um Provedor MMS e o ACSE através da troca de primitivas do serviço de Controle de Associação;
- Interação entre o Provedor MMS e o Provedor do serviço de Apresentação através da troca de primitivas do serviço de Apresentação.

A figura 2.4. mostra estas interações, definindo as interfaces dos elementos do Protocolo MMS na Camada de Aplicação.

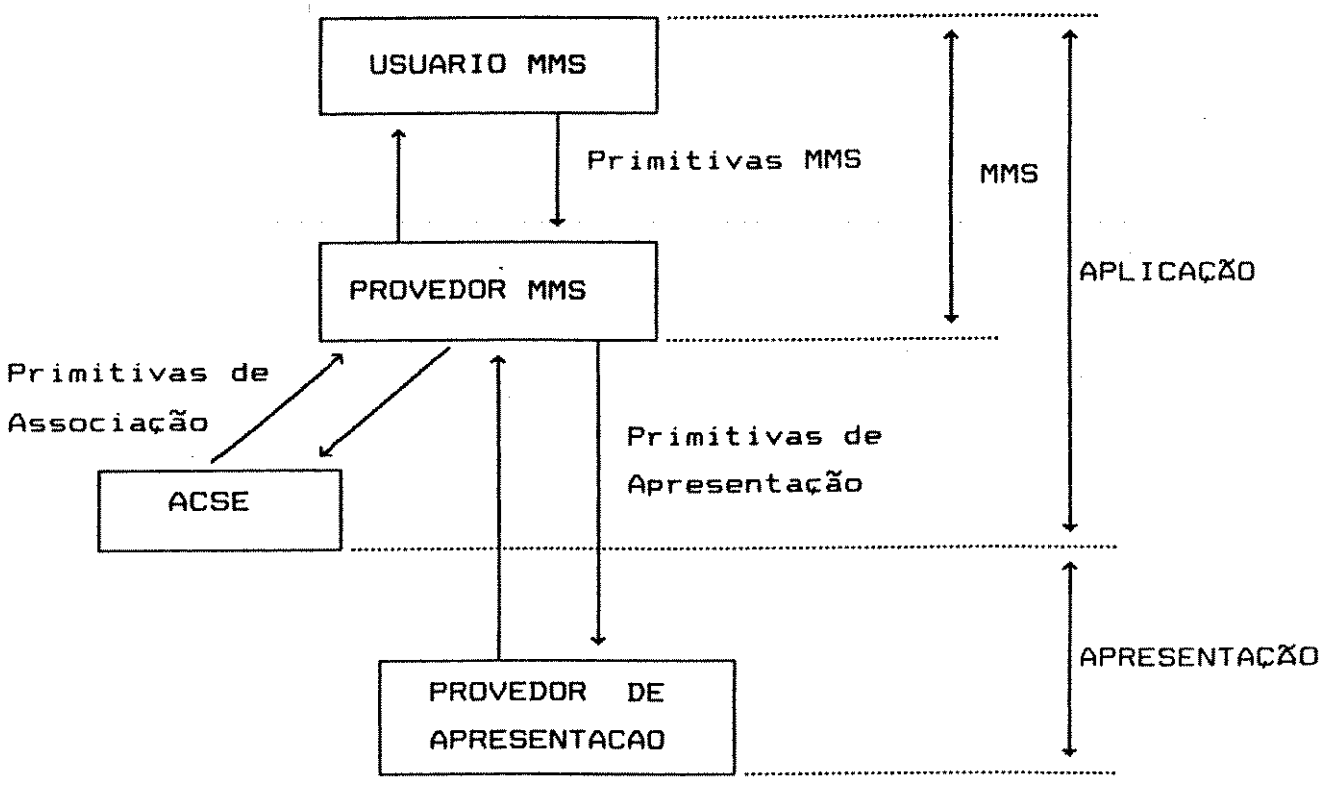

Figura 2.4 : Interações entre os elementos no Protocolo MMS

2.3.2. Os Elementos de Procedimentos do Protocolo

Os elementos de procedimentos de protocolo são descritos em relação ao envio e recebimento de PDU's MMS e suas relações com os eventos e primitivas de serviços na fronteira entre o usuário e o Provedor MMS.

O Protocolo MMS define 14 tipos de PDU's, conforme mostra a tabela 2.3.

PDU MMS	Primitiva que mapeia / Função
Confirmed Request_PDU	"request" de um serviço MMS confirmado
Confirmed_Response_PDU	"response" (positiva) de um serviço MMS confirmado
Confirmed_Error_PDU	"response" (negativa) de um serviço MMS confirmado
Unconfirmed_Request_PDU	"request" de um serviço MMS não confirmado
Reject_PDU	Indica a ocorrência de um erro de protocolo detectado na AE remota
Cancel_Request_PDU	"request" de cancelamento de um serviço MMS confirmado, pendente
Cancel_Response_PDU	"response" (positiva) de cancelamento de um serviço MMS confirmado, pendente
Cancel_Error_PDU	"response" (negativa) de cancelamento de um serviço MMS confirmado, pendente
Initiate_Request_PDU	"request" do serviço de inicialização de contexto MMS
Initiate_Response_PDU	"response" (positiva) do serviço de inicialização de contexto MMS
Initiate_Error_PDU	"response" (negativa) do serviço de inicialização de contexto MMS
Conclude_Request_PDU	"request" do serviço de conclusão de contexto MMS estabelecidos
Conclude_Response_PDU	"response" (positiva) do serviço de conclusão de contexto MMS
Conclude_Error_PDU	"response" (positiva) do serviço de conclusão de contexto MMS

Tabela 2.3 : PDU's MMS

Todas as PDU's MMS são repassadas no campo "user_data" das primitivas do ACSE ou do serviço de Apresentação, conforme mostrado na figura 2.4.

O mapeamento das PDU's é o seguinte :

- PDU's mapeadas nas primitivas da Camada de Apresentação (todas mapeadas nas primitivas "P_data_request" ou P_data_indication") :

  - "Confirmed_Request_PDU";
  - "Confirmed_Response_PDU";
  - "Confirmed_Error_PDU";
  - "Unconfirmed_Request_PDU";
  - "Reject_PDU";
  - "Cancel_Request_PDU";
  - "Cancel_Response_PDU";
  - "Cancel_Error_PDU";
  - "Conclude_Request_PDU";
  - "Conclude_Response_PDU";
  - "Conclude_Error_PDU";

- PDU's mapeadas nas primitivas do ACSE :

  - "Initiate_Request_PDU"
    (Primitiva "A_Associate_request" ou "indication");
  - "Initiate_Response_PDU"
    (Primitiva "A_Associate_response" ou "confirm" positivo);
  - "Initiate_Error_PDU"
    (Primitiva "A_Associate_response" ou "confirm" negativo");

A primitiva do serviço "Abort" é mapeada diretamente na primitiva A_U_abort do ACSE.

2.3.3. A codificação das PDU's MMS

As PDU's MMS representam a sintaxe abstrata do Protocolo na notação ASN.1. A figura 2.5 mostra um exemplo de PDU correspondente ao serviço "Initiate". A sua interpretação é facilitada comparando-se a figura 2.5. com a tabela estrutural deste serviço, na figura 2.3. Na figura 2.5. apresenta-se as PDU's "Initiate_request", "Initiate_response" e "Initiate_Error". As PDU's "Request" e "Response" são representadas como um registro estruturado (tipo "SEQUENCE" em ASN.1) dos parâmetros das primitivas. Cada parâmetro corresponde a um campo no "SEQUENCE" e possui um "TAG", de forma a identifica-lô. Os "TAG's" são numerados sequenciamente a partir de zero.

Um campo cujo "TAG" é seguido de "IMPLICIT", significa que o tipo do campo pode ser deduzido a partir do "TAG", não necessitando que a codificação daquele tipo seja novamente construida. No caso das PDU's "Initiate_Request" ou "Initiate_Response", todos os parâmetros são do tipo "IMPLICIT", ou seja, todos os parâmetros são derivados a partir de tipos já definidos no Protocolo.

Os parâmetros do tipo "U" na tabela da figura 2.5. são declarados em ASN.1 com "OPTIONAL", possuindo uma representação própria, para que a AE receptora da PDU possa discriminar a sua existência ou não. Os parâmetros dos tipos "Integer8, "Integer16" e "Integer32" são definidos no Protocolo MMS como inteiros positivos de 8, 16 e 32 bits, respectivamente, enquanto o tipo "INTEGER" é um tipo padrão definido na ASN.1.

O parâmetro "listOfVersionNumbers" (da PDU "Initiate_Request") é definido como uma "SEQUENCE OF INTEGER", que representa uma estrutura de elementos do mesmo tipo (vetor), no caso de valores inteiros.

Os tipos "HorizontalSupportOptions" e "VerticalSuportOptions" constituem no Protocolo MMS vetores de bits, onde cada bit está associado a um CBB Horizontal e Vertical, respectivamente.

```
Initiate_RequestPDU ::= SEQUENCE
{
 listOfVersionNumbers                [0] IMPLICIT  SEQUENCE OF INTEGER,
 proposedMaxMessageSize              [1] IMPLICIT  Integer32 OPTIONAL,
 proposedMaxServOutstandingCalling [2] IMPLICIT Integer16,
 proposedMaxServOutstandingCalled  [3] IMPLICIT Integer16,
 proposedDataStructureNestingLevel [4] IMPLICIT Integer8 OPTIONAL,
 proposedHorizontalCBB               [5] IMPLICIT
                                          HorizontalSupportOptions,
 clientVerticalCBBCalling            [6] IMPLICIT
                                          VerticalSupportOptions,
 serverVerticalCBBCalling            [7] IMPLICIT
                                          VerticalSupportOptions
}

Initiate_ResponsePDU ::= SEQUENCE
{
 negotiatedVersionNumber               [0] IMPLICIT INTEGER,
 negotiatedMaxMessageSize              [1] IMPLICIT Integer32 OPTIONAL,
 negotiatedMaxServOutstandingCalling [2] IMPLICIT Integer16,
 negotiatedMaxServOutstandingCalled  [3] IMPLICIT Integer16,
 negotiatedDataStructureNestingLevel [4] IMPLICIT Integer8 OPTIONAL,
 negotiatedHorizontalCBB               [5] IMPLICIT
                                           HorizontalSupportOptions,
 clientVerticalCBBCalled               [6] IMPLICIT
                                           VerticalSupportOptions,
 serverVerticalCBBCalled               [7] IMPLICIT
                                           VerticalSupportOptions
}

Initiate_ErrorPDU  ::= ServiceError
```

Figura 2.5 : Codificação da PDU "Initiate"

O tipo "Service_Error" é definido no Protocolo MMS como um tipo genérico, para os casos de resposta negativa em diversas primitivas de serviços.

2.4. As Especificações MMS e a Comunicação no SISDI_MAP

Todos os serviços definidos na Especificação de Serviços MMS podem ser executados no SISDI_MAP, desde que a Classe Funcional destes serviços tenha sido implementada pela API. Isto é possível uma vez que o Provedor MMS é capaz de tratar todos as primitivas de serviço mapeadas nas PDU's "Confirmed" e "Unconfirmed" de forma genérica, controlando apenas a máquina de protocolo destas. Entretanto os serviços básicos, que são os de gerenciamento de contexto no MMS, de controle das associações no ACSE e de controle de conexão e dados na Camada de Apresentação, são individualizados tanto no tratamento efetuado por cada máquina de protocolo, como na estrutura de mensagens respeitando-se o mapeamento das PDU's MMS e o mapeamento das PDU's ACSE nas primitivas de Apresentação.

Não existe uma relação direta entre PDU's MMS, primitivas ACSE, PDU's ACSE e primitivas de Apresentação, uma vez que :

- O Protocolo MMS utiliza os serviços ACSE apenas no estabelecimento de contexto (incluindo liberação e aborto);
- O Provedor MMS utiliza o serviço de transferência de dados de Apresentação (primitiva "P_data" para o mapeamento de diversas PDU's MMS;
- Nem todas as primitivas são mapeadas em PDU's, como as de "abort" do Provedor de Apresentação ("P_P_abort") e do Provedor ACSE ("P_A_abort").

A tabela 2.4. mostra a relação entre as primitivas e PDU's (mapeamento) do MMS, ACSE e Apresentação.

Primitiva MMS	Tipo	PDU MMS	Tipo	Primitiva ACSE	Tipo	PDU ACSE	Tipo	Primitiva de Apresentação	Tipo
initiate	1	Initiate	1	A_associate	1	A_Associate	1	P_connection	
conclude	1	Conclude	1	—		—		P_data	
abort	2	Abort	2	A_abort	2	A_Abort	2	P_u_abort	
cancel		Cancel	1	—		—		P_data	
reject	2	Reject	2	—		—		P_data	
confirmed service	1	Confirmed PDU	1	—		—		P_data	
Unconfirmed service	2	Unconfir_ med PDU		—		—		P_data	
—		—		A_release	1	A_release	1	P_release	
				A_p_abort	3	—		P_p_abort	

- As Primitivas do tipo 1 são "request, "indication" "response" ou "confirm", as do tipo 2 são "request" e "indication" e as do tipo 3 são somente "indication";
- As PDU's MMS do tipo 1 são "request", "response" ou "error", e as do tipo 2 são apenas "request";
- As PDU's ACSE do tipo 1 são "request" e "response" e as do tipo 2 são apenas "request".

Tabela 2.4 : Mapeamento das PDU's/Primitivas na comunicação do SISDI_MAP

3. O MODELO DE IMPLEMENTAÇÃO

O Protocolo MMS, necessita, para sua implementação, de um ambiente multitarefa, voltado para a comunicação em tempo real. Neste ambiente é importante um modelamento da estrutura de mensagens entre processos, que corresponda ao mapeamento das primitivas e PDU's definidas no Capítulo 2.

Na implementação do SISDI_MAP, usa-se um modelo com estas características, detalhado neste capítulo. O SISDI_MAP apesar de ser um sistema didático, procura representar com fidelidade os ambientes da manufatura propostos no MAP. Dentro desta filosofia, os AP's e o Provedor MMS podem se comportar no SISDI_MAP tanto como o cliente quanto o servidor MMS. Estes são caracterizados como os "usuários finais" MMS que solicitam serviços e que os executam, respectivamente.

3.1. O ambiente multitarefa

O ambiente multitarefa presente no modelo de implementação do SISDI_MAP é provido por um núcleo em tempo real [16], que apesar de não ser necessariamente dedicado, tem características que atendem requisitos do modelo para :

- Fornecer um ambiente para execução de diversos processos computacionais de forma simples, eficiente, e de fácil utilização;
- Prover serviços de controle e gerenciamento da comunicação entre processos, através de esquemas bem definidos como o de "mailboxes";
- Oferecer portabilidade suficiente para execução em sistemas diversos.

Este núcleo é escrito quase totalmente na linguagem "C", a mesma do Software Aplicativo, e é oferecido na forma de uma biblioteca de funções. Cada conjunto de funções correlatas está estruturado em um grupo denominado "gerência". O núcleo define as seguintes gerências :

- Gerência de Escalação de Tarefas;
- Gerência de Filas;
- Gerência de Memória;
- Gerência de Interrupções;
- Gerência de Sincronização e Comunicação entre processos;
- Gerência de Semáforos;
- Gerência de Entrada/Saída.

Algumas destas gerências estão diretamente associadas ao modelo de implementação do SISDI_MAP, por oferecerem serviços relacionados à comunicação e execução de processos. São as seguintes gerências :

- Gerência de Escalação de Tarefas : Responsável pela execução de tarefas (processos), através das seguintes primitivas :

  - Cria_tarefa : Cria uma tarefa para o núcleo com prioridade selecionada e possibilidade de passagem de parâmetros para esta tarefa em criação;
  - Destrói_tarefa : Termina uma tarefa para o núcleo;
  - Reescalona : Entrega o controle do processador a uma tarefa mais prioritária que a corrente;
  - Suspende_tarefa : Coloca uma tarefa em estado de "suspensão" durante um período de tempo determinado;
  - Resume_tarefa : Retira uma tarefa do estado suspenso, recolocando-a como ativa para execução.

- Gerência de Memória : Responsável pela alocação de áreas de memória, com posterior controle de sua liberação, através das seguintes primitivas :

  - Pede_mem : Solicita uma certa quantidade de memória;
  - Libera_mem : Libera uma quantidade de memória previamente alocada por "pede_mem";
  - Mem_disp : Informa sobre a quantidade de memória livre ainda existente.

- Gerência de Sincronização e Comunicação : Responsável pela comunicação entre tarefas através da manipulação de portas (áreas de manutenção de apontadores para elementos genéricos), oferecendo as seguintes primitivas :

  - Cria_porta : Cria uma porta para o núcleo;
  - Libera_porta : Deleta uma porta no núcleo;
  - Envia_mens : Coloca mensagem em uma porta especificada;
  - Espera_mens : Espera por mensagens (por um tempo determinado) em uma porta;
  - Obtem_mens : Lê, caso exista, mensagem em uma porta.

3.2. A utilização dos recursos oferecidos pelo núcleo

Os processos aplicativos que implementam o SISDI_MAP são tratados como tarefas que são criadas com prioridades diferentes e parâmetros próprios, e nunca são finalizadas em tempo de execução.

Estas tarefas interagem através das portas de comunicação, definidas pelo núcleo como bidirecionais e independentes de processos, que podem colocar e retirar mensagens nestas portas, num esquema de fila gerenciado pelo núcleo. Para fins de

otimização as mensagens são colocadas na forma de apontadores para a área de dados efetiva.

A estrutura de comunicação entre processos do SISDI_MAP provida pelo núcleo pode ser considerada, conforme classificação das classes de linguagens para Programação concor r ente [24], como troca de mensagens assíncrona. Isto se deve a colocação dos "objetos de sistema" (os blocos funcionais d a figura 1.1., apresentados no Capítulo 1), em processos que se comunicam através de primitivas do tipo envia e recebe mensagens, cuja execução independe da situação dos processos emissor e receptor. A figura 3.1. mostra os processos que implementam o SISDI_MAP, oriundos do mapeamento direto dos blocos funcionais citados acima, e também o escopo do modelo de comunicação proposto nesta tese, que se preocupa apenas com a estrutura das mensagens nas interfaces de protocolo (MMS, ACSE e Apresentação), e na interface de operação para monitoração e inserção de erros nas PDU's dos protocolos.

Durante a comunicação provida pelo núcleo, o mecanismo de envio de mensagens é não bloqueante, enquanto o de recepção é opcionalmente bloqueante, ou seja, é possível indicar a suspensão da tarefa que recebe a mensagem caso a porta especificada esteja vazia. Esta suspensão é temporizada de forma a evitar esperas infinitas. No caso não bloqueante é informada a quantidade de mensagens na porta.

