UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA DA COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

"ASPECTOS DE ESPECIFICAÇÃO E IMPLEMENTAÇÃO DA INTERFACE DE APLICAÇÃO PARA O MMS: SERVIÇOS DE ACESSO A VARIÁVEIS"

por: **VERÔNICA LIMA PIMENTEL DE SOUSA**

orientador: **Prof. Dr. MANUEL DE JESUS MENDES**

Tese apresentada à Faculdade de Engenharia Elétrica - FEE - UNICAMP, como parte dos requisitos exigidos para a obtenção do título de MESTRE EM ENGENHARIA.

Julho de 1990

AGRADECIMENTOS E DEDICATÓRIA

Meus sinceros agradecimentos às entidades e às pessoas que, direta ou indiretamente, contribuíram para a realização desse trabalho:

À CAPES, pelo incentivo financeiro;

À UNICAMP, pelas condições propícias de trabalho;

Ao SEPROCE, pela concessão da licença de afastamento do cargo;

À Ana Maria Silva e Ademilde Félix, pelo trabalho de edição do texto, em CHIWRITER;

Ao Prof. Mendes, pela orientação e apoio ao longo do desenvolvimento do trabalho;

Ao colega Edmundo, pela dedicada co-orientação;

Ao grupo SISDI-MAP (Edmundo, Jayme, Durval, Cristina, Ivo, Antenor, Zé) pelos inúmeros contatos, pela dedicação e momentos de alegria que proporcionaram;

Aos colegas-amigos - Heraldo, Luiz (Pingo), Miguel "CIFEE", Armando, Marcélia, Paulo, Ana Maria - cujo apoio técnico ou moral foi fundamental não só no âmbito desse trabalho, mas para toda a vida;

E principalmente à minha família que, sem perceber o quanto, contribuiu fortemente com seu amor, tolerância e paciência; em especial ao meu marido Tel e a meus filhos (Estêvão, Érica, André e Nayara) a quem dedico esse trabalho, com muito carinho.

SUMÁRIO

Este trabalho apresenta um modelo de Interface de Aplicação proposto pela comissão do projeto MAP/TOP para redes MAP.

Descrevem-se os elementos funcionais estipulados pelo modelo, suas definições e requisitos de especificação.

Analisam-se as funções atribuídas à interface e seu papel no ambiente OSI.

Discute-se a implementação de seus elementos funcionais, definindo-se um subconjunto de suas funções para fins de implementação concreta e mostra-se a adaptação da implementação ao projeto SISDI-MAP, no qual a interface se integra como um processo num ambiente multi-tarefa. São apresentados e discutidos também os aspectos envolvidos na compatibilização da versão atual da interface com a nova versão do protocolo MMS (Draft 6) e mostram-se as restrições e considerações adotadas.

ABSTRACT

An Application Interface model proposed by a MAP/TOP project commission, to be used in MAP network is presented.

The functional elements of the interface, their definitions and specification requirements are described.

The interface functions are analyzed as well as its behavior in the OSI environment.

The interface functional blocks implementation is analyzed and, afterwards, a subset of its functions is defined in order to implementation. The adaptation to SISDI-MAP project, in which it works as an independent process, is presented and discussed.

All the aspects related to compatibility between current version of this interface and the new one of the MMS protocol specification (Draft 6) are also presented and discussed. Adopted constraints and assumptions are shown opportunely.

I N D I C E

LISTA DE FIGURAS iv

LISTA DE TABELAS v

TERMINOLOGIA E ABREVIATURAS vi

CAPITULO 1 - INTRODUÇÃO 1.1
1.1. Objetivo do Trabalho 1.2
1.2. O Projeto SISDI-MAP 1.3
1.3. Escopo do Trabalho 1.4

CAPITULO 2 - MODELO GERAL DA API 2.1
2.1. Objetivos da Padronização 2.2
2.2. O Modelo Conceitual da API 2.3
2.3. A Especificação da API 2.8
2.3.1. Chamada de Funções 2.8
2.3.2. Linguagem de Implementação 2.9
2.3.3. Suporte de Sistema Operacional 2.9
2.3.4. Suporte aos Modelos de Comunicação 2.10
2.4. As Funções da API 2.11
2.4.1. Funções Requisitantes e Respondedoras 2.11
2.4.2. Funções de Alto Nível e de Baixo Nível 2.11
2.4.3. Funções Livres de Contexto e Sensíveis ao Contexto .. 2.13
2.5. A Arquitetura da API 2.13
2.5.1. Região do Programa de Usuário 2.16
2.5.1.1 Módulo HLS 2.17
2.5.1.2 Módulo LLS 2.17
2.5.1.3 Módulo RS 2.17
2.5.2. Região do Provedor de Serviço 2.18
2.5.2.1 Módulo HLSP 2.19
2.5.2.2 Módulo CSP 2.20
2.5.2.3 Módulo PSP 2.21
2.5.2.4 Módulo IFA 2.21

2.6. Os Serviços da API 2.25
2.6.1. Os Serviços de Suporte 2.25
2.6.2. Os Serviços de Gerenciamento de Conexão 2.28
2.6.3. Os Serviços Específicos da Aplicação 2.32
2.7. Conclusões 2.32

CAPITULO 3 - ASPECTOS DE ESPECIFICAÇÃO DA API-MMS NO SISDI-MAP .. 3.1
3.1. O MMS 3.1
3.1.1. Modelo Abstrato do MMS 3.4
3.1.1.1 Interação de Processo Aplicativos..... 3.5
3.1.1.2 Dispositivo Virtual de Manufatura 3.6
3.1.1.3 Objetos, Atributos e Operações sobre objetos. 3.7
3.1.1.4 Descrição de Serviços e Requisitos de Conformidade 3.10
3.1.2. Os Serviços MMS 3.11
3.1.2.1 Os Serviços de Acesso a Variáveis 3.14
3.2. A API-MMS 3.16
3.2.1. O MMS como parte de uma AE..... 3.17
3.2.2. As estruturas das funções da API-MMS..... 3.18
3.2.3. Os tipos de dados da API-MMS..... 3.22
3.3. O SISDI-MAP 3.29
3.3.1. Objetivo do SISDI-MAP 3.29
3.3.2. Princípios de especificação do SISDI-MAP 3.30
3.3.3. Princípios de operação do SISDI-MAP 3.30
3.4. Conclusões 3.32

CAPITULO 4 - ASPECTOS DA IMPLEMENTAÇÃO DA API-MMS NO SISDI-MAP .. 4.1
4.1. Escopo de Implementação 4.2
4.1.1. Subconjunto de Serviços MMS 4.3
4.1.2. Funções Requisitantes x Funções Respondedoras 4.3
4.1.3. Procedimentos de Mapeamento dos Serviços Confirmados. 4.4
4.1.4. Tratamento de Indicação 4.4

4.2. A Implementação dos Serviços de Acesso a Variáveis 4.5
4.2.1. As Funções Auxiliares de Construção de Definições MMS 4.6
4.2.1.1 A Função New Restriction 4.7
4.2.1.2 A Função New Partial Access..... 4.8
4.2.1.3 A Função Make List 4.10
4.2.1.4 A Função New Component 4.13
4.2.1.5 A Função New Type 4.14
4.2.1.6 A Função New Variable Specfication 4.15
4.2.1.7 A Função New Scattered Access 4.16
4.2.1.8 A Função New Variable Access 4.17
4.2.1.9 A Função New Variable Association 4.20
4.2.1.10 A Função New Data 4.22
4.2.2. As Funções Auxiliares de Interpretação de Definições MMS 4.23
4.2.3. A Função de Leitura de Variável: READ VARIABLE 4.25
4.3. A Integração da API-MMS no SISDI-MAP..... 4.30
4.3.1. Comunicação e sincronização entre processos 4.31
4.3.2. Formatos de Mensagens 4.31
4.4. Conclusões 4.36

CAPITULO 5 - CONSIDERAÇÕES GERAIS, CONCLUSÕES FINAIS E SUGESTÕES 5.1
5.1. Padronização e Modelo OSI 5.1
5.2. Abrangência do Modelo da Interface 5.2
5.3. Desempenho AP/API e Aspectos Locais 5.2
5.4. Extensão das Funcionalidades da API 5.3
5.5. Compatibilização API-MMS 5.3
5.6. Interação da API no SISDI-MAP 5.4
5.7. Comunicação API-AP 5.4
5.8. O Teste de Implementação da API 5.5
5.9. Outras formas de projetar interfaces 5.5
5.10. Benefícios Finais da Interface 5.6

REFERÊNCIAS BIBLIOGRÁFICAS R.1

APÊNDICES
A: Definições dos tipos MMS para Serviços de Acesso a Variáveis, em C
B: Definições dos tipos API para Serviços de Acesso a Variáveis, em C