3.2.1. A espera de mensagens

No SISDI_MAP a suspensão da execução de um processo (tarefa) só ocorre na chamada da primitivas "espera_mens", com opção "CONSOME". A espera pode ser apenas de verificação da existência de mensagem (Opção "VERIFICA"), com posterior obtenção da mensagem para tratamento através da primitiva "obtem_mens".

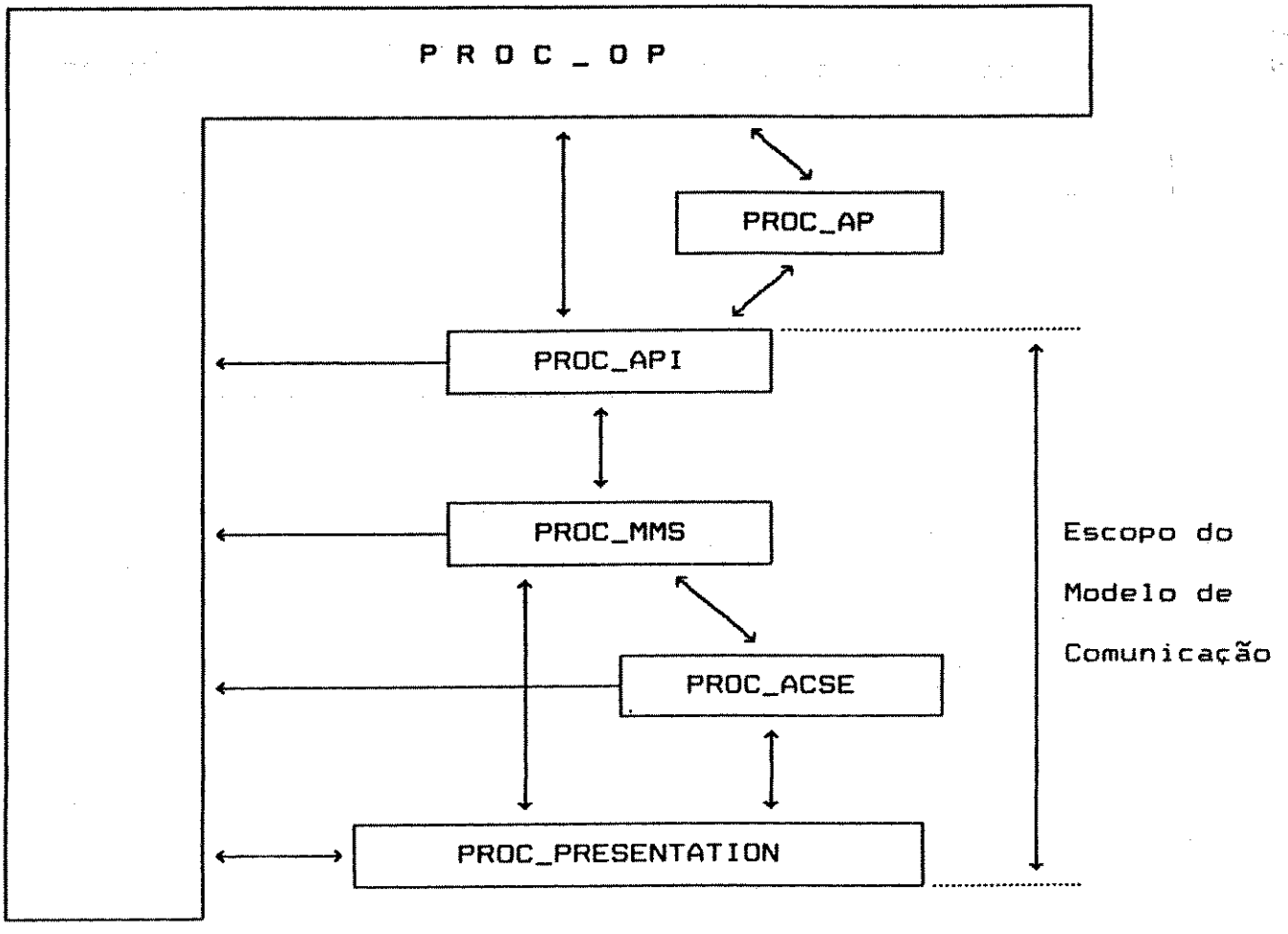

Figura 3.1 : Processos do SISDI_MAP

Quando um processo espera uma mensagem numa porta com opção "CONSOME", se não houver mensagem, o processo é colocado no estado "suspenso" e outro processo toma a CPU. Existe uma fila de processos (controlada pelo núcleo) que esperam mensagens em uma mesma porta, ordenada segundo o momento de chamada da primitiva.

Quando chegar uma mensagem na porta em questão, o primeiro processo da fila a obtêm. Deve-se observar que esta fila independe da prioridade do processo, mas somente da ordem de espera das mensagens.

No SISDI_MAP apenas um processo estará esperando mensagem numa porta por vez (as portas são "alocadas" aos processos). Além disso a espera não é prorrogada, ou seja os processos esperam uma mensagem com tempo mínimo (1 segundo) e opção "CONSOME". Se houver mensagem na porta o núcleo devolve "OPERAÇÃO_OK" como parâmetro, caso contrário o parâmetro indica que não há mensagem de saída.

Outra opção seria implementar as primitivas "espera_mens", com opção "VERIFICA", e se houver mensagem, utilizar a primitiva "obtem_mens". Neste caso seria necessário a utilização de outro mecanismo para cessão do processo à outra tarefa, como por exemplo a chamada da primitiva "reescalona". Por não trazer qualquer vantagem adicional, esta opção foi descartada.

O procedimento adotado simplifica a utilização das primitivas e permite uma busca cíclica em todas as portas de um processo, sem delongas na espera por mensagens, uma vez que o SISDI_MAP é um tipo de sistema multiprogramável, denominado na literatura "event driven", ou seja toda a execução é orientada pela ocorrência de eventos provocados externamente.

3.2.2. O envio de mensagens

Quando um processo coloca uma mensagem numa porta, através da primitiva "envia_mens", e há um processo de maior prioridade do que ele à espera de mensagens nesta porta, a primeira mensagem da porta é entregue ao processo mais prioritário. Este processo é colocado no "estado ativo", sendo suspenso o processo corrente (que enviou a mensagem), e tomando o processador a tarefa mais prioritária dentre as ativas. Entretanto se o processo que enviou a mensagem for mais prioritário, o processo receptor só será ativado se o que enviou se suspender ou for bloqueado por espera de mensagem. No SISDI_MAP a suspensão do processo mais prioritário (Proc_OP : Interface de Operação de Usuário) só ocorre quando este se bloqueia à espera de mensagens em suas portas. Os demais

processos têm a mesma prioridade; neste caso é eleito para execução o processo ativo com maior tempo de espera, numa filosofia "round-robin".

Estes recursos de comunicação providos pelo núcleo, aliados a escala de prioridade dos processos do SISDI_MAP fornecem um ambiente adequado à implementação deste tipo de protocolos orientados a eventos, pois facilitam a hierarquia vertical de comunicação no sistema, processada no sentido AP →API → MMS →(ACSE) → Presentation e vice-versa.

3.3. A Estrutura de Comunicação

O modelo da estrutura de mensagens do SISDI_MAP é baseado nas seguintes premissas :

- Minimização da alocação de memória e cópia de informação na troca de mensagens;
- Otimização do mecanismo de identificação das mensagens pelo processos que as recebem;
- Fidelidade com relação a sequência, tipo e conteúdo dos parâmetros das primitivas e PDU's MMS circulantes na rede;
- Facilidade de monitoração da troca de mensagens na rede, visando rastrear o comportamento do sistema.

Para atender estas premissas definiu-se no modelo do SISDI_MAP que as portas :

- São utilizadas de forma unidirecional e associadas a um processo, ou seja, cada processo recebe na sua criação, de forma parametrizada, suas portas de entrada, através das quais só recebe mensagens, e conhece a identificação das portas de saída, pertencentes aos processos que tem interface, para onde só envia as mensagens. A figura 3.2. mostra estas portas dentro do modelo do SISDI_MAP;

- Nunca são liberadas por algum processo em tempo de execução;
- Implementam para os processos Proc_MMS, Proc_ACSE e Proc_Presentation o conceito de Ponto de Acesso de Serviço (SAP). Para cada conexão de Aplicação que o sistema estabelece, utilizando um SAP, existe um Identificador Final da Conexão (CEP) interno ao SAP, denominado no SISDI_MAP, conforme [12], de identificador da conexão ("connection_id").

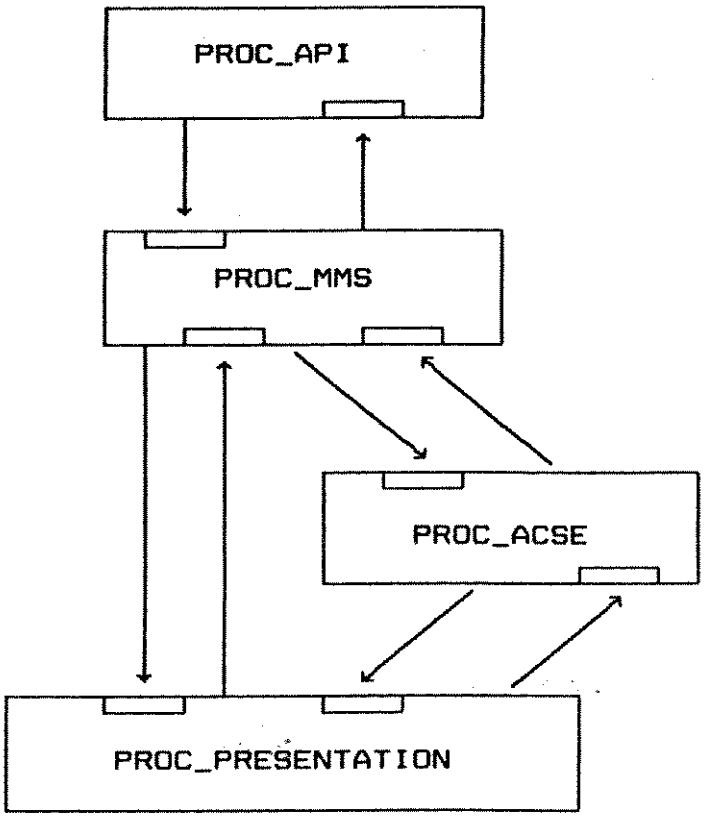

Figura 3.2 : As portas de comunicação no Modelo.

Quanto à estrutura das mensagens o modelo define dois blocos. As mensagens de comunicação entre os processos que implementam os protocolos são estruturadas no "Bloco de mensagens de Protocolo", e as mensagens de controle e interface com o usuário para simulação de erros no sistema são estruturadas no "Bloco de mensagens de Controle e Interface". O Anexo I mostra o arquivo de mensagens do SISDI_MAP contendo a codificação na linguagem "C" destes blocos.

3.3.1. Bloco de mensagens de Protocolo

O Bloco de mensagens de Protocolo define a estrutura de todas as primitivas, PDU's e seus parâmetros, utilizados no SISDI_MAP, mapeadas conforme a Tabela 2.1. Este bloco implementa o conceito estrutural de uma primitiva, composto por :

- Informações de controle de Protocolo, denominadas PCI ("Protocol Control Information");
- Unidades de Dados de Serviço, denominada SDU ("Service Data Unit"), que transporta a PDU da camada superior.

A figura 3.3. mostra um exemplo prático desta estrutura, de concatenação de PCI's e SDU's, utilizando-se do mapeamento de primitivas/PDU's no SISDI_MAP para o serviço de Gerenciamento de contexto MMS. Nesta comunicação as primitivas MMS são mapeadas em PDU's MMS, que são enviadas ao ACSE nas primitivas deste. Estas por sua vez são mapeadas em PDU's ACSE e enviadas à Camada de Apresentação, através da primitivas desta.

Neste esquema o MMS acresce aos dados de usuário (SDU) o seu PCI, formando a PDU MMS. O ACSE trata a PDU MMS como sua SDU e a esta acresce o seu PCI para formação da PDU ACSE, que constitui o SDU do "Presentation", quando do envia da primitiva desta camada.

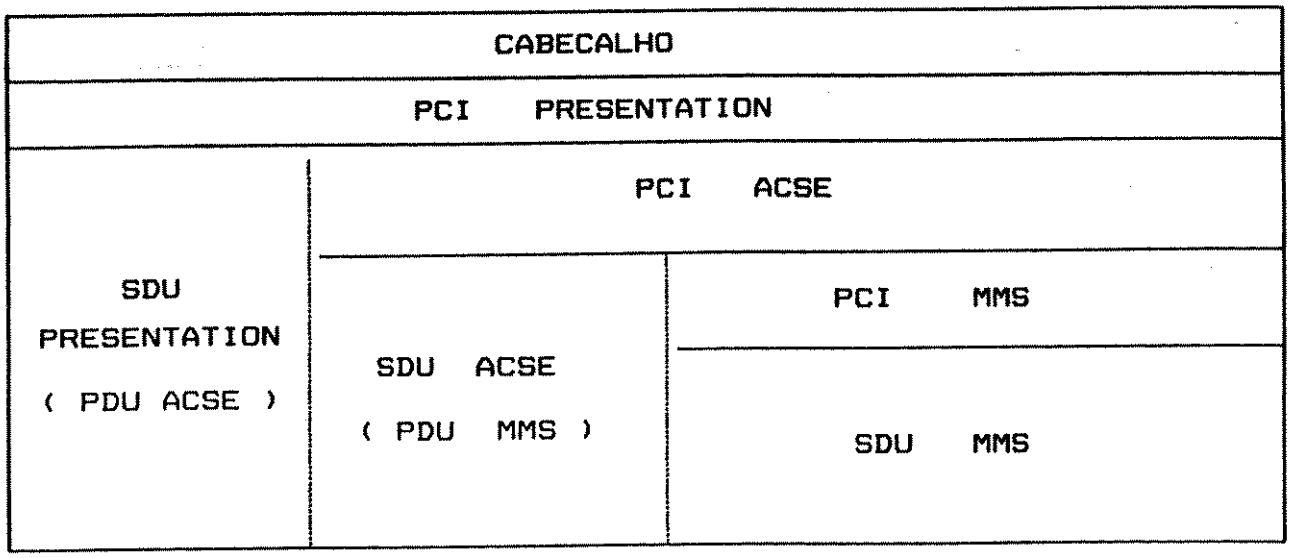

Figura 3.3 : Estrutura dos PCI's e SDU's nos serviços de Gerenciamento de Contexto

A implementação do Bloco de mensagens de Protocolo apresenta de forma genérica a codificação das primitivas e PDU's no SISDI_MAP, através de um cabeçalho comum e uma parte específica para cada tipo de mensagem (utilizando-se o comando "union" da linguagem "C"). O cabeçalho comum define :

(1) - o identificador de conexão (CEP)
(2) - o tipo de primitiva ("request, indication, response ou confirm")
(3) - o tipo de serviço (MMS confirmado, MMS não confirmado, gerenciamento de contexto, ACSE ou Apresentação)
(4) - o tamanho da área de dados de mensagem (tamanho da parte específica).

A parte específica é um registro (tipo "struct" da linguagem "C") definido para cada um dos serviços apresentados em (3),

contendo os parâmetros das primitivas.

A figura 3.4. mostra a estrutura do bloco de mensagens de protocolo, que faz parte do Anexo I.

As mensagens do Bloco de Protocolo fazem o mapeamento direto das primitivas em PDU's MMS, seguindo a Tabela 2.1 . Este mapeamento é feito alterando-se somente alguns parâmetros, quando necessário, representando uma simplificação, que é possível na implementação do *SISDI_MAP* uma vez que a PDU é definida no modelo como um elemento abstrato, por não estar codificada segundo a sintaxe ASN.1. Entretanto o modelo prevê a simulação dos erros de codificação, para que o sistema mantenha a fidelidade ao ambiente real de rede.

```
typedef struct
      {
       Connection_id    connection_id;
       Primitive_type   primitive;
       Service_type     service;
       sint16           data_area_lenght;   /*    maior primitiva
                                                ocupa 296 bytes */
       union    /* Area de Dados de cada tipo de servico */
               {
                Mms_conf_struct       mms_conf_serv;
                Mms_unconf_struct     mms_unconf_serv;
                mms_cont_struct       mms_cont_serv;
                Acse_struct           acse_serv;
                Presentation_struct   pres_serv;
               } data;
     }Block_struct;
```

Figura 3.4 : Bloco de mensagem de Protocolo do *SISDI_MAP*

As mensagens do Bloco de Protocolo são visíveis para os processos através de "máscaras" adequadas a cada um, de forma que nenhuma informação precisa ser copiada de um bloco para outro, ou

seja, os dados tratados por uma primitiva são "vistos" pelo processo que os trata com uma máscara manipulada por aquele processo, enquanto outro processo vê os mesmos dados, na mesma área de memória com outra máscara, mais adequada a seu universo de tratamento dos dados. Os campos não utilizados por uma determinada máscara são definidos como elementos "fantasmas", denominados "dummy" (elemento que ocupa 1 byte ou vetor de n bytes. Ver Anexo I). Este mecanismo :

- Permite a separação dos PCI's e SDU's tratados por cada Protocolo;
- Garante a um processo o conforto no acesso às estruturas, na maioria das vezes complexa;
- Oferece maior clareza da programação.

Um exemplo de como as máscaras são estruturadas é mostrado na figura 3.5. para as primitivas de estabelecimento de conexão, "P_connection, A_associate, initiate", definidas para os processos Proc_Presentation, Proc_ACSE e Proc_MMS, respectivamente.

A definição das máscaras é um dos pontos críticos do modelo, que apesar de eficiente é complexo e pouco versátil às mudanças nos parâmetros. Esta susceptibilidade a alterações torna-se mais clara se imaginarmos que um determinado parâmetro do "P_connection", no exemplo da figura 3.5., necessite de mais um "byte" para sua representação. Isto provoca a redefinição das máscaras utilizadas pelo ACSE (A_associate) e MMS (initiate), onde o tamanho dos campos "dummy" devem ser redefinidos.

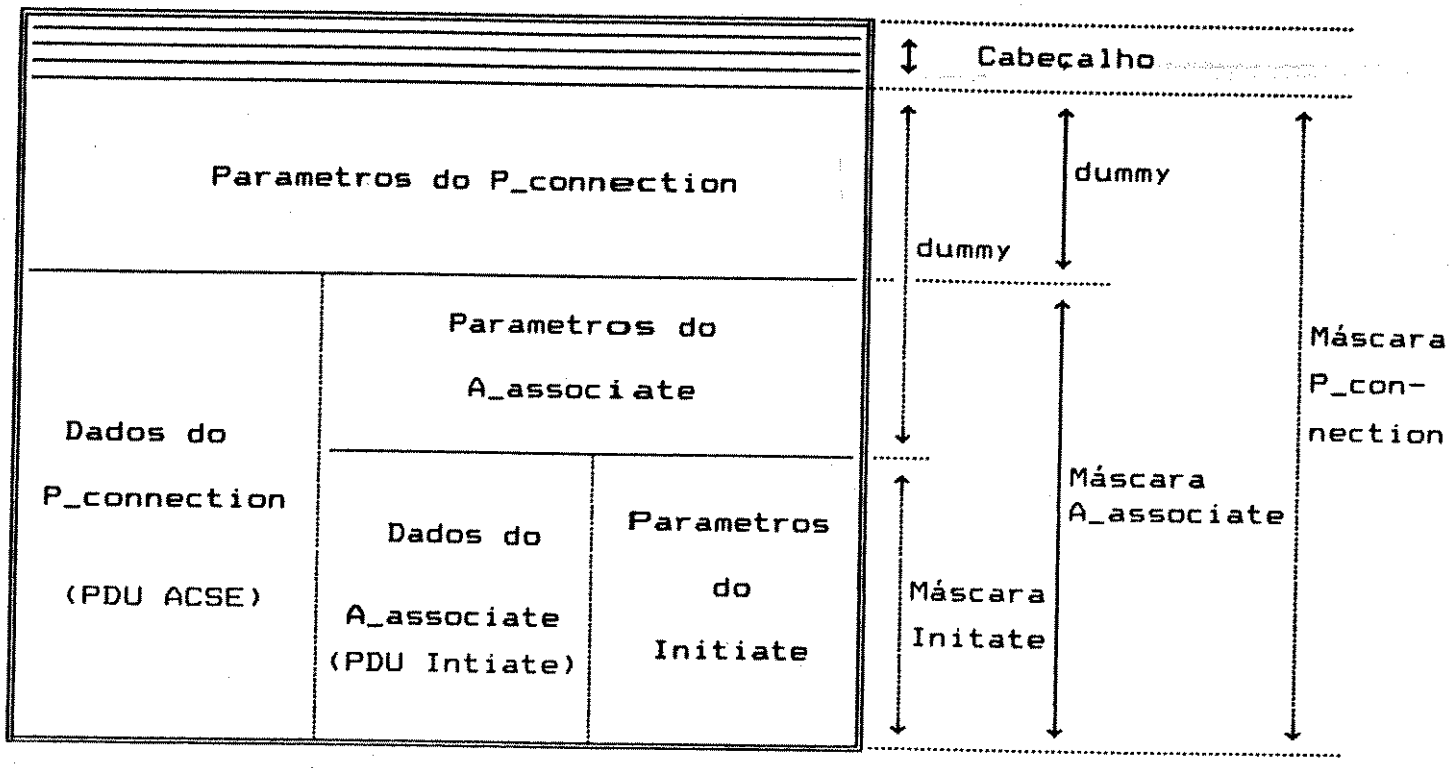

Figura 3.5 : Exemplo de máscaras das mensagens do Bloco de Protocolo

3.3.2. Bloco de mensagens de Controle e Interface

As mensagens do Bloco de Controle e Interface seguem um padrão diferente do bloco de protocolos e estão associadas as funções de monitoração e controle de execução e operação dos protocolos executados no SISDI_MAP, em cooperação com o gerenciamento do sistema.

O gerenciamento do sistema é provido no SISDI_MAP por um processo independente, de maior prioridade, responsável pela interface de operação de usuário do sistema [17] (Proc_OP). O Proc_OP oferece várias facilidades de monitoração através de

"menus" e permite a execução interativa dos mesmos serviços requeridos pelo AP ou API, ou seja, se um AP não for definido, o usuário pode emular o AP com os mesmos comandos que este se utiliza para requisição de serviços ao sistema.

Para a execução da facilidade de simulação de erros de protocolo o modelo define uma estrutura utilizada pela interface de Operação e pelo simulador de Apresentação, que constitui as mensagens do Bloco de Controle e Interface. São elas :

- Solicitação de transferência de PDU : Ao chegar uma PDU na Camada de Apresentação, o processo solicita à interface, através desta mensagem, o envio da PDU ao destino (Proc_ACSE ou Proc_MMS), na forma de uma primitiva "indication ou confirm". Para tal o Proc_Presentation envia uma cópia da mensagem contendo a PDU e aguarda a resposta (primitiva "espera_mens" do núcleo), ficando bloqueado nesta espera;
- Resposta de transferência de PDU : Após receber a mensagem de solicitação, a interface interage com o usuário do sistema antes de enviar a resposta. Se o usuário desejar a inserção de um erro (que pode ser erro de identificação da invocação do serviço ("invoke_id"), erro de serviço confirmado ou não confirmado, erro de PDU inválida ou erro de primitiva de Apresentação com confirmação negativa), a interface envia esta resposta com o tipo de erro e o seu parâmetro (novo "invoke_id" ou novo serviço), se houver. Caso contrário a interface envia a resposta indicando a não colocação de erros;
- Solicitação de aborto de conexão de Apresentação : Esta mensagem provoca a simulação da primitiva de aborto do Provedor de Apresentação ("P_P_abort"), enviada pelo Proc_Presentation ao Proc_ACSE, liberando todas as conexões estabelecidas.

O item 3.5. apresenta detalhadamente os tratamentos efetuados

pelo Proc_Presentation no envio e recepção de mensagens na Inteface de Operação.

A figura 3.6. mostra a estrutura principal do Bloco de mensagens de Controle e Interface do SISDI_MAP, codificada na linguagem "C". O Anexo I mostra a codificação completa deste bloco.

```
typedef struct
     {
      Select_msg_user_interface   msg;
      union
           {
            Pdu_tranf_req     transf_req;     /*  se solicitacao
                                                 de transferencia */
            Connection_id     connection_aborted
                                              /* se abort_by_user */
            Pdu_transf_res    transf_res;   /* se resposta  de
                                                 transferencia */
           }user_msg
     }User_com_block;
```

Figura 3.6 : Bloco de mensagens de Controle e Interface no SISDI_MAP

A figura 3.7. mostra um exemplo de mensagens no modelo do Bloco de Controle e Interface, para o caso de inserção de erro solicitada na resposta do usuário à transferência da PDU. O usuário solicita a alteração do "invoke_id" com valor 1, para a PDU do serviço confirmado "status" para o valor 3, que provocará a detecção do erro pela máquina de estado do Proc_MMS [19]. Na figura 3.7. a sequência de mensagens é numerada conforme a ordem de ocorrência destas. Deve-se notar que no Proc_Presentation o apontador para o bloco de protocolo (mensagem 1) é enviado ao Proc_OP (campo "pdu_block" na mensagem 2), para que este informe ao usuário o conteúdo da mensagem recebida.

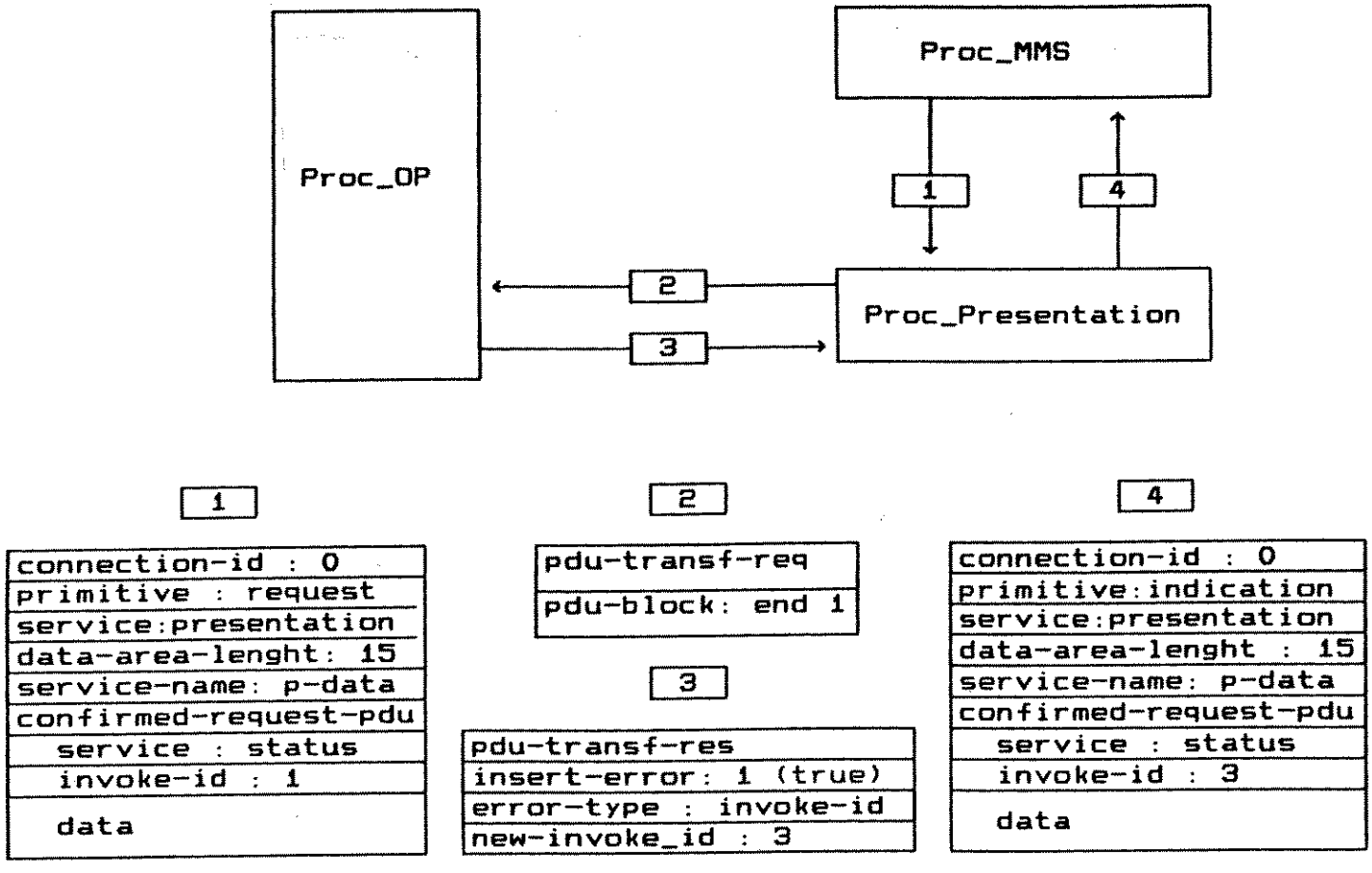

Figura 3.7 : Exemplo de inserção de erros pelo usuário numa PDU MMS

Outro aspecto importante do modelo, implícito nas mensagens 1 e 4 da figura 3.7. é a "comutação" das conexões existentes no sistema, identificadas pelo "connection_id" das mensagens 1 e 4, contendo os valores 0 e 5, respectivamente. Esta "comutação" é realizada pelo processo Proc_Presentation, que ao estabelecer uma conexão, simula a rede de comunicação selecionando, conforme o Endereço das Entidades de Apresentação (AE's), chamada e chamadora, um CEP de saída, associando-o ao CEP de entrada selecionado pelo usuário (Ver item 3.5.3.).

3.4. Comparação com um modelo clássico

Em Halsall [25], apresenta-se uma arquitetura clássica de implementação de um sistema de comunicação OSI, que será mostrada a seguir, e comparada com o modelo do SISDI_MAP.

3.4.1. O modelo de Halsall

O Sistema de comunicação OSI de Halsall é implementado através de um conjunto de tarefas (processos), uma por camada de protocolo, com tarefas adicionais para execução de determinadas funções, como por exemplo gerenciamento local. As tarefas se comunicam através de um conjunto de filas tipo FIFO ("First_In_First_Out") ou mailboxes, como mostra a figura 3.8. A comunicação entre tarefas é gerenciada por um núcleo em tempo real, que executa também funções de escalonamento de tarefas e controle de interrupções.

A comunicação entre camadas de protocolo é feita através de primitivas de serviço, cada qual criada num formato definido (de domínio local), ocupando um "buffer" de memória denominado ECB ("Event Control Block"). A estrutura do ECB é dependente da estrutura de dados da linguagem de implementação de cada entidade de protocolo (processos), uma vez que diferentemente da sintaxe rígida associada com a PDU, os parâmetros das primitivas do serviço estão na sintaxe local. Além disso a ECB é única para todas as primitivas numa mesma interface entre camadas, possuindo um cabeçalho para identificação do tipo de primitiva.

A passagem de primitivas entre camadas (processos) é feita através de uma primitiva de comunicação do núcleo, tendo como parâmetro o apontador para o ECB. Isto provoca a inserção pelo núcleo do apontador no final da fila de comunicação. Quando o processo receptor é escalonado para execução, ele examina suas

filas de entrada para verificar se há um ECB esperando processamento. Se houver, o evento recebido é processado.

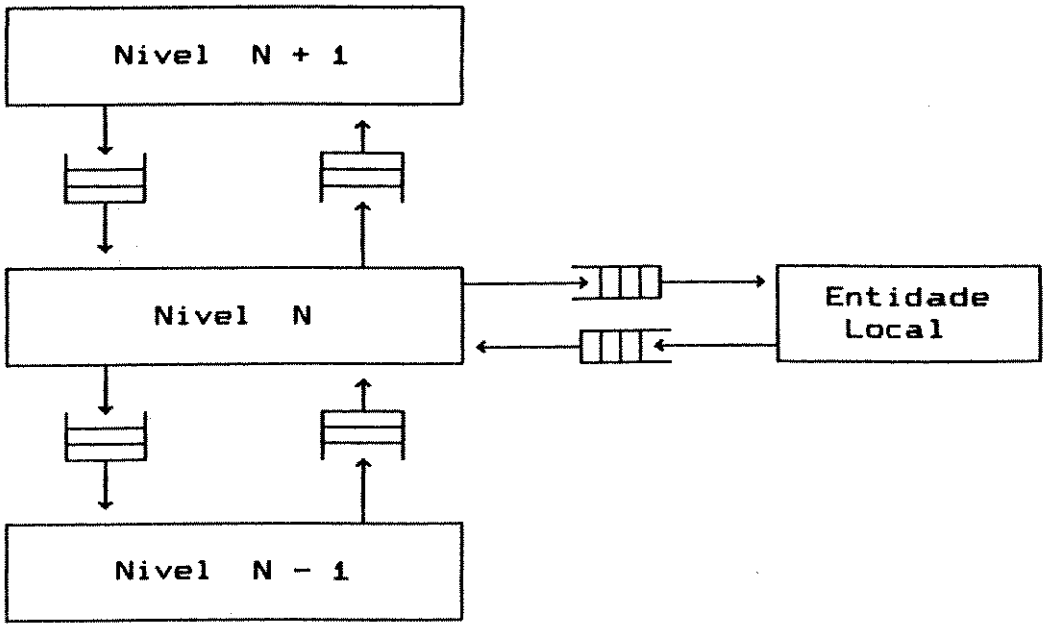

Figura 3.8 : A comunicação entre tarefas no Modelo de Halsall

O modelo discute também a necessidade de se definir uma fila específica para cada ponto de acesso de serviço (canal) que estiver ativo, uma vez que existem situações de protocolo que devem ser trabalhadas de forma e em tempos diferentes para cada canal, cujo gerenciamento se torna complicado se houver uma fila única para todos os canais. Este mecanismo aumenta consideravelmente o número de filas a medida que muitos canais se tornam simultaneamente ativos, porém possibilita um controle eficiente do fluxo de mensagens em cada canal. Um aspecto que o modelo não discute é o gerenciamento destas filas, feito pelo núcleo, com relação a ativação e desativação dos canais.

A formação de uma PCI é feita por uma estrutura de

protocolo contendo os parâmetros associados com a primitiva recebida (ECB), juntamente com as informações de estado atual do protocolo, associadas à conexão. A concatenação da PCI com os dados de usuário da primitiva recebida, forma a PDU que é passada à camada inferior. O modelo propõe que os dados associados com a primitiva sejam armazenados num buffer separado, denominado UDB ("User Data Buffer"). O UDB contém os PCI's e PDU's acumulados por cada uma das camadas superiores, constituindo, na camada mais inferior, o conjunto de bytes a ser transmitido pelo nível físico.

Cada ECB possui um campo apontador para a UDB, que é o ponteiro para o endereço que contém os dados de usuário associados com a primitiva, e um campo que define o tamanho da UDB.

O modelo declara as ECB's e suas UDB's associadas, como estruturas de dados globais, uma vez que estas devem ser acessíveis à todas as camadas. Um "pool" de ECB's (para cada camada) e UDB's é criado para a camada na inicialização do sistema e os apontadores para estes buffers são ligados na forma de uma lista. O núcleo deve controlar a utilização deste "buffers" (obtenção e devolução de e para a lista) e implicitamente conhecer a estrutura das ECB's.

3.4.2. Comparação entre os modelos

Os aspectos de comunicação do modelo de implementação de protocolos OSI de Halsall [25], citados no item 3.4.1, são comparados com o modelo do SISDI_MAP. A tabela 3.1. mostra os itens comparados para os dois modelos e uma observação em cada item, que explica as diferenças e semelhanças entre os modelos.

Item em Comparação	Modelo de Halsall	Modelo do SISDI_MAP	Observações
Forma de Comunicação entre tarefas	Filas "FIFO" ou mailboxes	Portas associadas aos processos como mailboxes	Mecanismos semelhantes
Gerenciamento da comunicação	Núcleo em tempo real	Núcleo em tempo real	Mecanismos Idênticos
Estrutura de Comunicação	Buffer ECB, único para cada interface entre camadas	Bloco estruturado, único para todas as interfaces de protocolo	Diferenças apenas na forma
Passagem de parâmetros entre tarefas	Primitiva do núcleo, por apontador para o ECB	Primitiva do núcleo, por apontador para o bloco	Mecanismos Idênticos
Número de filas entre Camadas	Um conjunto de filas para cada SAP ativo	Um único conjunto de filas; cada fila com um único SAP	SISDI_MAP é um caso particular do modelo de Halsall
Estrutura dos dados de usuário	Cada ECB aponta para um buffer de UDB	Os dados fazem parte da estrutura de cada máscara de mensagens do bloco	SISDI_MAP simplifica a estrutura dos dados, pois utiliza sintaxe local
Controle de buffers	Gerenciado pelo núcleo, que conhece os ECB's	Núcleo gerencia memória, mas não conhece os ECB's	O Núcleo do SISDI_MAP não é dedicado

Tabela 3.1 : Comparação entre modelos de Halsall e do SISDI_MAP

3.5. Simulação da Camada de Apresentação

A Simulação da Camada de Apresentação tem por objetivo criar uma interface transparente aos Protocolos MMS e ACSE, aos quais presta serviços através das primitivas de Apresentação, emulando para a Camada de Aplicação, a comunicação fim-a-fim suportada pela rede.