LISTA DE FIGURAS

Figura 1.1 - Interface: um elo essencial 1.2
Figura 1.2 - A API no ambiente SISDI-MAP 1.4
Figura 2.1 - O papel da API no modelo hierárquicoda ISO 2.1
Figura 2.2 - Generalidade, e modulariadade e portabilidade da API 2.4
Figura 2.3 - Invocações e associações múltiplas de AE 2.6
Figura 2.4 - Sistemas heterogêneos: capacidades modulares na API 2.7
Figura 2.5 - Um serviço de alto-nível da API e os serviços de aplicação correspondentes 2.12
Figura 2.6 - A API no ambiente OSI e no ambiente de sistemas reais 2.14
Figura 2.7 - A arquitetura da API: visão externa e visão interna 2.15
Figura 3.1 - Transformaçõa de dados entre AP, API e protocolo MMS 3.2
Figura 3.2 - Modelo da rede MAP e interação entre elementos 3.3
Figura 3.3 - Modelo simplificado de um VMD 3.7
Figura 3.4 - Comunicação entre processos no SISDI-MAP 3.32
Figura 4.1 - Exemplo de definições aninhadas requeridas no MMS 4.19
Figura 4.2 - Simetria das funções auxiliares da API-MMS 4.23
Figura 4.3 - Diagrama temporal de uma operação de leitura de variável 4.30
Figura 4.4 - Formato de mensagens: mensagem API-AP e mensagem API-MMS 4.32

LISTA DE TABELAS

Tabela 2.1 Procedimentos da API no desempenho de suas funções 2.4

Tabela 3.1 Classes dos objetos MMS e seus escopos de operação 3.9

Tabela 3.2 Os serviços MMS 3.14

Tabela 3.3 As funções da API-MMS 3.21

Tabela 3.4 Tipos primitivos da API-MMS 3.23

Tabela 3.5 Relação dos tipos MMS dos Serviços de Acesso a Variáveis, nome do parâmetro MMS correspondente e funções auxiliares que os utilizam 3.29

TERMINOLOGIAS E ABREVIATURAS

AA	Application Association
ACSE	Association Control Service Element
AE	Application Entity
AP	Application Process
ANSI	American National Standards Institute
API	Application Program Interface
ASE	Application Service Element
ASN.1	Abstract Syntax Notation One
CASE	Common Application Service Element
CCR	Commitment Concurrency and Recovery
CEP	Connection End Point
CIM	Computer Integration Manufacturing
DS	Directory Services
FTAM	File Transfer Access and Management
ISO	International Standards Organization
MAP	Manufacturing Automation Protocol
MAP/TOP	Manufacturing Automation Protocol/Technical Office Protocol
MMS	Manufacturing Message Specification
OSI	Open System Interconnection
OSIE	Open System Interconnection Environment
PCI	Protocol Control Information
PDU	Protocol Data Unit
PM	Protocol Machine
PSAP	Presentation Service Access Point
RDA	Remote Data Access
SASE	Specific Application Service Element
SAP	Service Access Point
SDU	Service Data Unit
SISDI-MAP	Sistema Didático dos Protocolos MAP
VMD	Virtual Machine Device

CAPÍTULO 1

INTRODUÇÃO

Num ambiente industrial existem dispositivos de diversos níveis de complexidade e com diversos aspectos de funcionalidade. Os programas aplicativos que operam esses dispositivos são, da mesma forma, de níveis de complexidade diversos e atingem áreas de operacionalidade de abrangências diferentes.

É fundamental a flexibilidade no crescimento da automação industrial, tanto no aspecto de introdução ou substituição de novos dispositivos, como na incorporação dos elementos já existentes ao sistema de monitoração automática, integrada e aberta a novas integrações, objetivando sempre a migração gradativa de um ambiente fabril convencional para um ambiente aberto integrado - CIM.

O esforço da ISO neste aspecto tem sido o de adotar procedimentos e padrões que permitam essa migração [11]. Quanto a diversidade dos dispositivos, a introdução do novo conceito de "dispositivo virtual" - (VMD) amenisa a problemática dos software aplicativos com tendência de restringir sua implementação a um dispositivo específico. O dispositivo virtual tem por objetivo a desvinculação do hardware e descreve um comportamento visível externamente a nível de aplicação. A implementação do VMD num programa aplicativo consiste no mapeamento desse comportamento abstrato nas funções do equipamento real de manufatura. O papel da interface, por sua vez, é o de generalizar o uso de softwares existentes em ambientes de aplicação diferentes. Ela forma um elo entre um programa aplicativo (AP) anteriormente dedicado ao hardware e/ou dependente do Sistema Operacional (SO) - e um sistema aberto e genérico de comunicação, como mostra a figura 1.1.

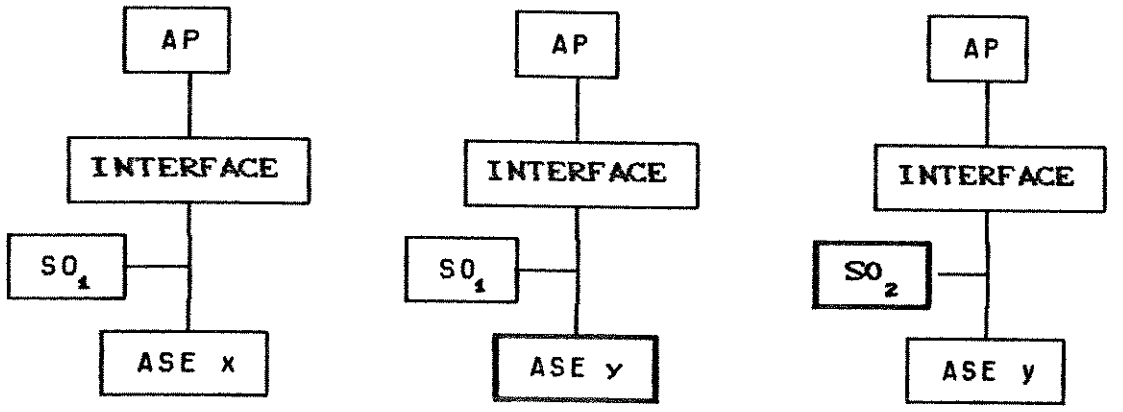


Figura 1.1 - Interface: um elo essencial.

Para atingir essa generalidade, uma interface deve descrever um comportamento externo bastante amplo e ser capaz de mapear esse comportamento nas funções de serviços reais do ambiente de aplicação.

O esforço em se estabelecer um modelo, uma semântica precisa de formatos de chamadas e procedimentos internos claros e gerais, permite definir um suporte único sobre o qual o ambiente de aplicação possa ser descrito de forma consistente.

Isso significa que a introdução de uma interface de aplicação padronizada para os sistemas abertos de comunicação guiará também os programas aplicativos a uma certa padronização e, como resultado, obtém-se as vantagens que toda padronização traz a um sistema em seus vários aspectos, como por exemplo: portabilidade de software aplicativo; transportabilidade do programador (facilidade de se transferir de sistema a sistema); independência tanto dos dispositivos como do sistema operacional residente.

1.1 - OBJETIVO DO TRABALHO

A motivação para o desenvolvimento desse trabalho encontra-se no estudo, interpretação e compreensão do modelo e da especificação da Interface de Programa de Aplicação - API, oriunda do projeto MAP/TOP [M1] e sua implementação e adequação a um sistema didático (SISDI-MAP), capaz de prover facilidades a um usuário, acadêmico ou industrial, de modo que ele observe e compreenda o

funcionamento de uma rede de comunicação de dados num ambiente de manufatura.

Sendo o modelo e especificação da interface genéricos e abrangentes a vários protocolos de aplicação (FTAM, MMS, e Private Communication), dirige-se nesse trabalho à especificação da interface para o protocolo MMS. Dentre os diversos serviços oferecidos pelo MMS (gerenciamento de contexto, suporte a VMD, gerenciamentos de domínio, de invocação remota de programas, de eventos, de semáforos, de acesso a variáveis, de jornal e de arquivos,) apenas alguns são implementados numa estação real do ambiente de manufatura. Utilizam-se subconjuntos, com as opções e variantes disponíveis em cada serviço, de acordo com o tipo de aplicação. Normalmente, os serviços de gerenciamento de contexto, responsáveis pelo estabelecimento e término das associações de aplicação, estão sempre presentes em qualquer aplicação.

Para compor a API no SISDI-MAP, foram selecionados os "serviços de gerenciamento de contexto", que, adicionado de outros aspectos envolvidos em programas aplicativos - AP (Application Program), é objeto de tese de J.N.C. [T1], e os "serviços de acesso à variáveis", objeto dessa tese.

1.2 - O PROJETO SISDI-MAP

O SISDI-MAP é um sistema multi-tarefa desenvolvido pela integração de vários trabalhos de tese de mestrado [T1], [T2], [T3], [T4], [T5] e [T6] e de doutorado [T7] com o objetivo de emular aplicações reais, típicas de um ambiente de manufatura, através da execução de vários AP's que se comunicam entre si, mostrando ao operador do sistema as transformações dos dados na comunicação fabril.