Assim uma primitiva "request" ou "response" é transformada numa primitiva "indication" ou "confirm", respectivamente, e entregue à AE remota (que nesta implementação é o mesmo processo que envio a primitiva : Proc_MMS ou Proc_ACSE), tornando esta comunicação no SISDI_MAP mais próxima da que ocorre num ambiente real. Esta simulação pode, através do Bloco de Controle e Interface (item 3.3.2), introduzir erros no conteúdo das PDU's que são transportadas pelas primitivas de Apresentação, ou indicar um erro de codificação ASN.1, com o intuito de testar o comportamento das máquinas de Protocolo ACSE e MMS nas situações de erro.

3.5.1. A Implementação da Simulação da Camada de Apresentação

A Simulação da Camada de Apresentação é implementada no SISDI_MAP através de uma única instância do processo "Presentation", denominado "Proc_Presentation", que serve simultaneamente à todas as Entidades de Apresentação configuradas no sistema.

A implementação desta simulação permite :

- Configuração de intervalos de CEP's para cada Endereço de Apresentação;
- Associação dos CEP's para comutação de primitivas, conforme o Endereço de Apresentação do chamado e do chamador;
- Transação com à Operação do SISDI_MAP antes do encaminhamento das primitivas em comutação.
- Inserção de erros nas PDU's;

3.5.2. O Processo "Presentation"

Este processo está detalhado a seguir com relação a suas interfaces, temporizadores, estados, estrutura de Dados, inicialização, condições de bloqueio e algorítmo de implementação.

3.5.2.1. Interfaces

O Proc_Presentation se interfaceia com os processos Proc_ACSE, Proc_MMS e Proc_OP, conforme figura 3.9.

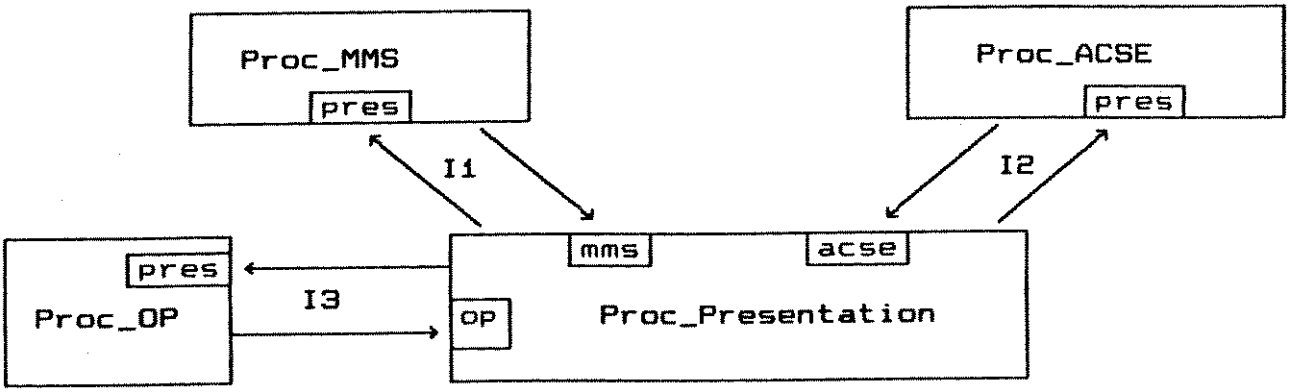

Figura 3.9 : Interfaces do Proc_Presentation

As mensagens trocadas pelo Proc_Presentation nas interfaces I1 e I2 da figura 3.9. são definidas no Bloco de Protocolo, e as trocadas na interface I3 são definidas no Bloco de Controle e Inteface. Estes blocos estão descritos no item 3.3.

O Proc_Presentation utiliza portas de recepção e portas de transmissão para cada interface da figura 3.9. As portas são as seguintes :

- ingate_mms : recepção de mensagens do Proc_MMS;
- ingate_acse : recepção de mensagens do Proc_ACSE;
- ingate_op : recepção de mensagens do Proc_OP;
- outgate_mms : transmissão de mensagens para o Proc_MMS;
- outgate_acse : transmissão de mensagens para o Proc_ACSE;
- outgate_op : transmissão de mensagens para o Proc_OP.

3.5.2.2. Temporizadores

O Proc_Presentation possui dois temporizadores para espera de mensagens nas portas citadas em 3.5.2.1., utilizados na primitiva do núcleo "espera_mens" :

- T_wait_loop : valor = 1 segundo. Este temporizador é definido para espera das mensagens no loop do programa principal (Ver item 3.5.2.7.1.) e seu valor é o mínimo permitido pelo núcleo, porque o Proc_Presentation utiliza a primitiva apenas para verificação da existência de mensagens nas portas, verificação que é realizada de forma cíclica em todas as portas de entrada. Se houver mensagem na porta verificada, esta é tratada de acordo com o tipo de porta, conforme pode ser visto no Anexo II.
- T_wait_op : valor = 1000 segundos. Este temporizador é definido para espera de mensagem na porta ingate_op, quando o Proc_Presentation realiza uma transação com o Proc_OP para solicitação de transferência de PDU (com ou sem simulação de erros). O seu valor é elevado para que o usuário do SISDI_MAP possa responder ao tipo de erro desejado, quando esta chave estiver selecionada na interface de operação [17]. Durante este intervalo de

espera o Proc_Presentation é bloqueado conforme discutido em 3.5.2.6. A espera de mensagem na porta ingate_op no corpo principal do processo é apenas para o caso de simulação de "abort" do Provedor de Apresentação.

3.5.2.3. Estados

O Proc_Presentation não possui estados definidos uma vez que não implementa a Máquina de Protocolo da Camada de Apresentação [15], atuando apenas como repassador de primitivas de Apresentação sob controle do Proc_OP. Assim o tratamento de eventos ocorre após a iniciação do processo (item 3.5.2.5.), idependente de estados.

3.5.2.4. Estrutura de Dados

A Estrutura de Dados do Proc_Presentation é composta por variáveis locais (utilizadas dentro do escopo processo) e Tabelas.

3.5.2.4.1. Variáveis locais

O Proc_Presentation possui as seguintes variáveis locais :

- block : apontador para as mensagens do Bloco de Protocolo, do tipo estruturado "Block_struct"; conforme item 3.3;
- block_ctr : apontador para as mensagens do Bloco de Controle e Interface do tipo estruturado "User_com_block", conforme item 3.3;

- con_id : identificador da conexão, recebido no campo "connection_id" das primitivas request e response, do tipo "Connection_id", definido no Anexo I;
- mbi_mms, mbo_mms : portas de entrada e saída de mensagens do e para o Proc_MMS, respectivamente;
- mbi_acse, mbo_acse : portas de entrada e saída de mensagens do e para o Proc_ACSE, respectivamente;
- mbi_op, mbo_op : portas de entrada e saída de mensagens do e para o Proc_OP, respectivamente.

As variáveis locais de portas acima citadas armazenam os parâmetros do item 3.5.2.1, recebidos na criação do Proc_Presentation.

3.5.2.4.2. Parâmetros de Configuração

O Proc_Presentation utiliza os seguintes parâmetros de configuração, definidos no Anexo I :

- Max_presentation : Número máximo de entidade de Apresentação existentes no SISDI_MAP. Valor = 3;
- Max_association : Número máximo de associações permitidas para cada entidade de Apresentação (pontos de acesso de Apresentação). Valor = 10;
- Max_connection_id : Número máximo de identificadores de conexão definidos pelo SISDI_MAP. Valor = 30.

Estes parâmetros representam uma restrição para limitação da memória utilizada na execução do sistema.

3.5.2.4.3. Tabelas

O Proc_Presentation possui as seguintes tabelas :

(a) - Tabela de relação dos intervalos de identificadores da conexão ("Connection_id") aos endereços de Apresentação (campo "presentation_address" da mensagem "P_connection") : Vetor estruturado cujo índice está no intervalo [0..Max_presentation] (Anexo I), e o conteúdo é composto do endereço de Apresentação, do tipo "P_address", e do intervalo de "Connection_id" relativo ao endereço (primeiro e último elementos do intervalo). Deve-se observar que o intervalo é dimensionado conforme a configuração do número máximo de associações permitido para as entidades de Aplicação, a fim de evitar colisões.

(b) - Tabela de associação dos identificadores de conexão utilizados : Vetor utilizado na "comutação" de primitivas, cujo índice é o "connection_id" da primitiva "request", no intervalo [0..Max_connection_id] (Anexo I), e o conteúdo é o "connection_id" selecionado pelo Proc_Presentation para primitiva "indication". O conteúdo "not_used" = 255 significa que o "connection_id" não está sendo utilizado;

O item 3.5.3. mostra um exemplo de utilização das tabelas (a) e (b).

3.5.2.5. Inicialização

A inicialização do Proc_Presentation ocorre após a criação do processo, onde os elementos da tabela (b) do item 3.5.2.4.3. são inicializados com valor "not_used" = 255 e os elementos da tabela (a) são passados como parâmetros de criação do processo. Os valores das portas de comunicação, passados como parâmetros, também são atribuidos às variáveis locais.

Em seguida o Proc_Presentation está apto à execução de suas funções, tratando a ocorrência de eventos provocados pelos processos que interfaceia.

3.5.2.6. Bloqueio do Processo

O Proc_Presentation pode ficar bloqueado em duas situações :

- Por espera de mensagens nas suas portas de entrada, quando não há mensagem disponível na porta esperada. Após o tempo T_wait_loop, o processo é desbloqueado e faz a busca na porta seguinte, ficando novamente bloqueado, e assim sucessivamente, até que uma mensagem seja colocada em alguma porta.
- Por espera de resposta a uma transação com o Proc_OP, nos procedimentos de tratamento de mensagens do MMS e ACSE. O bloqueio pode durar até T_wait_op.

3.5.2.7. Algoritmos

Este item apresenta os principais procedimentos que constituem o Proc_Presentation, definindo um pseudo-código para este processo numa linguagem "Pascaloíde". O Anexo II mostra a listagem contendo a programação completa do Proc_Presentation na linguagem "C".

3.5.2.7.1. Programa Principal

```
BEGIN
  Inicialização da estrutura de dados;
  LOOP  /* Loop eterno */
    Espera_mensagem;
    CASE (origem da mensagem) OF
      ACSE : Treat_acse !
      MMS : Treat_mms !
      Interface de Operação de Usuário : Treat_user
    END
  END /* Loop */
END;
```

3.5.2.7.2. Tratamento das mensagens do ACSE

```
PROCEDURE Treat_acse;
  BEGIN
    Obtem identificador da conexão;
    IF (serviço = P_connection) AND (primitiva = request)
      THEN BEGIN
             Obtem um connection_id de saida;
             IF (Não há connection_id livre na tabela)
              THEN Envia P_p_abort ao ACSE
              ELSE Atualiza tabela de associação de conexões;
           END;
      ELSIF (serviço = P_release) AND (primitiva =
             confirm_pos)
        THEN Libera connection_id associados;
    IF (Não foi detectado erro na tabela de associção)
      THEN BEGIN
             Armazena mensagen recebida;
             Envia copia da mensagem à interace de usuário
             numa solicitação de transferência de PDU;
             Espera_mensagem;
             Trata resposta de transferência de PDU;
                      (Ver 3.5.2.7.5.)
           END;
  END;
```

3.5.2.7.3. Tratamento das mensagens do MMS

```
PROCEDURE Treat_mms;
  BEGIN
    Armazena mensagem recebida;
    Envia copia da mensagem à interface de usuário numa
    solicitação de transferência de PDU;
    Espera_mensagem;
    Trata resposta de transferência de PDU; (Ver 3.5.2.7.5.)

  END;
```

3.5.2.7.4. Tratamento da Interface de Operação de Usuário

```
PROCEDURE Treat_user
  BEGIN
    IF (mensagem = Abort pelo usuário)
    THEN BEGIN
           Envia P_P_abort_indication ao ACSE;
           Libera connection_id associados
         END;
  END;
```

3.5.2.7.5. Tratamento de Resposta de Transferência de PDU

```
PROCEDURE Treat_transf_res;
  BEGIN
    IF  (mensagem = resposta de transferência de PDU)
      THEN BEGIN
             CASE (tipo de primitiva) OF
               request : tipo de primitiva = indication !
               response_pos : tipo de primitiva = confirm_pos !
               response_neg : tipo de primitiva := confirm_neg
             END;
             IF (Operador solicita inserção de erros na resposta
                 na transação) THEN
                BEGIN
                  CASE (tipo de serviço) OF
                    gerenciamento de contexto, mms confirmado,
                      mms não confirmado  :
                      CASE (tipo de erro) OF
                        serviço : muda nome do serviço  !
                        codificação, tipo de PDU : preenche
                          campo da P_data_indication como
                          PDU inválida !
                        invoke_id  :  IF (serviço confirmado)
                                        THEN Muda invoke_id;
                      END  !
                    acse : Preenche campo da primitiva
                           indication ou confirm como PDU
                           inválida  !
                  END /* CASE */
                END /* IF */
             IF  (tipo de serviço = acse)
               THEN Envia mensagem ao ACSE destino
               ELSE Envia mensagem ao MMS destino
           END
  END;
```

3.5.3. Estabelecimento de Conexão de Apresentação

O estabelecimento de uma conexão de Apresentação é uma simulação de uma situação real, de forma transparente aos demais processos, e se faz através do preenchimento das Tabelas (a) e (b) do item 3.5.2.4.3.

A tabela de relação de intervalos de "connection_id" aos endereços de Apresentação deve ser conhecida também pelo Proc_API e pelo Proc_OP, que escolherá o "connection_id" na solicitação de uma conexão, encaminhada ao Proc_MMS, Proc_ACSE e finalmente ao Proc_Presentation. Por convenção a primeira metade dos "connection_id" de cada intervalo é alocada pelo Proc_OP (nas primitivas "request"), e a segunda metade é alocada pelo Proc_Presentation (nas primitivas "indication"). A tabela 3.2. mostra um exemplo desta tabela preenchida com 3 endereços de Apresentação, sendo que cada endereço possui um intervalo de 10 unidades para "connection_id".

ÍNDICE	ENDEREÇOS DE APRESENTAÇÃO	INTERVALOS DE connection_id	
		Primeiro	último
1	P_adr_1	0	9
2	P_adr_2	10	19
3	P_adr_3	20	29

Tabela 3.2 : Associação de Endereços de Apresentação e Intervalos de " connection_id"

Ao receber, num determinado instante, uma primitiva "P_connection_request" com endereço de Apresentação de chamador (P_adr_2) e "connection_id" 12 o Proc_Presentation obtêm o endereço de Apresentação do chamado (que se refere à Entidade de Apresentação remota) no campo "called_presentation_address". Se por exemplo este endereço for de P_adr_1, o Proc_Presentation procura na tabela de associação dos "connection_id" (tabela (b)) um "connection_id" de saída, no respectivo intervalo, que não esteja alocado, iniciando a procura pelo valor 9 que é o elemento de maior valor deste intervalo. Se esta tabela estiver preenchida conforme a Tabela 3.3., o valor de "connection_id" de associação escolhido será o 7. Isto ocorre porque os valores 9 e 8 da tabela 3.3. estão alocados aos "connection_id" 20 e 0, respectivamente. O Proc_Presentation deve então preencher o índice 12 da Tabela 3.3. com o valor 7, e o índice 7 com o valor 12. Quando a conexão é liberada ou abortada, a associação é desfeita, e os índices respectivos recebem o valor not_used = 255.

Deve-se observar que a associação dos valores 0 e 8 da Tabela 3.3. significa que duas Entidades de Aplicação que estão sobre a mesma Entidade de Apresentação estão se comunicando.

Se não houver "connection_id" livre para alocação, o Proc_Presentation envia um "P_P_abort" para o Proc_ACSE (Ver item 3.5.2.7.2.). Isto ocorre quando o Operador do SISDI_MAP excede o número máximo de associações permitido para a Entidade de Apresentação chamada.

ENDEREÇOS DE APRESENTAÇÃO	Connection_id	Connection_id associado
P_adr_1	0	8
	1	255
	:	:
	:	:
	:	:
	7	255 (12)
	8	0
	9	20
P_adr_2	10	29
	11	255
	12	255 (7)
	13	255
	:	:
	:	:
	:	:
	19	255
P_adr_3	20	9
	21	255
	:	:
	:	:
	:	:
	28	255
	29	10

* Obs.: Os valores entre parênteses são os atribuidos no instante seguinte ao estabelecimento da conexão

Tabela 3.3 : Tabela de Comutação do Proc_Presentation

4. EXEMPLO DE EXECUÇÃO DOS SERVIÇOS MMS NO SISDI_MAP

Este Capítulo apresenta um exemplo da execução dos serviços MMS, sob o ponto de vista das mensagens de comunicação entre os processos do SISDI_MAP, utilizando-se dos blocos de mensagens descritos no Capítulo 3.

Para melhor ilustrar a utilização dos serviços MMS, o exemplo situa um ambiente de manufatura, onde é mostrada uma seqüência de mensagens na comunicação entre dois AP's, como proposto em [6], numa Célula Flexível da Manufatura. Os AP's representam um dispositivo programável (robô) e o Controlador da Célula. Deve-se observar que a seqüência de mensagens descrita no exemplo é apenas uma ilustração, cuja execução pode ser realizada na operação do SISDI_MAP, não correspondendo necessariamente à que ocorre no ambiente real.

Os serviços são executados obedecendo os seguintes passos :

- Estabelecimento de Contexto entre o Controlador e o Robô, por iniciativa do primeiro;
- Definição pelo Controlador de uma "variável", para monitoração de uma determinada medida executada pelo Robô, armazenada nesta variável;
- Informação, por iniciativa do Robô, sobre o valor da variável definada, (serviço não confirmado);
- Deleção da variável definida, por solicitação do Controlador;
- Cancelamento, pelo Controlador, do serviço solicitado de Deleção da variável;
- Leitura do valor armazenado na variável, por solicitação do Controlador;
- Rejeição da PDU do serviço de Leitura, no Robô, provocada pela simulação de erro na PDU, por solicitação do usuário do SISDI_MAP, na interface de Operação;
- Solicitação de Fechamento do Contexto estabelecido, por inicitativa do Controlador.

A seguir encontra-se detalhada a troca de mensagens e seus respectivos parâmetros, para cada um dos passos acima, considerando a seguinte situação de operação do sistema :

- O Controlador é suportado pela Entidade de Apresentação com endereço 100, que no caso é a chamadora; com "connection_id" no intervalo [0..9];

- O Robô é suportado pela Entidade de Apresentação com endereço 200, que no caso é a chamada; com "connection_id" no intervalo [10..19].

Os parâmetros não preênchidos nas máscaras das primitivas, mostradas nos itens seguintes, significam que não pertecem à máscara do processo, ou são opcionais. Neste último caso o campo anterior ao parâmetro (tipo Booleano, com a denominação "valid" no final), foi preenchido com o valor 0 (que representa "false" no SISDI_MAP).

4.1. Estabelecimento do Contexto entre os AP's

O Proc AP do Controlador interage com o API (Proc_API), que inicia o envio de primitivas que seguem a seqüência Proc_API → Proc_MMS → Proc_ACSE → Proc_Presentation → Proc_OP → Proc_Presentation → Proc_ACSE → Proc_MMS → Proc_API, para a solicitação do estabelecimento de contexto. O envio da resposta pelo Robô, segue a mesma seqüência, porém para as primitivas "response/confirm".

As primitivas trocadas na seqüência acima são detalhadas a seguir para a solicitação do serviço. As primitivas trocadas na resposta possuem estrutura semelhante, como se pode ver no Anexo I, portanto, não serão mostradas.

4.1.1. Primitiva "Initiate_request"

O Proc_API preênche os parâmetros desta mensagem, num bloco de memória alocado (via primitiva "pede_mem" do núcleo), e a coloca na porta do Proc_MMS. Os parâmetros da primitiva "Initiate_request" são "vistos" pelo Proc_MMS com a seguinte máscara, conforme mostra o Anexo I :

- connection_id : 0
- primitive : request
- service : mms_cont
- data_area_lenght : 296
- data : mms_cont_serv :
    - service_name : initiate
    - dummy [5] :
    - prim_cont_manag : initiate_serv :
        - dummy [256] :
        - initiate_req_pdu :
            - list_of_version_numbers : 1 0 0 0 0 0 0 0
            - prop_max_msg_size_valid : 1 (true)
            - max_msg_size : 256
            - prop_max_serv_outst_calling : 5
            - prop_max_serv_outst_called : 7
            - prop_data_str_nest_level_valid : 0 (false)
            - prop_data_str_nest_level :
            - prop_max_nesting_level : 3
            - prop_horizontal_cbb : str1, socs, rcv1
            - client_vert_cbb_calling : con1, con2, con3
              mms1, mms2, var3, var4, var6, vnam, vadr1
            - server_vert_cbb_calling : con1, con2, con3,
              mms1, mms2, var3, var4, var6, vnam, vadr1

4.1.2. Primitiva "A_associate_request"

O Proc_MMS trata a mensagem do item 4.1.1., e utilizando-se do mesmo bloco de mémoria, a coloca a na porta do Proc_ACSE. Os parâmetros da primitiva "A_associate_request" são "vistos" pelo Proc_ACSE com a seguinte máscara :

- connection_id : 0
- primitive : request
- service : acse
- data_area_lenght : 296
- data : acse_serv :
    - service_name : a_associate
    - prim_acse : a_associate_serv :
        - associate_req :
            - dummy :
            - calling_pres_address : 100
            - called_pres_address : 200
            - pres_contex_definition_list_valid : 0
            - pres_context_definition_list [4] :
            - pres_context_def_list_result_valid : 0
            - pres_context_def_list_result [4] :
            - default_pres__context_name_valid : 0
            - default_pres_context_name :
            - qualit_of_service : 1
            - pres_requeriments__valid : 0 (false)
            - pres_requeriments :
            - mode : 1
            - session_requeriments : 132
            - initial_synchr_point_serial_number : 12
            - initial_assigment_of_tokens : 1
            - associate_pdu :
                - protocol_version_valid : 1 (true)
                - protocol_version : 1

- application_context_name_valid : 0 (false)
- application_context_name [4] :
- calling_AP_title_valid : 0 (false)
- calling_AP_title [5] :
- calling_AE_qualifier_valid : 0 (false)
- calling_AE_qualifier :
- calling_AP_inv_ident_valid : 0 (false)
- calling_AP_inv_ident :
- calling_AE_inv_ident_valid : 0 (false)
- calling_AE_inv_ident :
- called_AP_title_valid : 0 (false)
- called_AP_title [5] :
- called_AE_qualifier_valid : 0 (false)
- called_AE_qualifier :
- called_AP_inv_ident_valid : 0 (false)
- called_AP_inv_ident :
- called_AE_inv_ident_valid : 0 (false)
- called_AE_inv_ident :
- user_information_valid : 1 (true)
- user_information :

4.1.3. Primitiva "P_connection_request"

O Proc_ACSE trata a mensagem do item 4.1.2., e utilizando-se do mesmo bloco de mémoria, a coloca na porta do Proc_Presentation. Os parâmetros da primitiva "P_connection_request" são "vistos" pelo Proc_Presentation com a seguinte máscara :

- connection_id : 0
- primitive : request
- service : presentation
- data_area_lenght : 296
- data : pres_serv :
    - service_name : p_connection

- prim_pres : p_connection_serv :
  - p_connection_req :
    - dummy1 :
    - calling_presentation_address : 100
    - called_presentation_address : 200

      :

    - user_data_valid : 1 (true)
    - user_data :

Como o Proc_Presentation implementa apenas uma simulação da Camada de Apresentação, este não se utiliza dos demais parâmetros da primitiva "P_connection".

4.1.4. Solicitação de Transfêrencia de PDU

O Proc_Presentation armazena o bloco recebido e envia a mensagem de Solicitação de transfência de PDU ao Proc_OP (como descrito no item 3.5.2.7.2.), contendo os seguintes parâmetros :

- msg :pdu_transf_req
- transf_req : pdu_block (Endereço da mensagem do Bloco de Protocolo)

4.1.5. Resposta de Transferência de PDU

O Proc_OP responde à Solicitação de Transferência de PDU, sem inserção de erros (por desejo do usuário do SISDI_MAP), enviando a mensagem de resposta ao Proc_Presentation, contendo os seguintes parâmetros :

- msg : pdu_transf_res
- insert_error : 0 (false)

4.1.6. Primitiva "P_connection_indication"

O Proc_Presentation comuta a primitiva (como exemplificado no item 3.5.3.), escolhendo um "connection_id" de saída adequado (no caso com valor 19), enviando ao Proc_ACSE a mensagem do Bloco de Protocolo armazenada, alterando apenas o tipo de primitiva de "request" para "indication" nos parâmetros do item 4.1.3.

4.1.7. Primitiva "A_associate_indication"

O Proc_ACSE trata a mensagem recebida do Proc_Presentation enviando-a ao Proc_MMS. A máscara utilizada é semelhante a do item 4.1.2., com a alteração do tipo de primitiva para "indication".

4.1.8. Primitiva "Initiate_indication"

O Proc_MMS trata a mensagem recebida do Proc_ACSE enviando-a ao Proc_API. A máscara utilizada é semelhante a do item .4.1.4, com a alteração do tipo de primitiva para "indication".

Neste instante o ciclo de trânsito das primitivas "request/indication" é completado, cabendo ao Proc_API, (que interagem com o Proc_AP) o envio, ao Proc_MMS, da resposta ao estabelecimento do Contexto, através das primitivas "response/confirm", num esquema semelhante ao descrito nos itens de 4.1.1. até este. Neste exemplo a primitiva de solicitação de estabelecimento de contexto é respondida positivamente.

4.2. Definição de Variável no Robô

Estando o contexto estabelecido o Proc_AP do Controlador interage com o Proc_API, que envia ao Proc_MMS uma solicitação do serviço confirmado "Define_Named_Variable", desencadeando a comunicação de mensagens na seqüência Proc_API → Proc_MMS → Proc_Presentation → Proc_OP → Proc_Presentation → Proc_MMS → Proc_API.

4.2.1. Primitiva "Define_Named_Variable_request"

O Proc_API envia a primitiva "Define_Named_Variable_request" ao Proc_MMS, que trata a mensagem com a seguinte máscara :

- connection_id : 0
- primitive : request
- service : mms_conf
- data_area_lenght :
- data : mms_conf_serv:
    - dummy [7] :
    - conf_request :
        - invoke_id : 0
        - modifier_valid : 0 (false)
        - modifier :
        - service_name : dnvariable
        - conf_req_data [Max_data] :

Deve-se observar que neste exemplo o identificador de invocação do serviço ("invoke_id") foi selecionado pelo Proc_API com o valor zero.

4.2.2. Primitiva "P_data_request" contendo PDU "Define_Named_Variable"

O Proc_MMS trata a primitiva recebida do Proc_API, e envia o mesmo bloco de memória recebido ao Proc_Presentation, porém preenchido com a máscara que este último se utiliza, contendo os seguintes parâmetros :

- connection_id : 0
- primitive : request
- service : presentation
- data_area_lenght :
- data : pres_serv :
  - p_data_serv : p_data_req :
    - dummy [3] :
    - user_data :
      - pdu : confirmed_request
      - mms_pdu :

O Proc_Presentation trata a mensagem recebida, executando uma transação com o Proc_OP para transferência da PDU, como descrito nos itens 4.1.4. e 4.1.5.. Neste caso o Proc_OP também response a "Solicitação de Transferência de PDU" sem inserção de erros.

4.2.3. Primitiva "P_data_indication" contendo PDU "Define_Named_Variable"

O Proc_Presentation comuta a primitiva recebida do Proc_MMS, enviando-a novamente a este. O Proc_MMS visualiza a mensagem com máscara contendo os seguintes parâmetros :

- connection_id : 19
- primitive : indication
- service : presentation
- data_area_lenght :
- data : pres_serv :
    - prim_pres : p_data_serv :
        - p_data_ind :
        - dummy [2] :
        - valid_pdu : 1 (true)
        - select_ind : user_data :
            - pdu : confirmed_request
            - mms_pdu :

4.2.4. Primitiva "Define_Named_Variable_indication"

O Proc_MMS trata a mensagem recebida do Proc_Presentation com a seguinte máscara :

- connection_id : 19
- primitive : indication
- service : mms_conf
- data_area_lenght :
- data : mms_conf_serv :
    - dummy [7] :
    - conf_request :
        - invoke_id : 0
        - modifier_valid : 0 (false)
        - modifier :
        - service_name : dnvariable
        - conf_req_data [Max_data] :

O Proc_MMS envia a mensagem ao Proc_API, encerrrando o ciclo de "request/indication" do serviço confirmado. Este serviço é respondido pelo Proc_API ao Proc_MMS, iniciando o ciclo de resposta, com uma estrutura semelhante a descrita nos itens de 4.2.1. até este. Neste exemplo a primitiva de solicitação de serviço é respondida positivamente.

4.3. Informação do valor da Variável

O Proc_AP do Robô monitora o valor da Variável definida no serviço descrito em 4.2., informando o seu valor periodicamente ao Proc_AP do Controlador. Esta informação se faz através do serviço "Information_Report", definido como serviço "não confirmado" no Protocolo MMS, ou seja, não necessita de resposta, possuindo somente primitivas do tipo "request" e "indication".

4.3.1. Primitiva "Information_Report_request"

O Proc_API envia a primitiva "Information_Report" ao Proc_MMS, que a trata com a máscara de serviço não confirmado, contendo os seguintes parâmetros :

- connection_id : 0
- primitive : request
- service : mms_unconf
- data_area_lenght :
- data : mms_unconf_serv :
  - dummy [7] :
  - unconf_request :
    - service_name : ireport
    - conf_req_data [Max_data] :

Deve-se observar que a primitiva de serviço não confirmado não possui o "invoke"id", uma vez que a AE requeridora do serviço (no caso a do Robô) não espera resposta à sua solicitação. Assim não é criada no Protocolo MMS uma máquina de estado para este serviço [1].

4.3.2. Primitiva P_data_request" contendo PDU "Information_Report"

O Proc_MMS trata a primitiva do Proc_API e envia o mesmo bloco de memória recebido ao Proc_Presentation, porém preenchido com a máscara que este último se utiliza, contendo os seguintes parâmetros :

- connection_id : 0
- primitive : request
- service : presentation
- data_area_lenght :
- data : pres_serv :
  - prim_pres : p_data_serv :
    p_data_req :
    - dummy [3] :
    - user_data :
      - pdu : unconfirmed_request
      - mms_pdu :

O Proc_Presentation trata a mensagem recebida, executando uma transação com o Proc_OP para transferência da PDU, como descrito nos itens 4.1.4. e 4.1.5.. Neste caso o Proc_OP também response a "Solicitação de Transferência de PDU" sem inserção de erros.

4.3.3. Primitiva "P_data_indication" contendo PDU "Information_Report"

O Proc_Presentation comuta a primitiva recebida do Proc_MMS, enviando-a novamente a este, que visualiza a mensagem com máscara contendo os seguintes parâmetros :

- connection_id : 19
- primitive : indication
- service : presentation
- data_area_lenght :
- data : pres_serv :
    - prim_pres : p_data_serv :
        p_data_ind :
            - dummy [2] :
            - valid_pdu : 1 (true)
            - select_ind : user_data :
                - pdu : unconfirmed_request
                - mms_pdu :

4.3.4. Primitiva "Information_Report_indication"

O Proc_MMS trata a mensagem do Proc_Presentation com a mesma máscara do item 4.3.1., com a diferença no valor dos parâmetros para "connection_id : 19" e "primitive : indication". Neste instante o ciclo de primitivas do serviço é encerrado.

4.4. Deleção da Variável definida

O Proc_AP do Controlador solicita a deleção da Variável

definida no Robô, interagindo com o Proc_API para a solicitação do serviço "Delete_Variable_Access". O Proc_API envia ao Proc_MMS uma primitiva do serviço confirmado com a mesma máscara descrita no item 4.2.1, onde o "invoke_id" pode ser o mesmo (0) ou outro valor qualquer, conforme seleção do Proc_API (no caso foi escolhido o valor 2), e o parâmetro "service_name : dvaccess".

O tratamento e o encaminhamento da primitiva é o mesmo descrito nos itens de 4.2.1. a 4.2.4., seguindo a seqüência Proc_API → Proc_MMS → Proc_Presentation → Proc_OP → Proc_Presentation → Proc_MMS → Proc_API. Entretanto, antes que o ciclo de resposta seja desencadeado pelo Proc_AP do Robô, o Proc_AP do Controlador solicita o cancelamento deste serviço, descrito no item seguinte.

4.5. Cancelamento do Serviço de Deleção da Variável

Quando o Proc_AP do Controlador solicita o cancelamento do serviço, o Proc_API desencadeia uma seqüência de mensagens do serviço "Cancel", estando ainda pendente a resposta do serviço "Delete_Variable_Access".

4.5.1. Primitiva "Cancel_request"

O Proc_API envia a primitiva "Cancel_request" ao Proc_MMS, que trata a mensagem com a máscara do serviço de gerenciamento de contexto, contendo os seguintes parâmetros :

- connection_id : 0
- primitive : request
- service : mms_cont
- data_area_lenght :

```
- data : mms_cont_serv :
    - service_name : cancel
    - dummy [5] :
    - prim_cont_manag : cancel_serv :
        - original_invoke_id : 2
```

O parâmetro "original_invoke_id : 2" identifica o "invoke_id" do serviço em cancelamento (item 4.4.).

O Proc_MMS envia o mesmo bloco de mensagem recebido ao Proc_Presentation, com a máscara da primitiva "P_data_request", contendo a PDU "Cancel_request". A seqüência de mensagens Proc_MMS → Proc_Presentation → Proc_OP → Proc_Presentation → Proc_MMS é semelhante à descrita nos items 4.2.2. e 4.2.3., porém para a PDU "Cancel_request".

4.5.2. Primitiva "Cancel_indication"

O Proc_MMS ao receber a primitiva "P_data_indication", contendo a PDU "Cancel", envia ao Proc_API a mensagem recebida com a mesma máscara do item 4.5.1., porém com a alteração nos parâmetros para "connection_id : 19" e "primitive : request".

4.5.3. Respostas ao Cancelamento do Serviço

O Proc_API interage com o Proc_AP do Robô, que responde positivamente ao cancelamento do serviço e negativamente à solicitação do serviço cancelado. O Proc_API envia ao Proc_MMS a primitiva "Cancel_response", que a trata com a máscara contendo os seguintes parâmetros :

```
- connection_id : 19
```

- primitive : response_pos
- service : mms_cont
- data_area_lenght :
- data : mms_cont_serv :
    - service_name : cancel
    - dummy [5] :
    - prim_cont_manag : cancel_serv :
        - original_invoke_id : 2

Em seguida o Proc_API envia ao Proc_MMS a primitiva "Delete_Variable_Access_response", negativa, com a máscara contendo os seguintes parâmetros :

- connection_id : 19
- primitive : response_neg
- service : mms_conf
- data_area_lenght :
- data : mms_conf_serv :
    - dummy [7] :
    - conf_neg_res :
        - invoke_id : 2
        - error :
            - error_class : (service_preempt)
            - error_code : (cancel)
            - additional_code_valid : 0
            - additional_code :
            - additional_description_valid : 0
            - additional_description :

Estas duas primitivas seguem a seqüência Proc_API → Proc_MMS → Proc_Presentation → Proc_OP → Proc_Presentation → Proc_MMS → Proc_API, perfazendo o ciclo de resposta postiva e negativa para os serviços "Cancel" e "Delete_Variable_Access", respectivamente, encerrando no Proc_AP do Controlador o ciclo destes passos no exemplo.

4.6. Leitura do valor da Variável

O Proc_AP do Controlador solicita o serivço "Read" para obtenção do valor armazenado na Variável definida no item 4.2., cuja "deleção" foi "cancelada". O que o exemplo mostra nesta seqüência é a inserção de erro no conteúdo da PDU, colocado pelo Proc_Presentation na comunicação com o Proc_OP. A seqüência inicia com o envio, pelo Proc_API, da primitiva "Read_request" ao Proc_MMS, que a trata com a máscara de serviço confirmado, semelhante a do item 4.2.1, com "invoke_id : 1" e "service_name : read". O Proc_MMS encaminha a mensagem ao Proc_Presentation com a máscara da primitiva "P_data_request", semelhante a do item 4.2.2.. O Proc_Presentation envia a "Solicitação de Transferência de PDU" ao Proc_OP, que a responde com inserção de erro de "invoke_id". A mensagem contêm os seguintes parâmetros :

- msg : pdu_transf_res
- user_msg : transf_res :
    - insert_error : 1 (true)
    - error_type : invoke_id_error
    - error_par : new_invoke_id : 3

O Proc_Presentation altera o valor do "invoke_id" da mensagem "P_data_request", que ele estava armazenando, e a envia ao Proc_MMS como "P_data_indication", como descrito no item 4.2.3.