O Sistema Didático SISDI-MAP abrange atualmente o protocolo de aplicação MMS, o protocolo ACSE, os programas aplicativos do usuário, implementados segundo a especificação do modelo OSI e simula o protocolo de Apresentação (nível 6), cuja implementação futuramente lhe será incorporada . O sistema conta também com uma interface de operação a qual interage continuamente com o operador do sistema, colocando-o frente ao comportamento interno dos protocolos acima

citados e refletindo a ele toda a interação entre protocolos, interface de aplicação e programas aplicativos de usuário. Assim, o sistema permite ao seu usuário o estudo e a compreensão do funcionamento da comunicação fabril [A1]. Todos esses softwares (protocolos, APs, API e Interface de Operação) estão sobre o suporte de um Núcleo de Tempo Real, que, através de suas funções , fornece ao ambiente de comunicação a operacionalidade necessária de comunicação entre os processos. A figura 1.2 ilustra, numa forma geral, o ambiente SISDI-MAP.

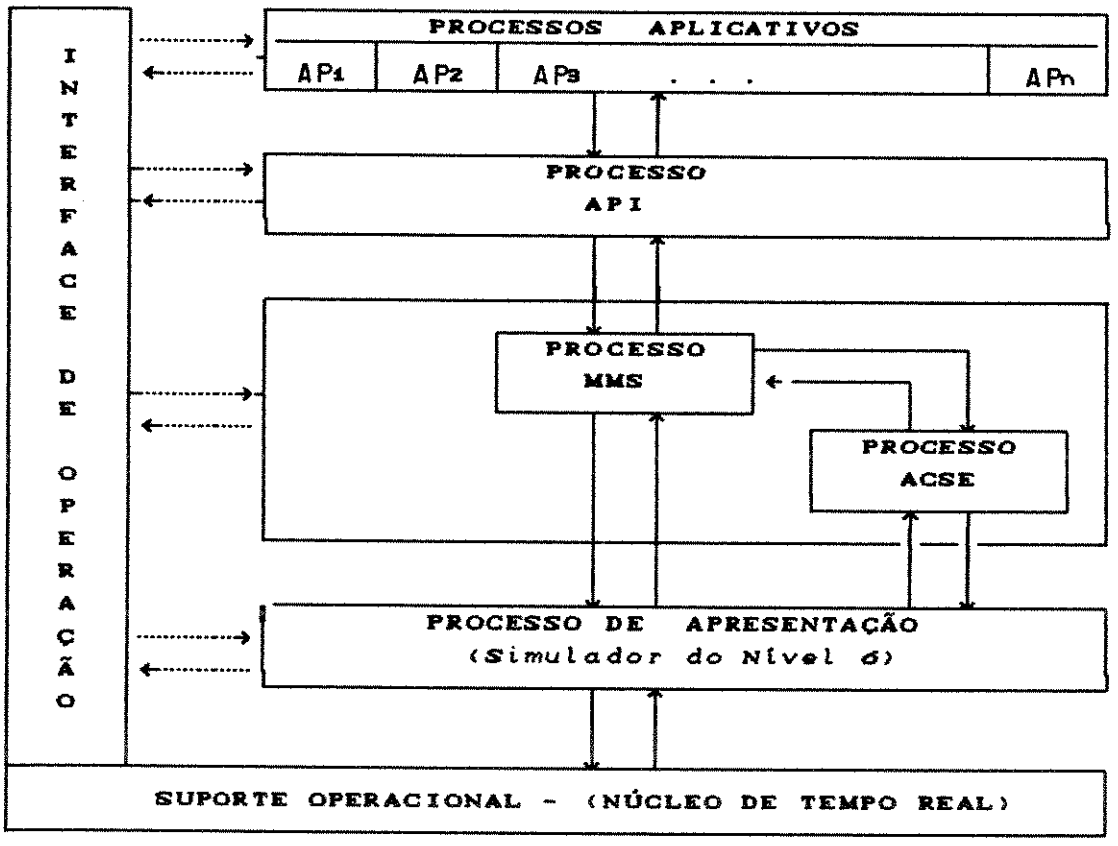


Figura 1.2 - A API no ambiente SISDI-MAP

1.3 - ESCOPO DO TRABALHO

Pretende-se ao longo dessa dissertação introduzir o conceito de Interface de Programa de Aplicação, e apresentar o modelo geral de uma interface genérica, apresentado pela GM, no projeto MAP, descrevendo sua arquitetura funcional e os requisitos gerais de

especificação. Em seguida, descreve-se o modelo abstrato do protocolo MMS, que se caracteriza pelas interações Cliente/Servidor e troca de serviços entre ambos. Mostram-seas particularidades da interface para esta aplicação específica (MMS) e os aspectos de implementação envolvidos na prática de implementar alguns serviços da interface para atender ao conjunto de serviços MMS selecionado. A implementação da API visa o sistema didático para o qual ela foi adaptada, incorporando-lhe os requisitos de um sistema multitarefa, no qual ela compreende um processo, como foi mostrado na figura 1.2.

Descreve-se o trabalho em 5 capítulos, sendo o Capítulo 1 esta introdução. Ao final de cada um dos capítulos seguintes, faz-se um pequeno resumo.

No Capítulo 2 apresenta-se o modelo geral da API e descrevem-se os requisitos gerais de especificação. Analisam-se os diversos tipos de funções que ela desempenha, funções essas que se diferenciam sob certos aspectos. Classificam-se os serviços executados pela interface nos seus diferentes papéis. Esses serviços são de suporte à comunicação, de gerenciamento de conexão e os serviços específicos da aplicação, no caso, MMS.

No Capítulo 3 especifica-se a Interface de Aplicação MMS, descrevendo seus diferentes tipos de função e delineando seus tipos de dados e seus procedimentos à luz do modelo geral da API. Devido ao documento da API que se dispunha referir-se aos serviços MMS da versão anterior - DRAFT 5, foi necessário um esforço no sentido de compatibilizar a API com a especificação do MMS na versão atual , DRAFT 6, já que foi para esta versão que o protocolo MMS foi implementado no SISDI-MAP.

No Capítulo 4 apresentam-se todos os aspectos relevantes da implementação da interface. O Sistema Didático (SISDI-MAP) é descrito com mais detalhes e são discutidos seus princípios de funcionamento. Observa-se o enquadramento da interface MMS no Sistema e sua interação com os outros processos.

O Capítulo 5 consta de algumas considerações gerais e de conclusões sobre o trabalho. Aqui chega-se a um fechamento de diversos aspectos de experiência em implementação de interface e abrem-se perspectivas de novas pesquisas nesse pequeno setor de automação industrial.

CAPÍTULO 2

O MODELO DA API

Um esforço em padronização em sistemas abertos está na especificação de interfaces, pois são elas que geralmente mudam de sistema para sistema, de ambiente computacional para ambiente computacional. Por exemplo, num elemento de aplicação (ASE), as entidades de aplicação são implementadas em equipamentos diferentes e sob sistemas operacionais diferentes. Por outro lado, um software aplicativo pode, em tempos diferentes ou até mesmo dentro de um certo intervalo de tempo (considerando-se uma instância), fazer uso de sistemas de comunicação distintos.

Uma interface de aplicação é uma fronteira entre dois subsistemas adjacentes num **modelo hierárquico**. Os processos aplicativos são os próprios usuários dos serviços da camada de aplicação no ambiente OSI [L2]. (figura 2.1.).

Figura 2.1 - O papel da interface de aplicação no modelo hierárquico da ISO

A utilização de uma interface visa deixar o programa de aplicação (AP) independente do sistema de comunicação e do sistema operacional.

O programa de aplicação fica, portanto, mais genérico e portátil para diferentes sistemas de comunicação, e menos específico do hardware, ou seja, mais genérico e portátil para diferentes equipamentos e/ou ambiente operacional.

2.1. OBJETIVOS DA PADRONIZACAO

Para que sistemas diferentes se comuniquem entre si é necesssário a padronização de interfaces. A padronização da interface deve atingir os seguintes objetivos:

. Torná-la independente dos dispositivos de uma configuração

A mesma interface poderá ser utilizada em qualquer dispositivo. Sua padronização garante fácil programação para dispositivos diferentes.

. Torná-la independente da linguagem que a implementa

Um Programador precisa de conhecimentos mínimos dos sistemas (aplicação e dispositivos) em questão, para implementar a interface na linguagem preferida.

. Torná-la independente da aplicação que a utiliza

A mesma aplicação de uma determinada instalação poderá ser transferida para outra com modificações mínimas.

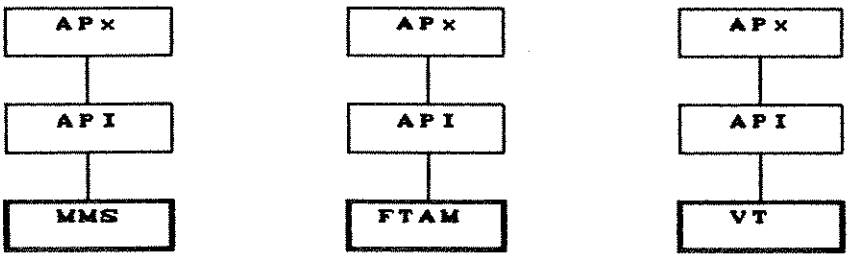
2.2. O MODELO CONCEITUAL DA API

O fato de haver num ambiente industrial dispositivos de diversos níveis de complexidade, significa que a operacionalidade de um dispositivo abrange mais ou abrange menos funções do conjunto de serviços suportados pela aplicação. A ISO previu essa heterogeneidade no ambiente operacional distribuído e definiu conceitos que refletem a realidade estática e dinâmica do ambiente [11].