4.7. Rejeição da PDU do serviço "Read"

O Proc_MMS trata a primitiva "P_data_indication" recebida do Proc_Presentation, e detectando a ocorrência do erro, rejeita a PDU, enviando ao Proc_API uma primitiva "Reject_indication", e ao Proc_Presentation uma primitiva "P_data_request", contendo a PDU

"Reject", especificando, nas duas mensagens, a causa da rejeição. A máscara da primitiva "Reject_indication" contêm os seguintes parâmetros :

- connection_id : 19
- primitive : indication
- service : mms_cont
- data_area_lenght :
- data : mms_cont_serv :
    - service_name : reject
    - dummy [5] :
    - prim_cont_manag : reject_serv :
        - detected_here : 1 (true)
        - original_invoke_id_valid : 0 (false)
        - original_invoke_id :
        - reject_class : invalid_request
        - reject_code : unrec_invoke_id

Ao receber a PDU "Reject", O Proc_MMS, no contexto relativo ao AP que solicitou o serviço, envia também ao Proc_API (conexão associada ao Proc_AP do Controlador, com "connection_id : 0") uma primitiva "Reject", com a mesma causa e o mesmo código de erro acima, porém com parâmetro "detected_here : 0 (false)".

4.8. Fechamento do Contexto entre os AP's

O Proc_AP do Controlador solicita o fechamento do contexto com o Proc_AP do Robô através do serviço "Conclude". O Proc_API envia ao Proc_MMS a primitiva "Conclude_request", que a trata com a máscara de serviço de gerenciamento de contexto, com os seguintes parâmetros :

- connection_id : 0

- primitive : request
- service : mms_cont
- data_area_lenght :
- data : mms_cont_serv :
    - service_name : conclude
    - dummy [5] :
    - prim_cont_manag : conclude_serv :

A seqüência de mensagens do serviço "Conclude" é a mesma para os serviços confirmados, como mostrado no item 4.2., ou seja, o mapeamento da PDU "Conclude" é feito diretamente na primitiva "P_data", sem a participação do Protocolo ACSE.

4.8.1. Primitiva "A_release_request"

O Protocolo ACSE somente participa do encerramento do contexto quando o serviço "Conclude" é completado com sucesso (Proc_MMS enviou a primitiva "Conclude_confirm" positiva ao Proc_API). Neste caso o Proc_MMS ao liberar o contexto envia ao Proc_ACSE a primitiva "A_release_request", que a trata com a seguinte máscara :

- connection_id : 0
- primitive : request
- service : acse
- data_area_lenght :
- data : acse_serv :
    - service_name : a_release
    - prim_acse : a_release_serv :
        - a_release_req :
            - dummy [2] :
            - reason_valid : 1 (true)
            - reason : normal

- user_information_valid : 0 (false)
- user_information :

4.8.2. Primitiva "P_release_request"

O Proc_ACSE trata a mensagem recebida e a encaminha ao Proc_Presentation, que a trata com a máscara da primitiva "P_release_request", contendo os seguintes parâmetros :

- connection_id : 0
- primitive : request
- service : presentation
- data_area_lenght :
- data : pres_serv :
    - service_name : p_release
    - prim_pres : p_release_serv :
        - p_release_req :
            - dummy :
            - user_data_valid : 1 (true)
            - user_data :

O Proc_Presentation trata a mensagem recebida, dando continuidade a seqüência de primitivas no sentido Proc_OP →Proc_Presentation → Proc_ACSE → Proc_MMS. O Proc_MMS envia a resposta através da primitiva "A_release_response", desencadeando a seqüência inversa.

4.8.3. Liberação do Contexto

As primitivas de "response" e "confirm" do serviço "Release" provocam a liberação do contexto nos protocolos, quando tratadas pelo Proc_Presentation (como mostra o item 3.5.2.7.2.), Proc_ACSE e Proc_MMS. Neste instante é desfeita a associação dos "connection_id" 0 e 19, iniciada no estabelecimento do contexto (item 4.1.).

5. CONCLUSÕES

O Protocolo MMS tornou-se, no contexto de automação industrial, elemento primordial de integração da manufatura. A sua implementação apresenta uma complexidade inerente a sua distribuição funcional discutida nesta tese. Assim é de vital importância a definição de um modelo de implementação, como o proposto para o SISDI_MAP, que suportado por um núcleo em tempo real, forneça um ambiente adequado à comunicação de serviços entre equipamentos de manufatura.

O modelo de implementação está centrado numa estrutura de comunicação que :

- Utiliza o conceito de comunicação através de "mailboxes", bastante discutido atualmente, e que não só supre as necessidades da implementação do SISDI_MAP, como também oferece um ambiente genérico de implementação de protocolos;
- Permite aos processos que implementam os protocolos, a passagem de apontadores na hierarquia do sistema (Proc_API → Proc_MMS → Proc_ACSE → Proc_Presentation e vice_versa), sem que haja cópia de informação entre "buffers". Este mecanismo foi possível pela definição de diversas máscaras para o mesmo bloco de informação, utilizadas por cada processo, conforme a suas necessidades de manipulação dos dados;
- Não faz distinção, dentro de uma máscara, entre os parâmetros das Primitivas e das PDU's, uma vez que o SISDI_MAP não se utiliza da sintaxe abstrata de regras de codificação do MMS, no caso a ASN.1.

Quanto as máscaras definidas no modelo, conclui-se que :

- Eliminam a desvantagem da cópia de parâmetros de um bloco para outro, na troca de mensagens entre processos, que no

pior caso, de estabelecimento de um contexto MMS, poderia chegar a seis cópias de parâmetros de um bloco para outro;
- Constituem o ponto crítico no sistema, pois são susceptíveis à modificações no tamanho dos parâmetros das primitivas e PDU's;
- Não perderão a sua funcionalidade com a expansão do SIDSI_MAP, já em andamento, através da implementação do codificador ASN.1 e das Camadas de Apresentação e Sessão.

Outro aspecto importante no modelo foi a possibilidade de na simulação da Camada de Apresentação, apresentada e discutida nesta tese, construir um ambiente de comunicação transparente aos protocolos da Camada de Aplicação. Este ambiente emula a rede de comunicação, permitindo ao SISDI_MAP a simulação dos possíveis erros que ocorrem numa situação real de comunicação de Processos Aplicativos dos sistemas de manufatura. Após a implementação da Camada de Apresentação, esta simulação pode ser transferida para a Camada de Sessão, e assim sucessivamente, utilizando-se dos mesmos conceitos de estruturação da comunicação presentes no SISDI_MAP.

O modelo de comunicação proposto permitiu também a definição de um mecanismo de testes dos processos, utilizando a mesma estrutura de comunicação através das máscaras, de forma abrangente e transparente, o que veio a facilitar a depuração do código implementado.

A operação do SISDI_MAP pode ser facilmente executada e compreendida, seguindo-se o exemplo de utilização dos serviços, descrito no Capítulo 4. O exemplo não corresponde necessariamente a seqüência e tipos de serviços que são executados num sistema real, entretanto abrange diversas situações, especificadas no Protocolo MMS, que podem ocorrer neste ambiente, e detalha as máscaras implementadas.

Através deste modelamento o SISDI_MAP pode vir futuramente a atingir o objetivo maior de implementação de um sistema real de comunicação entre equipamentos, conforme as especificações MMS definidas no MAP.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] ISO/DIS 9506 - "Manufacturing Message Specification, Part 1 : Service Specification, Part 2 : Protocol Specification", DRAFT 6, Agosto/1988.

[2] ISO/IS 7498 - "Information Processing System - Open System Interconnection - Basic Reference Model", Outubro/1985.

[3] GM - MAP 3.0 - "MAP Specification" Abril/1987.

[4] LEITE, J. R. E. e outros - MAP : A interconexão de sistemas em Automação Industrial, Revista Máquinas e Metais, Outubro/1986.

[5] PAGLIONI A. J, e outros - Protocolo RS-511 - Um modelo de Implementação, Anais 7o. Simpósio Brasileiro de Redes de Computadores, Março/1989.

[6] PAGLIONI, A. J. e outros - SISDI_MAP - Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP, Anais do Simpósio Franco-brasileiro em Sistemas Informáticos Distribuídos, Setembro/1989.

[7] GM - MAP 2.1 - "Map Specification", Março/1985.

[8] ISO/DIS 8571 - "Information Processing System - Open Systems Interconnection - File Transfer, Access and Management", Junho/1986.

[9] DWYER, J. e IOANNOU, A. - "MAP and TOP - Advanced Manufacturing Communications" - Kogan Page, 1a. edição, 1987.

[10] ISO/DIS 8824 - "Information Processing Systems - Open System Interconnection - Specification of Abstract Syntax Notation One (ASN.1)", Setembro/1986.

[11] MENDES, M. J. e outros - Interconexão de Sistemas Computacionais Abertos em Automação Industrial, Revista SBA - Controle e Automação, Vol. 1, No. 1, Janeiro/1987.

[12] MAP 3.0 : "MMS Application Interface Specification", Abril/1987.

[13] PAGLIONI, A. J. e outros - Aspectos de Implementação do Protocolo RS-511 do MAP, Anais do 3o. CONAI, Setembro/1988.

[14] ISO/DIS 8649 - "Information Processing System - Open System Interconnection - Definition of Association Control Service Element", Part 2, Abril/1986.

[15] ISO/DIS 8822 - "Information Processing Systems - Open Systems Interconection - Connection Oriented Presentation Service Definition", Junho/1986.

[16] ZABEU, M.C. - Um modelo de núcleo, Tese de Mestrado em andamento, FEE-UNICAMP, Outubro/1989.

[17] LIMA, J. M. S, - Uma Interface de Comunicação Homem-Máquina para execução de Protocolos, Tese de Mestrado em andamento, FEE-UNICAMP, Outubro/1989.

[18] MADEIRA, E. R. M. e MENDES, M.J. - Interfaces de Programas de Aplicação para o Protocolo MMS (RS-511), Anais do 3o. CONAI, Setembro/1988.

[19] FERNANDES, I. A. - Implementação do Protocolo MMS para um Sistema Didático, Tese de Mestrado em andamento, FEE-UNICAMP, Outubro,/1989.

[20] PAGLIONI, A. J. - Implementação do Protocolo ACSE para um Sistema Didático, Tese de Mestrado em andamento, FEE-UNICAMP, Outubro,/1989.

[21] MENDES, M. J. - Comunicação Fabril e o Projeto MAP/TOP, IV Escola Brasil-Argentina de Informática (EBAI), Janeiro/1989.

[22] ISO/TC97/SC21 N1494 - Information Processing Systems - Open Systems Interconnection - Aplication Layer Structure".

[23] ISO/DP 8831 - "Information Processing Systems - Open Systems Interconnection - Job Transfer and Manipulation, Concepts and Services", Outubro/1985.

[24] KIRNER, C. e MENDES, S.B.T. - Sistemas Operacionais Distribuídos, Aspectos Gerais e Análise de sua Estrutura, Editora Campus, 1a. edição, 1988.

[25] HALSALL, F. - "Data Comunication, computer Networks and OSI", Addison_wesley, 2a. edição, 1988.

7. ANEXOS

7.1. Anexo I : Listagem do arquivo de Mensagens do SISDI_MAP

```
  /* ARQUIVO DE MENSAGENS NAS INTERFACES DO SISTEMA DIDATICO */


/* OBS. : Todos os tipos declarados como apontador para "char"
/*        terão um tamanho limitado pelo SISDI_MAP */

  /* BLOCO DE MENSAGENS  PROTOCOLO */


/* DEFINIÇÃO DE CONSTANTES */

  /* Configuração do MMS */
#define Max_outstanding 16 /* num. máximo de serviços pendentes */
#define Max_context 8 /* num. máximo de contextos estabelecidos
                                                    por  usuario */
#define Max_nesting_level 5 /* num. máximo de niveis de
                                                      aninhamento */
#define Max_data 100 /* num. máximo de bytes dos serviços MMS */

  /* Configuração Geral */
#define Max_association 8 /* num. máximo de associacoes por AE */
#define Max_connection_id 128 /* num. máximo de connection_id
                                          permitido no  SISDI_MAP */
#define Max_msg_size 150 /* tamanho máximo das mensagens */

  /* Configuraçã do Presentation */
#define Max_presentation 4 /* num. máximo de Entidades de
                                                    Apresentação */
```

```
/* DEFINIÇÃO DE TIPOS SIMPLES */

typedef unsigned char  uint8;  /* 1 byte  : 0 a 255 */
typedef unsigned int  uint16; /* 2 bytes : 0 a 65535*/
typedef unsigned long int uint32; /* 4 bytes : 0 a (2**32 - 1) */
typedef char   sint8;  /* 1 byte  : -128 a 127 */
typedef int   sint16; /* 2 bytes : -32768 a 32767 */
typedef long int  sint32; /* 4 bytes : -2**31 a (2**31 - 1) */

     /* CAMPOS DEFINIDOS COMO OPCIONAIS SÃO BOOLEANOS */

typedef unsigned char            Boolean;


     /* CAMPOS NÃO UTILIZADOS POR UMA MÁSCARA */

typedef sint8   Dummy;


     /* Identificador da Conexão */

typedef uint16   Connection_id;

 /* Identificador da transação de serviço */

typedef uint16   Invoke_id;

 /* Lista de números de versões */
typedef  uint8   List_version_numbers[8];

 /* Endereco de Apresentação */
typedef uint16   P_address;

/* DEFINIÇÃO DE TIPOS  ENUMERADOS */
  /* Tipos de primitivas utilizados, conforme Modelo OSI  */

typedef enum
    {
     request, indication, response_pos, response_neg, confirm_pos,
```

```
    confirm_neg
   }Primitive_type;
/* Serviços dos Protocolos do SISDI_MAP :
          - MMS confirmado : mms_conf
          - MMS nao confirmado : mms_unconf
          - Contexto MMS : mms_cont
          - ACSE : acse
          - Apresentação  : presentation */
typedef enum
   {
    mms_conf, mms_unconf, mms_cont, acse, presentation
   }Service_type;

  /* Serviços Confirmados : Se o serviço poaaui um nomw composto por
     mais de uma palavra, a relação o apresenta formado com as iniciais
     das primeiras palavras, seguida pela última palavra, com exceção
     para os casos de coincidência */
typedef enum
   {
    status, gnlist, identify, renamevar, read, write, gvaattributes,
    dnvariable, dsaccess, gsaattributes, dvaccess, definenvlist,
    gnvlattributes, dnvlattributes, deletenvlist, definentype,
    gntattributes, deletentype, input, output, tcontrol, rcontrol,
    definesemaphore, deletesemaphore, rsstatus, rpsstatus, rsestatus,
    idlsequence, ulsegment, dlsegment, tdlsequence, iulsequence,
    tulsequence, rddload, rduload, ldcontent, sdcontent, ddomain,
    gdattribute, cpinvocation, dpinvocation, start, stop, resume,
    reset, kill, gpiattribute, ofile, defineecondition,
    deleteecondition, gecattributes, reestatus, aecmonitoring, tevent,
    defineeaction, deleteeaction, geaattributes, reastatus,
    defineeenrollment, deleteeenrollment, aeenrollment, geenrollments,
    aenotification, aecondition, gasummary, gaesummary, rjournal,
    wjournal, ijournal, rjstatus, fileopen, fileread, fileclose,
    frename, fdelete, fdirectory, asrequest
   }Mms_confirmed_service;

  /* Serviços nao Confirmados : A apresentação seque a mesma regra
```

```
     do serviços confirmados */

typedef enum
     {
      ireport, ustatus, enotification, auservice
     }Mms_unconfirmed_service;

  /* Serviços de Contexto */

typedef enum
     {
      initiate, conclude, abort_mms, reject, cancel
     }Mms_context_manag_service;

  /* Serviços do ACSE */

typedef enum
     {
      a_associate, a_release, a_abort, a_p_abort
     }Acse_service;


       /* Serviços de Apresentação */

typedef enum
     {
      p_connection, p_release, p_u_abort, p_p_abort, p_data
     }Presentation_service;

  /* Classes de Rejeição na Primitiva "Reject" */

typedef enum
 {
  invalid_request, invalid_response, invalid_error_response, pdu_error
 }Reject_class;

  /* Códigos de Rejeição na Primitiva "Reject" */
```

```
typedef enum
 {
  unknown_pdu_type, mistyped_pdu, invalid_pdu, unspecified,
          duplicate_invoke_id, unrec_service, mistyped_modifier,
          mistyped_argument, max_invoc_exceeded, excessive_lng_pdu,
          value_out_of_range, excessive_recursion, unrec_invoke_id,
          result_resp_unexpected, service_mismatch, mistyped_result,
          error_resp_unexpected, mistyped_error
         }Reject_code;
  /* Resultados de Associate na primitiva de resp/conf do ACSE */

typedef enum
 {
  accepted, rejected_permanent, rejected_transient
 }Assoc_result;

  /* Diagnóstico de Associate na primitiva de resp/conf do ACSEc */

typedef enum
 {
  no_reason_given, no_common_acse_version,
  application_cont_name_not_supported,
  calling_ap_title_not_recognized,
  calling_ae_qualifier_not_recognized,
  calling_ap_invoc_ident_not_recognized,
  calling_ae_invoc_ident_not_recognized,
  called_ap_title_not_recognized,
  called_ae_qualifier_not_recognized,
  called_ap_invoc_ident_not_recognized,
  called_ae_invoc_ident_not_recognized
 }Assoc_diagnostic;


       /* Fontes de Resultado de Associate ou Abort */
     nas primitivas indication ou confirm do ACSE */

typedef enum
```

```
 {
  acse_service_user, acse_service_provider,
  presentation_service_provider
 }Source;

  /* Resultados da primitiva de resp/conf de A_Release */

typedef enum
 {
  release_accepted, release_rejected
 }A_release_result;

/* Resultados da primitiva de resp/conf de P_Release */

typedef enum
 {
  affirmative, negative
 }P_release_result;

  /* Causas de Release na primitiva A_Release */

typedef enum
 {
  normal, urgent, not_finished, user_defined
 }Release_reason;

  /* Resultados na primitiva P_connection ind/conf */

typedef enum
 {
  acceptance, user_rejection, provider_rejection
 }P_con_result;

/* Tipos de PDU MMS */

typedef enum
 {
```

```
    confirmed_request, confirmed_response, confirmed_error,
    unconfirmed_request, reject_pdu, cancel_request,
    cancel_response, cancel_error, initiate_request,
     initiate_response, initiate_error, conclude_request,
     conclude_response, conclude_error
          }Pdu_type;


/* DEFINIÇÃO DE TIPOS ESTRUTURADOS DE BITS */

  /* Tipos de CBB's - "Conformance Building Blocks" : Cada CBB ocupa
     um bit da estrutura, que representa a sua presença ou não nos
     parâmetros de horizontal e vertical CBB das primitivas e PDU's
     "Initiate" */

     /* CBB Horizontal */

typedef struct
 {
  unsigned str1:1;
  unsigned str2:1;
  unsigned vnam:1;
  unsigned valt:1;
  unsigned vadr:1;
  unsigned vsca:1;
  unsigned socs:1;
  unsigned rcvl:1;
  unsigned btrl:1;
}Horizontal_cbb;

  /* CBB Vertical */

typedef struct
 {
  unsigned con1:1;
  unsigned con2:1;
```

```
unsigned con3:1;
unsigned vmd1:1;
unsigned vmd2:1;
unsigned mms1:1;
unsigned mms2:1;
unsigned ocs1:1;
unsigned ocs2:1;
unsigned sem1:1;
unsigned sem2:1;
unsigned sem3:1;
unsigned dom1:1;
unsigned dom2:1;
unsigned dom3:1;
unsigned dom4:1;
unsigned dom5:1;
unsigned dom6:1;
unsigned dom7:1;
unsigned dom8:1;
unsigned evn1:1;
unsigned evn2:1;
unsigned evn3:1;
unsigned evn4:1;
unsigned evn5:1;
unsigned evn6:1;
unsigned evn7:1;
unsigned var1:1;
unsigned var2:1;
unsigned var3:1;
unsigned var4:1;
unsigned var5:1;
unsigned var6:1;
unsigned var7:1;
unsigned var8:1;
unsigned var9:1;
unsigned jou1:1;
unsigned jou2:1;
```

```
  unsigned prg1:1;
  unsigned prg2:1;
  unsigned prg3:1;
  unsigned prg4:1;
  unsigned prg5:1;
  unsigned prg6:1;
  unsigned prg7:1;
  unsigned fil1:1;
  unsigned fil2:1;
  unsigned fil3:1;
 }Vertical_cbb;

 /* Número de Versões do Protocolo ACSE : Cada bit presenta uma
    versão implementada pelo Protocolo ACSE */

typedef struct
 {
  unsigned version1:1;
  unsigned version2:1;
  unsigned version3:1;
  unsigned version4:1;
  unsigned version5:1;
  unsigned version6:1;
  unsigned version7:1;
  unsigned version8:1;
 }Prot_version;


/* TIPOS ESTRUTURADOS */
 /* Estrutura de definição de Nome de Contexto de Apresentação */

typedef struct
 {
  char   abstract_syntax_name [4];
  char   transfer_syntax_name [4];
 }Pres_cont_name;
```

```
 /* Estrutura de definição de Lista de Contexto de Apresentação */
typedef struct
 {
  uint8  pres_context_ident;
  char   abstract_syntax_name [4];
  char   transfer_syntax_name [4] [4];
 }Pres_cont_def;

 /* Estrutura de identificador de Conexão de Sessão */

typedef struct
 {
  char   id1 [8];
  char   id2 [8];
  char   id3 [4];
 }Session_con_id;

 /* Estrutura de "request/indication" confirmado */

typedef struct
 {
  Invoke_id                invoke_id;
  Boolean                  modifier_valid; /* Modificador  */
  sint8                    modifier;
  Mms_confirmed_service    service_name; /* Servico confirmado */
  sint8                    conf_req_data[Max_data]; /* Parâmetros */
 }Confirmed_request_struct;

 /* Estrutura de "response/confirm" positiva */

typedef struct
 {
  Invoke_id                invoke_id; /* Identificador da Transação */
  Mms_confirmed_service    service_name; /* Serviço confirmado */
  sint8                    conf_res_data[Max_data]; /* Parâmetros */
```

```
}Confirmed_pos_res_struct;

/* Estrutura de "response/confirm" negativa : diagnóstico de erro */

typedef struct
 {
  uint8  error_class;
  uint8  error_code;
  Boolean additional_code_valid;
  uint8  additional_code;
  Boolean additional_description_valid;
  char  additional_description [30];
 }Error_block;  /* tamanho 35 bytes */
/* Estrutura de "response/confirm" negativa */

typedef struct
 {
  Invoke_id           invoke_id; /* Identificador da Transação */
  Error_block   error;
 }Confirmed_neg_res_struct;

 /* Estrutura de Serviços Confirmados  na Interface API/MMS */

typedef struct
 {
  Dummy     dummy [7];
  union
       {            /* Seleção conforme o tipo de Primitiva */
        Confirmed_request_struct   conf_request;
        Confirmed_pos_res_struct   conf_pos_res;
        Confirmed_neg_res_struct   conf_neg_res;
       }prim_mms_conf;
 }Mms_conf_struct;


/* Estrutura de Serviços nao Confirmados  na Interface API/MMS */
```

```
typedef struct    /* PDU Unconfirmed_request */
 {
  Mms_unconfirmed_service   service_name;
  sint8                     unconf_req_data[Max_data];
 }Unconfirmed_request_struct;

  /* Primitiva unconfirmed_request/indication */
typedef struct
 {
  Dummy                             dummy [7];
  Unconfirmed_request_struct        unconf_request;
 }Mms_unconf_struct;

 /* Estrutura do serviço "Initiate" */

typedef struct     /* PDU Initiate_request */
 {
  List_version_numbers list_of_version_numbers;
  Boolean              prop_max_msg_size_valid;
  uint32               prop_max_msg_size;
  uint16               prop_max_serv_outst_calling;
  uint16               prop_max_serv_outst_called;
  Boolean              prop_data_str_nest_level_valid;
  uint8                prop_max_nesting_level;
  Horizontal_cbb       prop_horizontal_cbb;
  Vertical_cbb         client_vert_cbb_calling;
  Vertical_cbb         server_vert_cbb_calling;
 }Initiate_req_pdu;    /* tamanho : 33 bytes  */

typedef struct    /* PDU Initiate_response : resposta positiva */
 {
  uint8             version_number;
  Boolean           negot_max_msg_size_valid;
  uint32            negot_max_msg_size;
  uint16            negot_max_serv_outst_calling;
```

```
  uint16              negot_max_serv_outst_called;
  Boolean             negot_data_str_nest_level_valid;
  uint8               negot_data_str_nest_level;
  Horizontal_cbb      negot_horizontal_cbb;
  Vertical_cbb        client_vert_cbb_called;
  Vertical_cbb        server_vert_cbb_called;
 }Initiate_res_pdu;          /* tamanho : 26 bytes */


typedef union  /* Primitivas Initiate */
{
  struct
        {   /*  Initiate  request  */
         Dummy                dummy [256];
         Initiate_req_pdu     initiate_req_pdu;
        }initiate_req;
  struct
        {     /* Initiate response pos */
         Dummy                dummy [158];
         Initiate_res_pdu     initiate_res_pdu;
        }initiate_pos_res;
  struct
        {    /* Initiate response neg */
         Dummy                dummy [158];
         Error_block          initiate_error_pdu;
        }initiate_error;
 }Initiate_struct;

  /* Estrutura do serviço "Conclude" */

 typedef struct    /* sem parâmetros no request/indication e */
  {  /* no response/confirm positivo                  */
   Error_block  conclude_error;
  }Conclude_struct;
```

```
        /* Estrutura do serviço "Abort" */

 /* PDU Abort request */

typedef struct
 {
  Boolean  source; /* True se usuário remoto, False se Provedor MMS */
 }Abort_pdu;

 /* Primitiva Abort : somente parâmetros no indication */
typedef struct
 {
  Abort_pdu abort_ind;  /* Os mesmos parametros da PDU */
 }Abort_struct;

 /* Estrutura do serviço "Cancel" */

typedef union
 {
  Invoke_id  original_invoke_id; /* req/ind e resp positivo */
  struct
        {
          Invoke_id    original_invoke_id;
          Error_block  error;
        }cancel_error;
 }Cancel_struct;

 /* Estrutura do serviço "Reject" */

typedef struct
 {        /* somente parametros no indication */
  Boolean detected_here; /* Erro detectado no Provedor local */
  Boolean original_invoke_id_valid; /* Condicional se invoke_id */
  Invoke_id original_invoke_id;        /* e' valido */
  Reject_class   reject_class;
  Reject_code    reject_code;
```

```
 }Reject_struct;

/* Estrutura de Servicos de Contexto na Interface API/MMS */

typedef struct
 {
  Mms_context_manag_service  service_name;
  Dummy        dummy [5];
  union
       {
        Initiate_struct  initiate_serv;
        Conclude_struct  conclude_serv;
        Abort_struct  abort_serv;
        Cancel_struct cancel_serv;
        Reject_struct reject_serv;
       }prim_cont_manag;
 }Mms_cont_struct;


  /* Estrutura de PDU's MMS mapeadas na primitiva P_data      */
  /* Obs : As PDU's Initiate nao esta nesta estrutura porque */
  /* sao mapeadas na primitiva A_associate/P_Connection       */

typedef struct
 { /* As PDU's Conclude nao possuem parametros : tipo Dummy */
  Pdu_type  pdu;
  union
       {
        Confirmed_request_struct  confirmed_request_pdu;
        Confirmed_pos_res_struct  confirmed_response_pdu;
        Confirmed_neg_res_struct  confirmed_error_pdu;
        Mms_unconf_struct         unconfirmed_request_pdu;
        Reject_struct             reject_pdu;
        Cancel_struct             cancel_request_pdu;
        Cancel_struct             cancel_response_pdu;
        Cancel_struct             cancel_error_pdu;
```

```
        Dummy                       conclude_request_pdu;
         Dummy                      conclude_response_pdu;
         Error_block                conclude_error_pdu;
        }mms_pdu;
 }Mms_pdu_struct;

  /* Estrutura do serviço "A_associate" */

  /* PDU A_associate request */
typedef struct
 {
  Boolean           protocol_version_valid;
  Prot_version      protocol_version;
  Boolean           application_context_name_valid;
  char              application_context_name [4];
  Boolean           calling_AP_title_valid;
  char              calling_AP_title [5];
  Boolean           calling_AE_qualifier_valid;
  char              calling_AE_qualifier [5];
  Boolean           calling_AP_inv_ident_valid;
  uint16            calling_AP_inv_ident;
  Boolean           calling_AE_inv_ident_valid;
  uint16            calling_AE_inv_ident;
  Boolean           called_AP_title_valid;
  char              called_AP_title [5];
  Boolean           called_AE_qualifier_valid;
  char              called_AE_qualifier [5];
  Boolean           called_AP_inv_ident_valid;
  uint16            called_AP_inv_ident;
  Boolean           called_AE_inv_ident_valid;
  uint16            called_AE_inv_ident;
  Boolean           user_information_valid;
  Initiate_req_pdu user_information;
 }A_assoc_req_pdu;          /* tamanho : 77 bytes */

 /* PDU A_associate response */
```

```
typedef struct
{
  Boolean              protocol_version_valid;
  Prot_version         protocol_version;
  Boolean              application_context_name_valid;
  char                 application_context_name [4];
  Boolean              responding_AP_title_valid;
  char                 responding_AP_title [5];
  Boolean              responding_AE_qualifier_valid;
  char                 responding_AE_qualifier [5];
  Boolean              responding_AP_inv_ident_valid;
  uint16               responding_AP_inv_ident;
  Boolean              responding_AE_inv_ident_valid;
  uint16               responding_AE_inv_ident;
  Assoc_result         result;
  Source               result_source; /* so na primitiva confirm  */
  Boolean              diagnostic_valid;
  Assoc_diagnostic     diagnostic;
  Boolean              user_information_valid;
  union
    {
     Initiate_res_pdu initiate_res_pdu;    /* tamanho 26 bytes */
     Error_block      initiate_error;      /* tamanho 35 bytes */
    }user_information;
 }A_assoc_res_pdu;          /* tamanho maximo : 68 bytes */

 /* Primitiva A_associate request/indication */

typedef struct
 {
  Dummy                dummy;
  P_address            calling_pres_address;
  P_address            called_pres_address;
  Boolean              pres_context_definition_list_valid;
  Pres_cont_def        pres_context_definition_list [4];
```

```
  Boolean              pres_context_def_list_result_valid; /* so na ind */
  Pres_cont_def        pres_context_def_list_result [4]; /* N6 preenche */
  Boolean              default_pres_context_name_valid;
  Pres_cont_name       default_pres_context_name;
  uint8                quality_of_service;
  Boolean              pres_requeriments_valid;
  uint8                pres_requeriments;
  Boolean              mode_valid;
  uint8                mode;
  uint16               session_requeriments;
  uint32               initial_synchr_point_serial_number;
  char                 initial_assigment_of_tokens;
  Session_con_id       session_connection_ident;
  Dummy                dummy1;
  A_assoc_req_pdu      associate_pdu;
 )A_associate_req;   /* tamanho 294 bytes  */

 /* Primitiva A_associate response/confirm */

typedef struct
 {
  Dummy                dummy;
  P_address            responding_pres_address;
  Boolean              pres_context_definition_list_valid;
  Pres_cont_def        pres_context_definition_list [4];
  Boolean              default_pres_context_name_valid;
  Pres_cont_name       default_pres_context_name;
  uint8                quality_of_service;
  Boolean              pres_requeriments_valid;
  uint8                pres_requeriments;
  uint16               session_requeriments;
  uint32               initial_synchr_point_serial_number;
  char                 initial_assigment_of_tokens;
  Session_con_id       session_connection_ident;
  Dummy                dummy3 [3];
  A_assoc_res_pdu      associate_pdu;
```

```
 }A_associate_res; /* tamanho maximo 198 bytes */

typedef union
 {
  A_associate_req  a_associate_req;
  A_associate_res a_associate_res;
 }A_associate_struct;

 /* Estrutura do serviço "A_release" */

 /* PDU A_release */
typedef struct
 {
  Boolean  reason_valid;
  Release_reason  reason; /* Causa do Release */
  Boolean         user_information_valid;
  char            user_information [30]; /* Dados usuario */
 }A_release_pdu;

 /* Primitiva A_release request/indication */
typedef struct
 {
  Dummy           dummy [2];
  Boolean         reason_valid;
  Release_reason  reason; /* Causa do Release */
  Boolean         user_information_valid;
  char            user_information [30]; /* Dados usuario */
 }A_release_req;




/* Primitiva A_release response/confirm */

typedef struct
```

```
 {
  Dummy              dummy [4];
  Boolean            reason_valid;
  Release_reason     reason; /* Causa do Release */
  Boolean            user_information_valid;
  char               user_information [30]; /* Dados usuario */
  A_release_result   result; /* Resultado do Release */
 }A_release_res;

typedef union
 {
  A_release_req    a_release_req;
  A_release_res    a_release_res;
 }A_release_struct;

 /* Estrutura do serviço "A_abort" */

 /* PDU A_abort request */
typedef struct
 {
  Source      source;
  Boolean     user_information_valid;
  Abort_pdu   user_information;    /* PDU MMS */
 }A_abort_pdu;

 /* Primitiva A_abort request : Mapeia PDU Abort request do MMS */
typedef struct
 {
  Dummy       dummy [4];
  Boolean     user_information_valid;
  Abort_pdu   user_information;
  }A_abort_req;
```

```
 /* Primitiva A_abort indication : Mapeia PDU Abort_request do MMS */
typedef struct
 {
  Dummy       dummy [2];
  Source      source;
  Boolean     user_information_valid;
  Abort_pdu   user_information;
 }A_abort_ind;

typedef union
 {
  A_abort_req a_abort_req;
  A_abort_ind a_abort_ind;
 }A_abort_struct;

 /* Estrutura do serviço "A_p_abort" */
typedef struct
 {
  /* somente parâmetros no Indication */
  char provider_reason [30];
 }A_p_abort_struct;

 /* Estrutura de Serviços ACSE na Interface MMS/ACSE */

typedef struct
 {
  Acse_service service_name;
  union
       {
        A_associate_struct a_associate_serv;
        A_release_struct   a_release_serv;
        A_abort_struct     a_abort_serv;
        A_p_abort_struct   a_p_abort_serv;
        }prim_acse;
 }Acse_struct;
```

```
/* Estrutura do serviço "P_connection" */

typedef union
 {
  struct      /* Primitiva P_connection_request */
  {
   Dummy              dummy1;
   P_address          calling_pres_address;
   P_address          called_pres_address;
   Boolean            pres_context_definition_list_valid;
   Pres_cont_def      pres_context_definition_list [4];
   Dummy              dummy85 [85]; /* so no indication */
   Boolean            default_pres_context_name_valid;
   Pres_cont_name     default_pres_context_name;
   uint8              quality_of_service;
   Boolean            pres_requeriments_valid;
   uint8              pres_requeriments;
   Boolean            mode_valid;
   uint8              mode;
   uint16             session_requeriments;
   uint32             initial_synchr_point_serial_number;
   char               initial_assigment_of_tokens;
   Session_con_id     session_connection_ident;
   Boolean            user_data_valid;
   A_assoc_req_pdu    user_data;
  }p_connection_req;          /* tamanho 294 bytes */

  struct      /* Primitiva P_connection_indication */
  {
   Boolean            valid_pdu;
   P_address          calling_pres_address;
   P_address          called_pres_address;
   Boolean            pres_context_definition_list_valid;
   Pres_cont_def      pres_context_definition_list [4];
```

```
 Boolean          pres_context_def_list_result_valid;
 Pres_cont_def   pres_context_def_list_result  [4];
 Boolean          default_pres_context_name_valid;
 Pres_cont_name   default_pres_context_name;
 uint8            quality_of_service;
 Boolean          pres_requeriments_valid;
 uint8            pres_requeriments;
 Boolean          mode_valid;
 uint8            mode;
 uint16           session_requeriments;
 uint32           initial_synchr_point_serial_number;
 char             initial_assigment_of_tokens;
 Session_con_id   session_connection_ident;
 Boolean          user_data_valid;
 A_assoc_req_pdu  user_data;
} p_connection_ind;            /* tamanho 294 bytes */

struct      /* Primitiva P_connection_response */
{
 Dummy              dummy;
 P_address          responding_pres_address;
 Boolean            pres_context_def_list_result_valid;
 Pres_cont_def     pres_context_def_list_result  [4];
 Boolean            default_pres_context_name_valid;
 Pres_cont_name     default_pres_context_name;
 uint8              quality_of_service;
 Boolean            pres_requeriments_valid;
 uint8              pres_requeriments;
 uint16             session_requeriments;
 uint32             initial_synchr_point_serial_number;
 char               initial_assigment_of_tokens;
 Session_con_id     session_connection_ident;
 P_con_result       result;
 Boolean            user_data_valid;
 A_assoc_res_pdu    user_data;
}p_connection_res;             /* tamanho 198 bytes */
```

```
  struct      /* Primitiva P_connection_confirm */
  {
   Boolean           valid_pdu;
   P_address         responding_pres_address;
   Boolean           pres_context_def_list_result_valid;
   Pres_cont_def   pres_context_def_list_result  [4];
   Boolean           default_pres_context_name_valid;
   Pres_c ont_name   default_pres_context_name;
   uint8             quality_of_service;
   Boolean           pres_requeriments_valid;
   uint8             pres_requeriments;
   uint16            session_requeriments;
   uint32            initial_synchr_point_serial_number;
   char              initial_assigment_of_tokens;
   Sessi o n_con_id  session_connection_ident;
   P_con_result      result;
   Boolean           user_data_valid;
   A_assoc_res_pdu   user_data;
  }p_connection_cnf;              /* tamanho 198 bytes */
 }P_connection_struct;

 /* Estrutura do serviço "P_release" */
typedef union
 {
  struct    /* primitiva request : transporta PDU A_release request */
  {
   Dummy           dummy;
   Boolean         user_data_valid;
   A_release_pdu   user_data;
  }p_release_req;
  struct /* primitiva indication : transporta PDU A_release request */
  {
   Boolean         valid_pdu;
   Boolean         user_data_valid;
   A_release_pdu   user_data;
```

```
  }p_release_ind;
  struct   /* primitiva response : transporta PDU A_release response */
  {
   Dummy    dummy;
   P_release_result  result;
   Boolean           user_data_valid;
   A_release_pdu     user_data;
  }p_release_res;
  struct   /* primitiva confirm : transporta PDU A_release_response */
  {
   Boolean           valid_pdu;
   P_release_result  result;
   Boolean           user_data_valid;
   A_release_pdu     user_data;
  }p_release_cnf;
 }P_release_struct;

 /* Estrutura do serviço "P_U_abort" */

typedef union
 {
  struct     /* primitiva request : transporta PDU A_abort_request */
  {
   Dummy         dummy;
   Boolean       user_data_valid;
   A_abort_pdu   user_data;
  }p_u_abort_req;
  struct     /* primitiva indication : transporta PDU A_abort_request */
  {
   Boolean        valid_pdu;
   Boolean        user_data_valid;
   A_abort_pdu    user_data;
  }p_u_abort_ind;
 }P_U_abort_struct;

 /* Estrutura do serviço "P_P_abort" */
```

```
typedef struct
 {   /* somente primitiva indication */
  char  provider_reason [30];
 }P_P_abort_struct;

/* Estrutura do servico "P_data" */
typedef union
 {
  struct    /* primitiva request : transporta PDU's request */
  {
   Dummy          dummy  [3];
   Mms_pdu_struct user_data;
  }p_data_req;
  struct  /* primitiva indication : transporta PDU's response/error */
  {
   Dummy   dummy  [2];
   Boolean valid_pdu;
   union
        {
         Mms_pdu_struct  user_data; /* se PDU valida */
         struct                     /* se PDU invalida */
        {
         Reject_class reject_class;
         Reject_code  reject_code;
        }invalid_pdu;
        }select_ind;
  }p_data_ind;
 }P_data_struct;

 /* Estrutura dos serviços da Camada de Apresentação
    nas Interfaces ACSE/N6 e MMS/N6 */
typedef struct
 {
  Presentation_service  service_name;
  union
```

```
        {
        P_connection_struct p_connection_serv;
         P_release_struct p_release_serv;
         P_U_abort_struct p_u_abort_serv;
         P_P_abort_struct p_p_abort_serv;
         P_data_struct  p_data_serv;
        }prim_pres;
 }Presentation_struct;


  /* ESTRUTURA PRINCIPAL DO BLOCO DE MENSAGENS DE PROTOTOCO */

typedef struct
        {
         Connection_id   connection_id;
         Primitive_type  primitive;
         Service_type    service;
         sint16          data_area_lenght;  /* maior primitiva ocupa 296
                                                bytes */
         union   /* Area de Dados de cada tipo de servico */
               {
                Mms_conf_struct      mms_conf_serv;
                Mms_unconf_struct    mms_unconf_serv;
                Mms_cont_struct      mms_cont_serv;
                Acse_struct          acse_serv;
                Presentation_struct pres_serv;
               }data;
        }Block_struct;


  /* BLOCO DE MENSAGENS DE CONTROLE E INTERFACE */


   /* Erros de Comunicação possíveis na Simulação de Apresentação */

typedef enum
```

```
 {
  service_error, invoke_id_error, invalid_pdu_error,
  pres_error
 }Communication_errors;

  /* Solicitação para transferência de PDU */

typedef struct
 {
  Block_struct  *pdu_block;
 }Pdu_transf_req;

/* Resposta para transferência de PDU */

typedef struct
 {
  Boolean  insert_error;
  Communication_errors error_type;
  union
       {
        Mms_confirmed_service      new_conf_serv;
        Mms_unconfirmed_service    new_unconf_serv;
        Invoke_id                  new_invoke_id;
       }error_par;
 }Pdu_transf_res;

     /* Seleção de Comunicação : Abort pelo Usuário, solicitação ou
     resposta de  transferência de PDU */

typedef enum
 {
  abort_by_user, pdu_transf_req, pdu_transf_res
 }Select_msg_user_interface;

typedef struct
 {
```

```
  Select_msg_user_interface  msg;
  union
        {
         Pdu_transf_req    transf_req;  /* se solicitação de tranf */
         Connection_id connection_aborted; /* se abort_by_user */
         Pdu_transf_res transf_res;  /* se resposta de tranf */
        }user_msg;
}User_com_block;
```