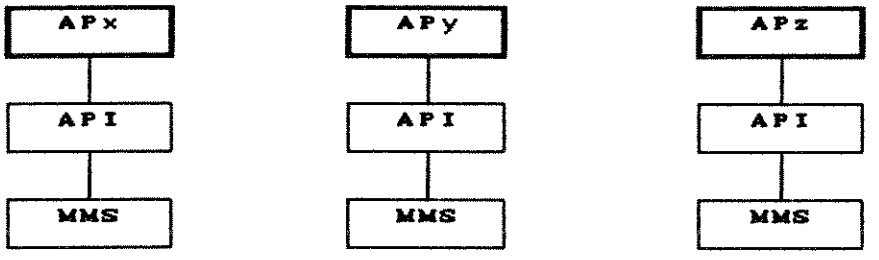
A seleção de um subconjunto de "atividades" que atendam as necessidades operacionais de um dispositivo qualquer, seja um robô, um CLP, ou um dispositivo gerenciador (PC, por exemplo), é levada em conta na implementação das entidades de aplicação de um programa aplicativo num dispositivo específico. Essas entidades são definidas dinamicamente dentro de uma instância do programa aplicativo e uma vez ativas, representam o conjunto de funções ou uma atividade que uma certa tarefa de aplicação deve desempenhar. Para dar suporte a atividades tão heterogêneas, o suporte operacional ou S.O. deve acompanhar essa capacidade funcional.

O modelo da API segue, do mesmo modo, a **heterogeneidade** e a **complexidade** desse ambiente. A API tem características que refletem o contexto onde ela for utilizada.

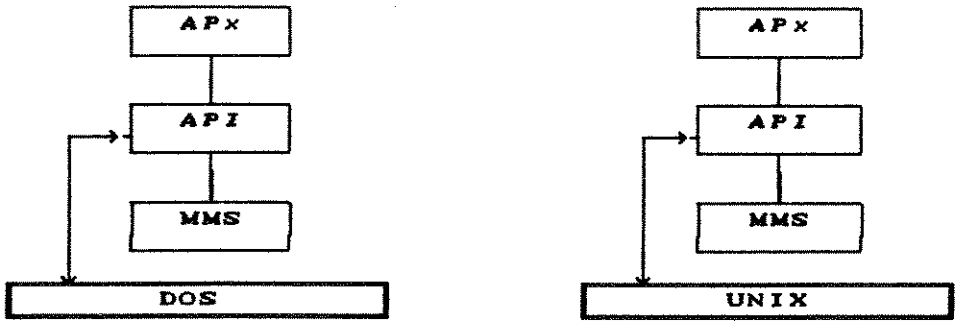
Na figura 2.2 mostra-se um esquema das diversidades de "situações" e a capacidade da API de adaptar-se a cada uma delas. Graças a sua padronização, ela é flexível.



a) Generalidade da API
A mesma API sendo utilizada em sistemas de comunicação distintos,ou seja, atendendo ao mesmo programa aplicativo que contem elementos de serviços de comunicação diferentes



b) Modularidade da API
A mesma API sendo configurada para Programas Aplicativos diferentes, os quais utilizam o mesmo Sistema de Comunicação



c) Portabilidade da API
A mesma API sendo utilizada com suportes operacionais diferentes

Figura 2.2 - Generalidade, Modularidade e Portabilidade da API

Generalidade da API

A característica de generalidade da API diz respeito ao modelo geral da API e à sua capacidade de operar com **múltiplas invocações de AE**. Significa que se um AP, em qualquer instante de sua execução, deseja invocar associações com AE's remotos, a API deve permitir que as associações requeridas pelo AE's, ora se refiram a um determinado ASE, ora a outro. Isso é satisfeito durante o estabelecimento de associação, quando, através da chamada de função CONNECT da API, o AP (instância do AP) faz uso de um parâmetro existente dentro do Bloco de Controle de Dados, chamado ASE_SPECIFIC_ASSO_REQ_INFORMATION, no qual são passadas informações específicas do ASE do qual ele deseja obter serviços de comunicação.

Porém, mesmo naquela "instância" do AP (pode ser noutra) uma nova AE pode ser invocada ou ativada, a qual estabelece serviço de um outro ASE diferente daquele existente na associação anterior: é necessário permitir total interação e a execução de várias atividades distintas para alcançar uma atividade com objetivo global.

A API pode gerenciar todas essas invocações de AE's e suas múltiplas associações (figura 2.3).

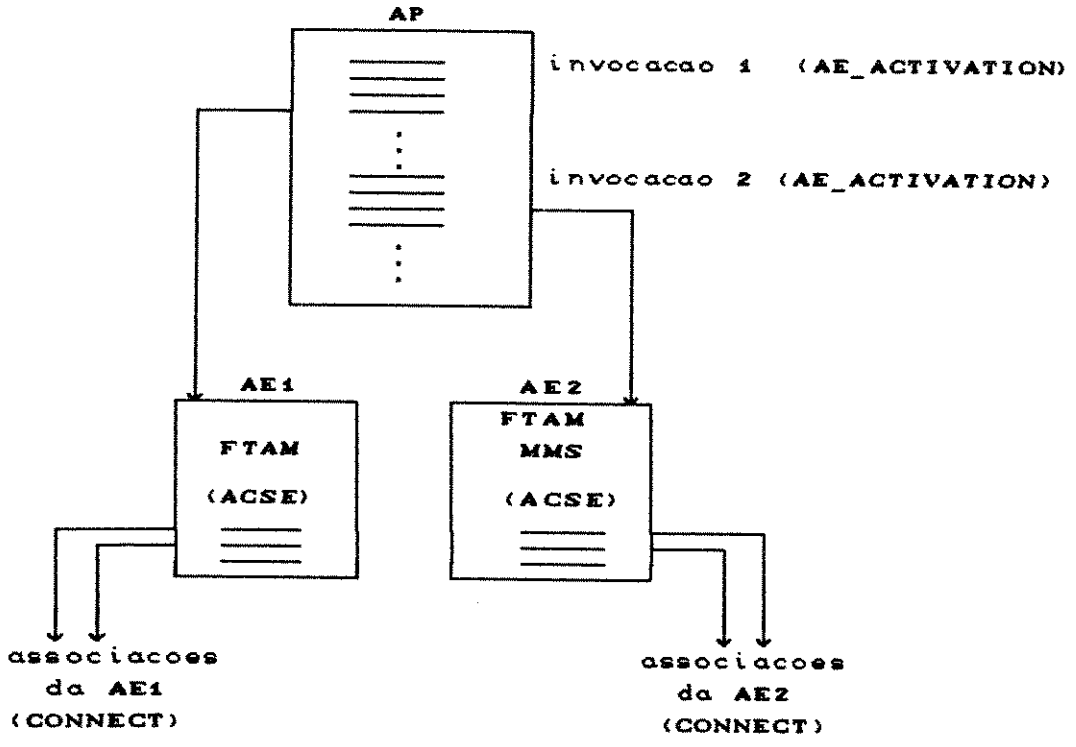


Figura 2.3 - Invocações e Associações múltiplas de AE.

Outras capacidades específicas da API visando atender a sua característica de generalidade estão descritas e especificadas no documento [M2], onde se poderá obter maiores detalhes.

MODULARIDADE DA API

Como veremos na seção 2.5, o modelo arquitetural da API contêm uma biblioteca de funções que poderão ser chamadas pelo usuário - AP (programa existente no dispositivo que o torna operacional).

o Modelo RF OSI/ISO estabelece que a interação entre AP's só é possível se se pre-estabelecer previamente um universo de discurso comum. Um subconjunto de um universo de discurso é determinado e esse subconjunto é então compartilhado. Daí surgiram os modelos virtuais que são refletidos no sistema de Aplicação e que tornam a

interação factível entre os AP's.

É através das chamadas à biblioteca de funções da API que um AP inicia sua comunicação com outro, ou outros AP's remotos. (figura 2.4). É nela que será revelado o **universo de discurso** comum a ambos os AP's comunicantes. A API é flexível o bastante para refletir essa adequação ao universo de discurso. A interface é implementada de forma que melhor se adeque às necessidades, evitando penalidades de espaço/tempo com a inclusão de capacidades adicionais desnecessárias.

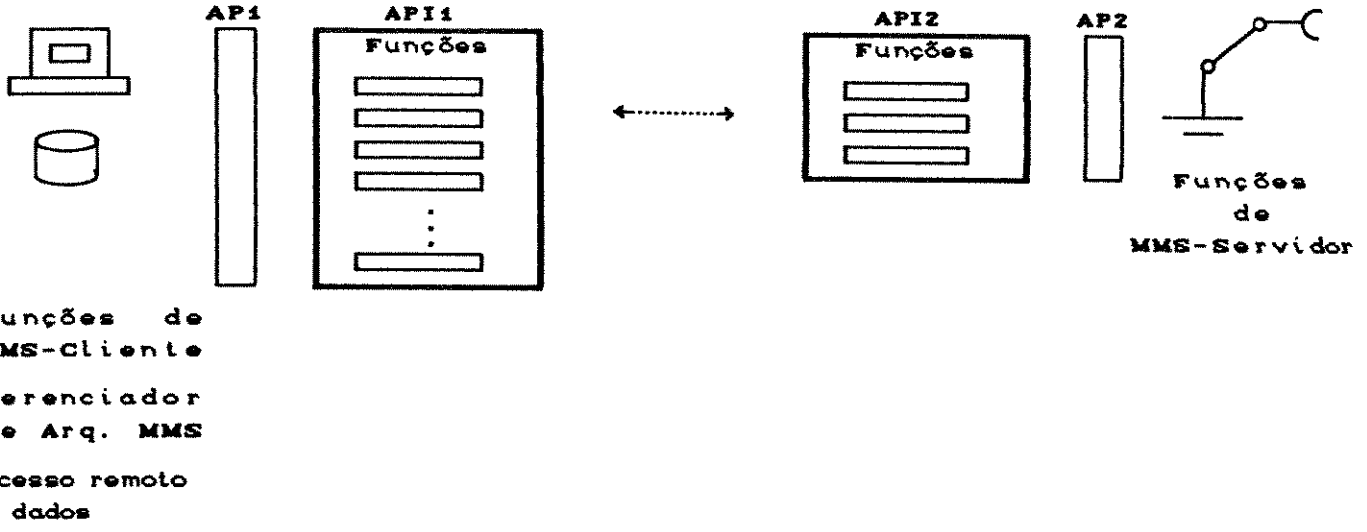


Figura 2.4 - Sistemas heterogêneos: capacidades moduláveis na API

Portabilidade da API

Tem acontecido recentemente um grande esforço em compatibilizar softwares aos hardwares existentes. A tendência será de diminuir o número de softwares dedicados a um simples ambiente de hardware. Os sistemas operacionais já contribuem significantemente para a portabilidade de aplicativos. Nessa mesma direção, a API foi especificada para acrescentar sua contribuição à tão desejada portabilidade. Ela conta com algumas **funções de suporte** (serão descritos na seção 2.6), que isolam mais ainda o AP do hardware e/ou Sistema Operacional. Esses serviços dizem respeito ao sincronismo de chamadas, a alocação de áreas de armazenamento em memória e outros.

2.3. A ESPECIFICAÇÃO DA API

Tendo tais pontos como fundamentos da padronização da API, foi utilizada uma metodologia que pudesse ser usada para descrever uma API genérica, não ambígua e testável e que pudesse ser implementada numa grande variedade de sistemas computacionais. Essa metodologia consiste de uma especificação de formatos de chamadas gerais à API, um modelo de interação entre API e os serviços de comunicação e um conjunto de utilidades necessárias a essa interação e que suporte o **contexto** e a **semântica** definidas no modelo.

Detalhes sobre os conceitos básicos requeridos na especificação do modelo geral da API não serão aqui enfatizados. Para obter maiores detalhes, deverá ser consultado o documento GM-MAP/TOP [M2].

Nesse item, mencionam-se brevemente os requisitos essenciais à especificação geral da API, com a intensão de fazer o leitor entender os objetivos que devem ser alcançados. Nos itens posteriores esplana-se a arquitetura da API, sua funcionalidade sob diversos aspectos e descrevem-se os serviços por ela prestados.

Ao longo da citação dos requisitos, faz-se menção também aos objetivos adicionais que se fizeram necessários para integrar a API ao SISDI-MAP (por exemplo, formato da mensagem entre os processos comunicantes).

2.3.1. CHAMADA DE FUNÇÕES

As chamadas às funções da API têm um formato padronizado, seguindo as regras de especificação da interface. As chamadas de funções ativam os serviços prestados pela interface ao usuário. Nas chamadas são passados os parâmetros de entrada/saída do usuário. Na adaptação da interface ao SISDI-MAP as chamadas se dão através de mensagens com padrão unificado cuja estrutura será mostradaa no Capítulo 3.

2.3.2. LINGUAGEM DE IMPLEMENTAÇÃO

A especificação da API desvincula-se da ferramenta de implementação, em particular, de uma linguagem específica em programação. Porém, está dirigida a linguagens que permitam associar diferentes estruturas de dados ao mesmo espaço de memória, ou seja, dados de tipos diferentes possam ser representados num mesmo espaço de armazenamento. Como ilustração, pode-se citar, por meio de um exemplo simples, como isso é feito na linguagem C.

```
typedef union {
                int             a;
                char            *b;
                float           c;
                struct tipo-y   d;
              } Tipo_var;

Tipo_var                        x;
```

A variável x é do tipo Tipo_var e, ora terá um valor do tipo inteiro, ora do tipo string de caracteres, ora do tipo float e poderá mesmo ser de um tipo estruturado mais complexo, ocupando o mesmo espaço reservado para a mesma.

"Uniões" fornecem um modo de se manipular tipos diferentes de dados numa única área de armazenamento, sem a necessidade de se manter qualquer informação dependente da máquina no programa [F1].

No capítulo 4 constata-se a utilidade dessa restrição presente na especificação da API.

2.3.3. SUPORTE DE SISTEMA OPERACIONAL

A especificação da interface é neutra quanto ao sistema operacional residente, a menos de que este deve prover um ambiente

multi-tarefa e suporte à comunicação entre processos, pois a API provê a habilidade de se desempenhar operações síncronas e assíncronas.

Operações síncronas são aquelas em que o usuário envia seu pedido de serviço e espera o resultado para, então, continuar executando sua tarefa.

Nas operações assíncronas o usuário é sinalizado pela API do envio de seu pedido ao provedor de serviço de rede e continua imediatamente a executar sua tarefa em paralelo com a tarefa do provedor. Ao término do serviço, a API notifica a tarefa do usuário e, a partir daí, o usuário pode dispor dos resultados obtidos.

Um mecanismo de gerenciamento de eventos, é previsto na especificação da API de forma a padronizar o gerenciamento de operações assíncronas.

Os serviços da API que suportam as operações assíncronas são WAIT e NOTE, descritos mais adiante, na Seção 2.6.1 - "Os Serviços de Suporte".

No SISDI-MAP o núcleo de tempo real provê o ambiente multi-tarefa que suporta a coordenação e a comunicação inter-processos [T2].

2.3.4. SUPORTE AOS MODELOS DE COMUNICAÇÃO

A ISO adotou serviços de dois tipos diferentes quanto ao modelo de comunicação entre as entidades pares. São eles: serviços orientados a conexão (connection oriented) e serviços não-orientados a conexão (connectionless). A interface de programa de aplicação deve refletir o modelo adotado no sistema de comunicação, ou seja, a API deve dar suporte aos dois modelos de comunicação.

A necessidade de se adotar um ou outro modelo vai depender do tipo de aplicação e especialmente da qualidade do serviço: maior controle do usuário final, **compromisso eficiência x overhead** e outros fatores abordados na filosofia de sistemas distribuídos [L3].

O Protocolo MMS, em particular, especificado para aplicações típicas de tempo-real, requer segurança e confiabilidade nos dados. Este protocolo suporta a comunicação orientada a conexão.

2.4. AS FUNÇÕES DA API

Como Já introduzimos no Capítulo 1, as funções da API diferem-se sob diversos aspectos de seus papéis. Elas podem ser classificadas sob esquemas diferentes e cada classificação é independente das outras.

2.4.1. FUNÇÕES REQUISITANTES E FUNÇÕES RESPONDEDORES

No Modelo de Referência da ISO tem-se esses conceitos bem definidos. Os modos de operação (requisitante ou respondedor) do AP, implica na API também no seu modo de operar. A API deve distinguir uma função requisitante de uma função respondedora e executar os procedimentos adequadamente.

Uma função requisitante pode acionar tipos de ações diferentes na API, por exemplo, quando se referem aos serviços confirmados. A identificação daquela requisição deve ser mantida internamente na API pois uma resposta é esperada da aplicação correspondente. Já para os serviços não-confirmados, a API simplesmente mapeia a requisição nas primitivas de REQUEST do provedor de serviços do protocolo de aplicação.

As funções respondedoras são as funções que correspondem às ações finais em resposta às requisições enviadas pela aplicação-par. Na API essas funções podem estar implementadas em duas formas diferentes: num mecanismo que automaticamente responda à função do modo requisitante da aplicação par e/ou pode contar com funções que permitam ao usuário responder sob o contexto da aplicação. Veremos, na seção 2.5 os procedimentos necessários para realizar esses mecanismos nos módulos funcionais da API.