7.2. Anexo II : Listagem da programação do Proc_Presentation

```
 /* ESPECIFICAÇÃO  DETALHADA DE REALIZAÇÃO DO Proc_PRESENTATION */

/* OBS : A chamada dos procedimentos "call_error" representa
         condições de erro que ocorreram na depurção do código
         ou podem ocorrer no sistema, e provoca notificação
          ao operador */

#include "gl.h"  /* Tipos Globais do Núcleo */

#include "master.arq" /* Arquivo de mensagem do SISDI_MAP - ANEXO I */

#include <stdio.h> /* Biblioteca das primitivas da linguagem "C"
                      par interface de I/O */

#define NULL 0

 /* Estrutura de Dados do Proc_Presentation */

/* Connection_id nao usado na tabela de comutação */

#define not_used 255

/* Tipos Estruturados */

typedef struct
 {
  P_address  pres_address; /* Endereço de Apresentação */
  Connection_id  first_con_id; /* Primeiro connection_id do intervalo */
  Connection_id  last_con_id; /* Último connection_id do intervalo */
 }Con_id_range;


  /* Tabelas do Proc_Presentation */
```

```
 /* Tabela de Endereços de Apres. e Intervalos de Connection_id */

Con_id_range  pres_adr_tab [Max_presentation];


 /* Tabela de Associação de connection_id utilizados */

Connection_id  con_tab [Max_connection_id + 1]; /* zero nao usado */


  /* Variáveis do Proc_Presentation */

Block_struct *block; /* Bloco de mensagem de Protocolo */

User_com_block  *block_ctr; /* Bloco de mensagens de Controle */

Connection_id con_id = 0; /* Identificador da Conexão */


  /* Portas de Entrada e Saida */

static T_QUEUE *mbi_mms;
static T_QUEUE *mbi_acse;
static T_QUEUE *mbi_op;
static T_QUEUE *mbo_mms;
static T_QUEUE *mbo_acse;
static T_QUEUE *mbo_op;

 /* Temporizadores   em segundos */

#define T_wait_loop   1  /* espera no loop do programa principal */
#define T_wait_op     1000  /* espera de resposta do Proc_op */

Boolean false = 0;
Boolean true = 1;
```

```
  /* PROGRAMA PRINCIPAL */

p_presentation (T_QUEUE *ingate_mms, T_QUEUE  *ingate_acse,
                T_QUEUE  *ingate_op, T_QUEUE *outgate_mms,
                T_QUEUE *outgate_acse, T_QUEUE *outgate_op,
          P_address pres_adr [Max_presentation],
  Connection_id con_id_values [2 * Max_presentation])
  {
   mbi_mms = ingate_mms;
   mbi_acse = ingate_acse;
   mbi_op = ingate_op;
   mbo_mms = outgate_mms;
   mbo_acse = outgate_acse;
   mbo_op = outgate_op;
   initial(pres_adr, con_id_values);
   for (;;) /* loop eterno */
     if (espera_mens (mbi_op, T_wait_loop, CONSOME, &block_ctr) ==
         OPERACAO_OK)
       {
        treat_user ( );
       }
     else
       {
 if (espera_mens (mbi_acse, T_wait_loop, CONSOME, &block) ==
     OPERACAO_OK)
   {
    treat_acse ( );
   }
 else
   {
    if (espera_mens (mbi_mms, T_wait_loop, CONSOME, &block) ==
        OPERACAO_OK)
      {
```

```
        treat_mms ( );
        }
    }
  }
  } /* Fim do programa principal */


            /* INICIALIZAÇÃO DE VARIÁVEIS */

initial (P_address pres_adr [Max_presentation],
         Connection_id con_id_values [2 *  Max_presentation]
  {
   int i, j;
  /* Inicialização da Tabela pres_adr_tab */
   for (i = 0; i < Max_presentation; i++ )
     {
      pres_adr_tab [i].pres_address = pres_adr [i];
      j = i * 2;
      pres_adr_tab [i].first_con_id = con_id_values [j];
      pres_adr_tab [i].last_con_id = con_id_values [++j];
     };
  /* Inicialização da Tabela con_tab */
   for (i = 0; i <= Max_connection_id; i++) con_tab [i] = not_used;
  }


     /* TRATAMENTO DA MENSAGEM DE ABORTO DO USUARIO */

treat_user ( )
  {
   Connection_id con_id_out;
   if (block_ctr->msg == abort_by_user)
     {
      send_p_p_abort_ind (block_ctr->user_msg.connection_aborted);
      liberate_connection (block_ctr->user_msg.connection_aborted);
     }
   else call_error (38);
```

```
    libera_mem (block_ctr);
  }


/* TRATAMENTO DAS MENSAGENS DO ACSE */

treat_acse ( )
  {
   int i;
   Connection_id j = 0, first, last;
   Boolean error_detected = 0;
   con_id = block->connection_id;
   /* Verifica se mensagem é de nova conexão,se for preenche tabela */
   if ((block->primitive = request) && (block->data.pres_serv.
        service_name = p_connection))
      /* Obtêm endereço de Apresentação  do Chamador e
         intervalo de connection_id */
     {
      for (i = 0; i < Max_presentation; i++)
        if (pres_adr_tab [i].pres_address == block->data.pres_serv.
            prim_pres.p_connection_serv.p_connection_req.
            called_pres_address)
          {
           first = pres_adr_tab [i].first_con_id;
           last = pres_adr_tab [i].last_con_id;
           break;
          }
        /* Preenche Tabela de Associacao de Conexoes */
      j = last;
      while ((con_tab [j] != not_used) && (j > (last + first)/2)) j--;
       if (con_tab [j] != not_used) /* Nao ha connection_id livre  */
          {
           send_p_p_abort_ind (con_id);
           liberate_connection (con_id);
           error_detected = true;
          }
       else
```

```
        {
         con_tab [j] = con_id;
         con_tab [con_id] = j;
        }
    }
  if (!error_detected)
    {
     /* Envia transacao ao P_OP */
     pede_mem (sizeof (User_com_block), &block_ctr);
     block_ctr->msg = pdu_transf_req;
     block_ctr->user_msg.transf_req.pdu_block = block;
     envia_mens (mbo_op, block_ctr);
     if (espera_mens (mbi_op, T_wait_op, CONSOME, &block_ctr ==
         OPERACAO_OK))
       {
         treat_pdu_transf_res ();
       }
     else
       call_error (36);
    }
 }

    /* TRATAMENTO DAS MENSAGENS DO MMS */

treat_mms ( )
 {
  con_id = block->connection_id;
   /* Verifica connection_id */
  if (con_tab [con_id] == not_used)  /* não existe connection_id
                                        associado */
    call_error (31);
  else
    {
     /* Envia transacao ao P_OP */
     pede_mem (sizeof (User_com_block), &block_ctr);
     block_ctr->msg = pdu_transf_req;
     block_ctr->user_msg.transf_req.pdu_block = block;
```

```
      envia_mens (mbo_op, block_ctr);
      if (espera_mens (mbi_op, T_wait_op, CONSOME, &block_ctr ==
          OPERACAO_OK))
 {
  treat_pdu_transf_res ();
 }
      else
 call_error (36);
     }
  }
 /* TRATAMENTO DE RESPOSTA DE TRANFERÊNCIA DE PDU */

treat_pdu_transf_res ( )
  {
   Connection_id con_id_out;

   if (block_ctr->msg == pdu_transf_res)
     {
 /* Muda parametro de tipo de primitiva */
      switch (block->primitive)
 {
  case request : block->primitive = indication;
   break;
  case response_pos : block->primitive = confirm_pos;
   break;
  case response_neg : block->primitive = confirm_neg;
 };
 /* Preenche connection_id de saida */
      con_id_out = con_tab [block->connection_id];
      block->connection_id = con_id_out;
 /* Verifica insercao de erros */
      if (block_ctr->user_msg.transf_res.insert_error == true)
 insert_error (block_ctr->user_msg.transf_res.error_type);
      else  /* primitiva com PDU sem errors */
 {
  if (((block->data.pres_serv.service_name == p_release) &&
```

```
    (block->primitive == confirm_pos)) ||
     (block->data.pres_serv.service_name == p_u_abort))
       liberate_connection (block->connection_id);
 }
 /* Comuta primitiva */
      if (block->data.pres_serv.service_name != p_data)
          /* servico ACSE */
        envia_mens (mbo_acse,  block);
      else  /* servico MMS */
 envia_mens (mbo_mms, block);
     }
   else call_error (38);
   libera_mem (block_ctr);
 }
    /* ENVIO DE ABORTO DE APRESENTAÇÃO AO ACSE */

send_p_p_abort_ind (Connection_id connection)
  {
   int i;
   Block_struct *aux_ptr;
   pede_mem (100, &aux_ptr);
   aux_ptr->connection_id = connection;
   aux_ptr->primitive = indication;
   aux_ptr->service = presentation;
   aux_ptr->data_area_lenght  = 31;
   aux_ptr->data.pres_serv.service_name = p_p_abort;
   for (i = 0 ; i == 29; i++)  aux_ptr->data.pres_serv.prim_pres.
            p_p_abort_serv.provider_reason [i] = '0';
   envia_mens (mbo_acse, aux_ptr);
 }
    /* LIBERAÇÃO DA CONEXÃO */
liberate_connection (Connection_id connection)
  {
   Connection_id  i;
   i = con_tab [connection];
```

```
    con_tab [i] = not_used;
    con_tab [connection] = not_used;
  }
     /* INSERÇÃO DE ERROS NAS PDUS */

insert_error (Communication_errors error_type)
  {
   Mms_pdu_struct pdu_trans;
   switch (block->data.pres_serv.service_name)  /* seleciona PDU
                                                   do bloco armazenado */
     {
      case p_connection :
        if (error_type == invalid_pdu_error)
          block->data.pres_serv.prim_pres.p_connection_serv.
          p_connection_ind.valid_pdu = false;    /* PDU A_associate
                                                    invalida */
        else
          if ((error_type == pres_error) &&
                (block->primitive == confirm_pos))
            {
             block->primitive = confirm_neg;
             block->data.pres_serv.prim_pres.p_connection_serv.
             p_connection_cnf.result = provider_rejection;
            }
        break;
      case p_release :
        block->data.pres_serv.prim_pres.p_release_serv.p_release_ind.
          valid_pdu = false;    /* PDU A_release invalida */
        break;
      case p_u_abort :
        block->data.pres_serv.prim_pres.p_u_abort_serv.p_u_abort_ind.
          valid_pdu = false;    /* PDU A_abort invalida */
        break;
      case p_data :   /* Erro nas PDU's MMS */
        pdu_trans = block->data.pres_serv.prim_pres.p_data_serv.
          p_data_ind.select_ind.user_data;
```

```
switch (error_type)  /* Altera PDU conforme erro selecionado */
  {
   case service_error :
     switch (pdu_trans.pdu)
       {
        case confirmed_request :
          pdu_trans.mms_pdu.confirmed_request_pdu.
            service_name = block_ctr->user_msg.transf_res.
            error_par.new_conf_serv;
          break;
        case confirmed_response :
          pdu_trans.mms_pdu.confirmed_response_pdu.
            service_name = block_ctr->user_msg.transf_res.
            error_par.new_conf_serv;
          break;
        case unconfirmed_request :
          pdu_trans.mms_pdu.unconfirmed_request_pdu.
            unconf_request.service_name = block->
            user_msg.transf_res.error_par.new_unconf_serv;
          break;
        default : call_error (32);
       };
    break;
   case invoke_id_error :
     switch (pdu_trans.pdu)
       {
        case confirmed_request :
          pdu_trans.mms_pdu.confirmed_request_pdu.
            invoke_id = block_ctr->user_msg.transf_res.
             error_par.new_invoke_id;
          break;
        case confirmed_response :
          pdu_trans.mms_pdu.confirmed_response_pdu.
            invoke_id = block_ctr->user_msg.transf_res.
            error_par.new_invoke_id;
          break;
```

```
                case confirmed_error :
                  pdu_trans.mms_pdu.confirmed_error_pdu.
                    invoke_id = block_ctr->user_msg.transf_res.
                    error_par.new_invoke_id;
                  break;
                default : call_error (33);
               };
              break;
            case pdu_error :
              block->data.pres_serv.prim_pres.p_data_serv.p_data_ind.
                select_ind.invalid_pdu.reject_class = pdu_error;
                block->data.pres_serv.prim_pres.p_data_serv.p_data_ind.
                select_ind.invalid_pdu.reject_code= invalid_pdu;
              break;
            default : call_error (34);
           };
        block->data.pres_serv.prim_pres.p_data_serv.p_data_ind.
          select_ind.user_data = pdu_trans;
        break;
      default : call_error (35);
    }
  }
    /* NOTIFICA OCORRÊNCIA DE ERROS */

call_error (uint8 error_code)
  {
   bloqueia_escalonador ();
   printf (" Erro de execucao do programa numero %d",error_code);
   libera_escalonador ();
  }
```