2.4.2. FUNÇÕES DE ALTO NÍVEL E DE BAIXO NÍVEL

A API desempenha diferenciadamente procedimentos de acordo com o tipo de serviço solicitado. Alguns dos serviços que ela pode prover ao AP (ex. transferência de arquivo - FILE GET) são

"quebrados" em vários serviços da camada de aplicação:

- Abertura do arquivo - FILE OPEN, que é um serviço confirmado,
- Leitura do arquivo - FILE READ, que é um serviço confirmado e
- Fechamento do arquivo - FILE CLOSE, que é um serviço confirmado

A figura 2.5 ilustra a sequência de primitivas geradas a partir de uma solicitação de serviço de alto nível da API feita pelo processo aplicativo.

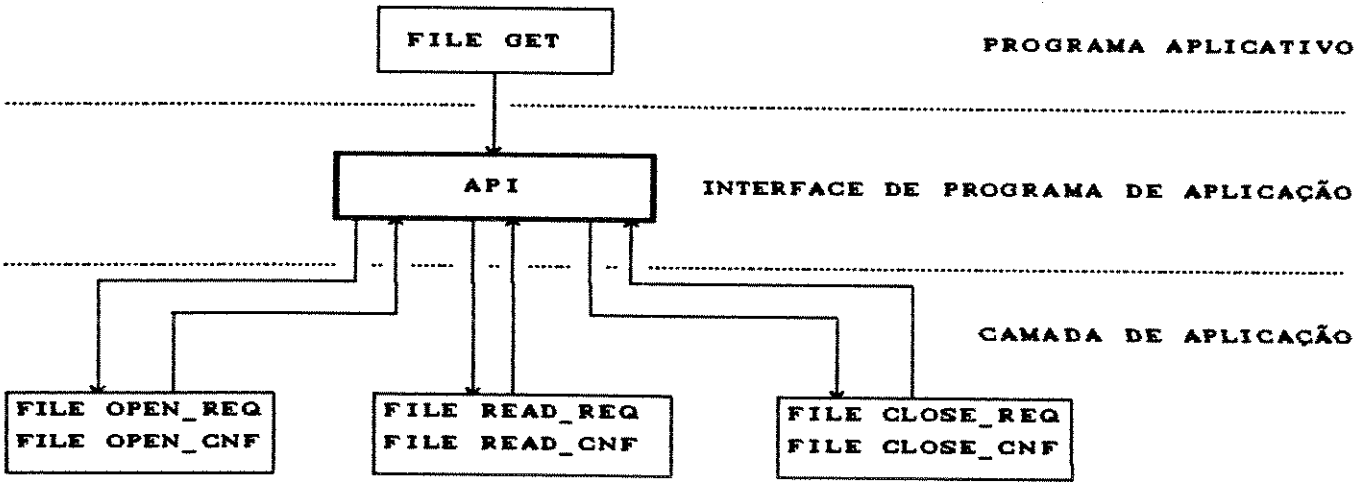


Figura 2.5 - Um serviço de alto nível da API e os serviços da aplicação correspondentes

Uma função típica de serviço de alto nível faz uso de várias funções de serviço de baixo nível para completar sua tarefa. São alocados, a nível da API, recursos mais abrangentes que possibilitem a complexidade do serviço: espaço de memória, disponibilidade dos recursos internos, etc.

As funções de baixo nível, por sua vez, fazem uso direto das primitivas de serviço fornecidas pelo provedor dos serviços de aplicação. Nesse caso, as máquinas de protocolo do nível de aplicação e da API são semelhantes; existe uma correspondência direta entre os serviços providos pelo protocolo de aplicação e os serviços

solicitadas à API pelo usuário.

2.4.3. FUNÇÕES LIVRES DE CONTEXTO E SENSÍVEIS AO CONTEXTO

Num modelo de comunicação orientado a conexão, os serviços devem enquadrar-se dentro dos parâmetros de configuração, e dentro dos padrões de qualidade negociados. Estabelece-se assim um contexto dentro do qual ocorrerá a comunicação.

Se um contexto é estabelecido, certas funções dependerão do contexto previamente estabelecido, enquanto o canal de comunicação entre as entidades de aplicação for mantido, até se completar a transação. Estas funções, que devem ser executadas dentro de uma associação particular, são funções sensíveis ao contexto.

Por outro lado, as funções livres de contexto são funções "stand alone" o que não significa que essas funções só ocorram fora de uma associação, mas que não serão influenciadas pelo contexto estabelecido numa associação particular entre dois AP's.

A título de exemplo, as funções de apoio aos serviços MMS providos pela API (funções auxiliares) são todas livres de contexto. As demais (funções principais), são sensíveis ao contexto.

2.5. A ARQUITETURA DA API

O problema da interconectabilidade universal ou interconexão de sistemas abertos foi amplamente analisado e discutido com o propósito de atingir padrões sobre os quais qualquer processo de aplicação do ambiente de sistemas reais pudesse comunicar-se com outro processo residente noutro sistema real. Na tentativa de minimizar as implicações provocadas pela heterogeneidade dos sistemas reais, a ISO formulou um modelo que serviria de base comum e que envolvem estruturas e atividades padronizadas. Através desse **ambiente padronizado** (ambiente OSIE) uma "linguagem" comum pode ser utilizada pelos diversos sistemas reais.

Dentro desse modelo foi também projetada a API. Através da interface e da estruturação do software de comunicação, toda a trans-

parência de dados e dos serviços de comunicação são alcançados para suportar uma grande variedade de aplicações. A figura 2.6 ilustra como se enquadra a API dentro dos ambientes operacionais envolvidos num sistema de comunicação [L4].

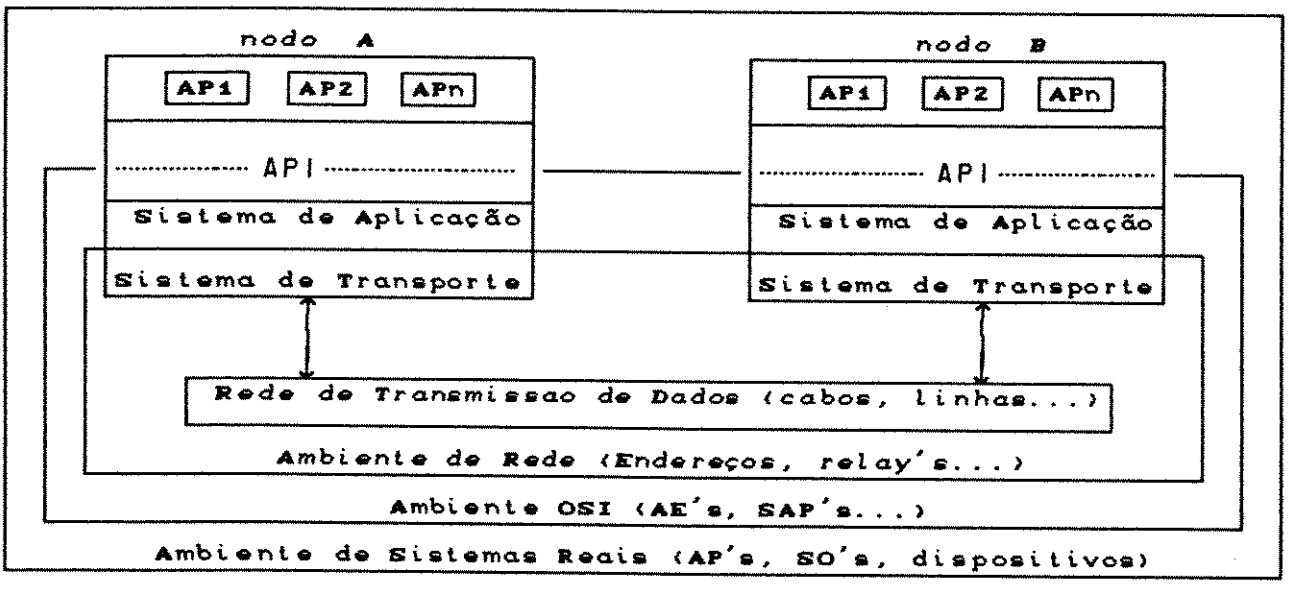
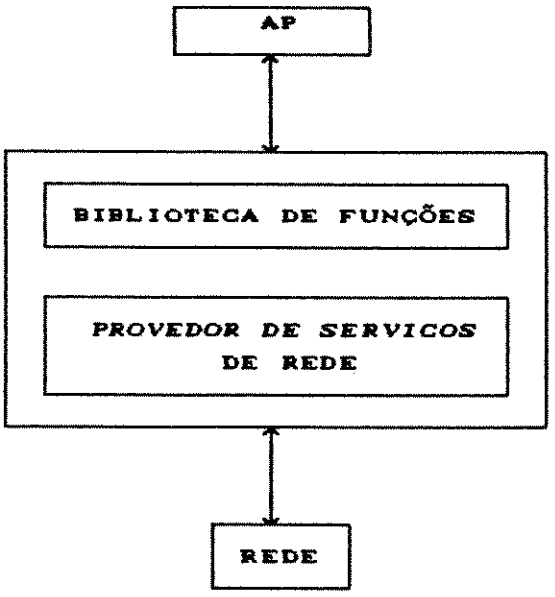
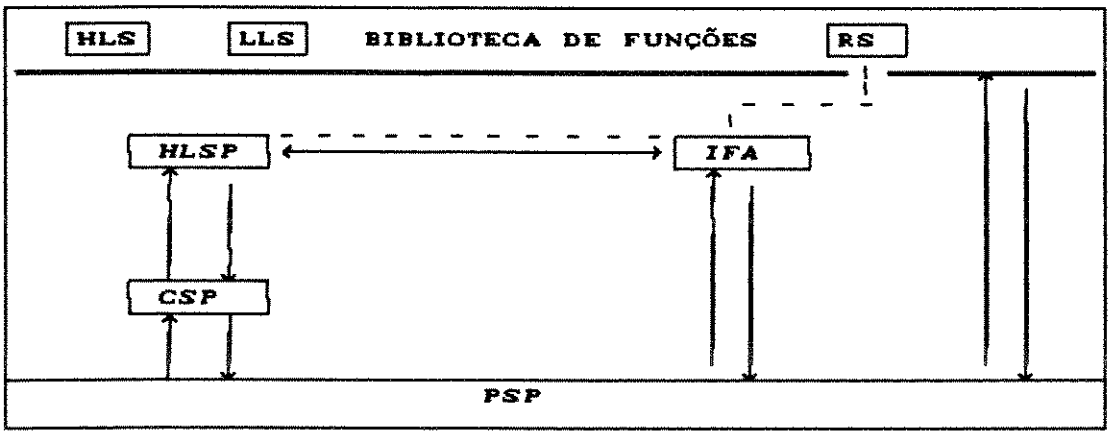


Figura 2.6 - A API no ambiente OSI e no ambiente de sistemas reais

A arquitetura funcional da API reflete o papel de elo entre o ambiente real do usuário e um ambiente padronizado de rede. Ela pode ser esquematizada dentro de duas grandes regiões: uma pertence ao ambiente do usuário e a outra é o próprio provedor de serviços de rede. A figura 2.7a tem-se a estrutura da API vista do lado externo e a figura 2.7b mostra os módulos funcionais componentes de cada uma das regiões e as interações inter-módulos, através de fluxos de dados e de controle.



a) Visão externa da API



b) Visão interna da API

Figura 2.5- Arquitetura da API: a) visão externa
b) visão interna

Esses módulos funcionais são usados no modelo da API para descrever a funcionalidade que ocorre ao se prover os serviços para um usuário (AP).

A interação entre os módulos é determinada pelo tipo de serviço solicitado.

As funções desempenhadas internamente pela API dizem respeito à seleção desses módulos funcionais e a interação necessária inter-módulos. Pode-se verificar que, dependendo do tipo de função a desempenhar (veja classificação das funções da API, na seção anterior-2.4), alguns módulos são selecionados enquanto outros ficam "inativos".

Estes blocos podem ser implementados como tarefas separadas com o intuito de prover um ambiente interno multitarefa. Para o Sistema Didático porém, como se verificará mais adiante, a API foi implementada como uma única tarefa, na qual os módulos funcionais são procedimentos ou "funções" (como são chamados na linguagem C), onde a interação é feita por passagem de parâmetros e o retorno indicado pelo valor retornado da função. [F1]

A seguir, descreve-se cada módulo funcional da API.

2.5.1. REGIÃO DO PROGRAMA DE USUÁRIO

Esta região é composta pela Biblioteca de Funções: é a parte da API que o usuário deve conhecer e utilizar adequadamente segundo um formato pré-definido. A biblioteca é constituída das funções que a API pode executar e que devem ser explicitamente chamadas pelo AP. Dependendo do tipo de serviço solicitado pelo usuário (Função Requisitante) ou do serviço solicitado ao usuário pela aplicação par (Função Respondedora), um procedimento diferente é disparado pela chamada da função. Esses serviços estão "agrupados" de acordo com a sua funcionalidade dentro da Biblioteca de Funções em:

a) Serviços de Alto Nível ou **HLS** ("High Level Service").
b) Serviços de Baixo Nível ou **LLS** ("Low Level Service")
c) Serviços Respondedores ou **RS** ("Responder Service")

As chamadas a esses serviços irão acionar as funções do Provedor de Serviço.

2.5.1.1. O MÓDULO HLS

Este módulo é composto de todas as funções que representam serviços de alto-nível da API. Esses serviços são aqueles que são "quebrados" em pedidos múltiplos de serviços da camada de aplicação. Um serviço de alto nível agrupa os serviços de baixo nível da API. Após o recebimento da(s) confirmação(ões) correspondente(s), aAPI responde à chamada do AP de forma única.

Um exemplo de função de alto nível foi citado no ítem 2.4.2.

2.5.1.2. O MÓDULO LLS

É constituído das funções de baixo-nível da API. As funções podem representar serviços confirmados, não-confirmados e serviços de indicação da API.

Os serviços confirmados casam os pares de primitivas de serviço de requisição e de confirmação da aplicação, daí a necessidade da interação com o módulo especializado em serviços confirmados, o CSP, que realiza os procedimentos necessários de um serviço que espera confirmação.

Já se o serviço é não-confirmado, a função do provedor é de apenas mapear os parâmetros contidos na função nos parâmetros da primitiva de requisição e enviar.

Quanto aos serviços de indicação da API, esta repassa primitivas de indicações, que tenham sido "filtradas" pelo módulo IFA, ao usuário.

2.5.1.3. O MÓDULO RS

É o módulo que lida com as funções respondedoras da API. Compreende o conjunto de serviços que irão disparar as ações adequadas como respostas às indicações de serviços não solicitados inseridos na API.

A inteligência de uma função respondedora existirá no

módulo IFA, que fica em estado dormente ao se estabelecer uma determinada associação e é disparada com a chegada à interface de um serviço não solicitado, porém aceito.

Uma função respondedora pode ser programada para responder a certas solicitações do par remoto que passem por certos critérios no filtro IFA e pode inclusive ser reconfigurada: novos requerimentos sobrepondo-se aos antigos (fluxos de controle - linhas pontilhadas - da figura 2.7b).

Para tal, o usuário delega recursos e capacidades ao módulo IFA e provê critérios e capacidades de aceitação de indicações, de tal modo que se uma indicação é julgada inaceitável, uma confirmação negativa irá retornar ao usuário par. Caso contrário, a função respondedora determinará ações específicas locais (tais como acesso a variáveis da rede, execução de tarefas físicas num robô ...), e uma confirmação positiva retornará ao usuáro par.

2.5.2. REGIÃO DO PROVEDOR DE SERVIÇO

É nesta região que a API realiza os procedimentos para prover serviços ao AP.

As funções de cada módulo contribuem de forma cooperativa na realização dos serviços.

Nos módulos existirão filas, variáveis locais de armazenamento e outros objetos necessários para o desempenho de suas funções.

Os seguintes módulos funcionais compõem a região do Provedor de Serviços:

a) Provedor de Serviços de Alto Nível ou **HLSP** ("High Level Service Provider")
b) Provedor de Serviços Confirmados ou **CSP** ("Confirmed Service Provider")
c) Filtro de Indicações e Arbitragem ou **IFA** ("Indication Filter & Arbitrator")
d) Provedor de Serviços de Primitivas ou **PSP** ("Primitives Service Provider")

Dentro da linha de raciocínio do modelo de referência da OSI/ISO, cada módulo funcional existente no Provedor de Serviços da API poderá ser um entidade particular. Múltiplos AP's (ou múltiplas instâncias do AP) podem requerer serviços simultaneamente. Para cada módulo funcional provedor (entidades da API) poderão existir chamadas diversas, pertencentes a AP's (ou instâncias de AP) distintos. Na implementação desses módulos, deve-se assegurar o processamento adequado (ordenação de solicitação, identificações, etc) para as múltiplas solicitações enviadas. Se os serviços forem orientados a conexão, também deverão ser introduzidos temporizadores durante as etapas do processamento.

A interação entre os módulos do Provedor de Serviços determinam a execução do serviço solicitado, mediante uma lógica e um relacionamento usuário-provedor embutidos no Provedor. Essa lógica depende do serviço solicitado através de uma chamada de função existente na interface.

2.5.2.1. O MÓDULO HLSP

Este é o provedor de serviços de alto nível da API. Ele é "ativado" por aquelas funções classificadas como de alto-nível (ex: CONNECT, FILE GET, FILE PUT). As funções desempenhadas por este módulo podem ser resumidas dentro de 3 etapas:

. o HLSP determina, em relação à função uma determinada sequência de pedidos de serviços a outros módulos (CSP, IFA OU PSP) necessárias à realização da função de alto nível.

. uma vez determinada, ele controla essa sequência de requisições de serviços de forma a poder identificá-las no conjunto.

. depois , o HLSP faz uso livremente das funções dos módulos CSP, PSP e IFA, que por sua vez, desempenham suas funções independentemente e respondem ao HLSP.

. após todas as requisições estarem concluídas além de todo o processamento relacionado ao atendimento do serviço solicitado, ou caso algum erro irrecuperável aconteça, o HLSP coloca a informação resultante na área de dados do usuário e sinaliza o resultado da chamada.

2.5.2.2. O MÓDULO CSP

Este módulo desempenha as funções relacionadas a serviços confirmados, ou seja, providencia o casamento das confirmações correspondentes às requisições feitas tanto pelo HLSP (no caso de um serviço do alto nível) ou diretamente pelo AP (no caso de uma chamada de serviço confirmado de baixo nível).

CSP desempenha essas funções fazendo uso das funções do módulo PSP.

As atividades do CSP consistem em:

. associar um identificador a cada requisição que ele receba, do usuário, ou HLSP, para casar com a confirmação que posteriormente chegará do usuário par

. mapear os parâmetros necessários na primitiva de request

. passar a primitiva de request, já identificada, ao PSP

. esperar pela resposta do PSP

. ao receber a primitiva de confirmação (positiva ou negativa) mapear os parâmetros vindos na primitiva de confirm no parâmetros do usuário e sinalizar o emissor do pedido (HLSP ou LLS).

2.5.2.3. O MÓDULO PSP

Sua função é de enviar primitivas de *request* e *response* e de receber primitivas de *indication* e *confirm* para/da máquina de protocolo.

O papel do PSP resume-se em:

. examinar as condições locais antes de enviar ou receber primitivas de serviço, verificando se não houve término de associação, se há permissão de receber primitivas de confirmação ou indicação, se a primitiva está semanticamente correta, etc.

. de posse da primitiva, passá-la ao seu destinatário (Máquina de Protocolo, CSP, IFA ou LLS)

. após passar, sinalizar o remetente.

2.5.2.4. O MÓDULO IFA

As propriedades de seleção de primitivas de indicação oriundas de usuários remotos, e que devem ser aceitas pelo AP, estão na interface, embutidas no módulo funcional IFA. O IFA mantém o controle da filtragem, através de alguns critérios de aceitação, passados pelo HLSP (corresponde ás funções de alto-nível) e pelas funções respondedoras (associadas aos tipos de indicação pre-estabelecidos), e diante desse controle, toma as ações adequadas ao tratamento das indicações aceitas.

O IFA processa as solicitações do HLSP da seguinte forma:

1- O HLSP informa ao IFA que tipo de indicação ele espera receber e passa um parâmetro de saída que deverá conter os dados esperados.

2- O IFA passa a receber as primitivas daquele tipo de

indicação fornecido pelo HLSP (dependendo do tipo, ele pode reservar todas as primitivas e responder de uma vez ao HLSP).

3- O HLSP informa posteriormente o critério de término de aceitação daquele tipo de indicação.

Um exemplo de função de alto-nível que provê a filtragem de indicações é o LISTEN. Quando o AP deseja receber pedidos de associação de iniciativa de APs remotos, ele faz uma chamada de função de alto-nível à API, que é o LISTEN. A partir daí, o IFA tem permissão para receber um tipo de indicação que atenda a essa solicitação do módulo HLSP. Esse tipo de indicação é a primitiva CONNECT-indication. Para informar ao IFA que não receba mais indicações desse tipo, o AP insere um STOP LISTEN na API. O AP pode fazer tantos LISTEN e STOP LISTEN quanto queira, desde que as aceitações correspondentes (pelo ANSWER) estejam limitadas até o número máximo de associações permitidas àquela AE, em particular.

O IFA processa as solicitações das funções LLS do seguinte modo:

1- A função LLS habilita o IFA a receber indicações

2- O IFA verifica, em sua área de armazenamento, se existe uma indicação e preenche o parâmetro Indication_name fornecido pelo usuário (parâmetro de saída explícito) com o nome da primitiva de indicação. Coloca os dados na área de saída fornecido pelo usuário e o sinaliza.

3- A função LLS desempenha as devidas ações, de acordo com o tipo de indicação recebida.

Como exemplo, considera-se uma função de baixo-nível INDICATION-RECEIVE é chamada pelo AP. O IFA já pode ter armazenado, à priori, várias indicações (depende da sofisticação e recursos da API). Caso haja indicação, retira uma, identifica-a e coloca os dados no parâmetro de saída do usuário. Caso não haja, fica à espera.

A cada função INDICATION-RECEIVE corresponde uma só primitiva de indicação a ser fornecida pelo IFA ao usuário LLS.

O IFA processa as solicitações das funções RS da seguinte maneira:

1- Registra no filtro o tipo de indicação associada à função respondedora. De acordo com o tipo, são estabelecidos critérios de aceitação e os recursos necessários que devem estar disponíveis (ex: funções de acesso a variáveis).

2- Para cada tipo de indicação e dentro dos critérios de aceitação, são desempenhadas as ações respondedoras associadas.

3- Define-se um critério de término de filtragem para as funções respondedoras.

A título de resumo, a tabela 2.1 permite uma visão mais clara das atividades dos módulos funcionais da API e de suas interações, de acordo com o tipo de função solicitada.

FUNÇÕES DE ALTO NÍVEL

1. recebe chamada de função
2. testa validade dos parâmetros
3. cria identificação de chamada alto-nível
4. processa informações de alto nível
5. cria identificação de chamada
6. mapeia parte dos parâmetros em primitiva de requisição
7. envia requisição
8. espera
9. recebe confirmação
10. mapeia primitiva em parâmetros de saída
11. verifica condição de término se não terminou, volta ao passo 4
12. mapeia parâmetros finais
13. retorna ao usuário

FUNÇÕES RESPONDEDORAS

1. Recebe autorização para aceitar indicações
2. Prepara função respondedora
3. Prepara o filtro (IFA)
4. Aguarda
5. Recebe indicação de indicação
6. Mapeia primitiva de indicação
7. Determina a ação
8. Sinaliza usuário

FUNÇÕES AUXILIARES

1. recebe chamada de função
2. testa validade dos parâmetros
3. mapeia parâmetros entrada/saída
4. retorna ao usuário

FUNÇÕES CONFIRMADAS

1. recebe chamada de função
2. testa validade dos parâmetros
3. cria identificação de chamada
4. mapeia parâmetros na primitiva de requisição
5. envia à Máquina de Protocolo
6. espera (com WAIT ou não)
7. recebe confirmação
8. mapeia primitiva confirmação nos parâmetros de saída
9. retorna ao usuário

FUNÇÕES NÃO-CONFIRMADAS

1. recebe chamada de função
2. testa validade dos parâmetros
3. mapeia parâmetros na primitiva de requisição
4. envia à máquina de Protocolo
5. Sinaliza envio ao usuário

Tabela 2.1 - Procedimentos da API no desempenho de suas funções

2.6. OS SERVIÇOS DA API

Os serviços da API podem ser separados em 3 classes, quanto ao seu tratamento e objetivo final.

O primeiro grupo, **serviços de suporte**, refere-se aos serviços da API que dão suporte aos outros serviços. São os serviços que mais dizem respeito ao objetivo da API de interpor-se entre o programa aplicativo e o sistema operacional, de forma que o AP fique mais desvinculado das particularidades do funcionamento do S.O.

O segundo grupo, de **serviços de gerenciamento de conexão**, são aqueles que tratam dos aspectos envolvidos num estabelecimento de associação entre os AP's.

O terceiro grupo, **serviços específicos da aplicação**, envolve todos os serviços associados a uma aplicação. Como a API é genérica, esse grupo pode ser formado de vários subgrupos, cada um representando uma aplicação específica.

2.6.1. OS SERVIÇOS DE SUPORTE

Num sistema de comunicação, além da preocupação da transferência de dados em si, outros aspectos são incorporados para alcançar o objetivo final da comunicação. Esses aspectos podem envolver mecanismos de sincronização, alocação de espaços de memória, etc.

A API conta com algumas funções especiais para dar suporte ao usuário, interfaceando-o com o suporte operacional residente. Sendo as chamadas a esses serviços padronizadas e bem definidas, o usuário (AP) as utiliza de modo transparente e torna-se mais independente do sistema operacional sob o qual está operando. A API absorve essas funções, provendo, assim , maior portabilidade aos programas aplicativos para ambientes de sistemas operacionais heterogêneos.

Esses serviços serão mencionados no capítulo 4, por ocasião de suas utilizações na implementação e aqui só é feita uma breve descrição de cada um. (Para maiores detalhes, ver [M3]).

a) Serviços de gerenciamento de eventos

Com a finalidade de prover um suporte de comunicação assíncrona entre os AP's comunicantes, dois serviços são providos pela API: **WAIT** e **NOTE**, que implementam as funções de semáforos de processos cooperantes. Com essas funções o AP não se preocupa com as especificidades do S.O. corrente, quanto à sua função de sincronizar a comunicação entre processos.

A API deve gerenciar nomes de eventos (passados pelo usuário como parâmetro de entrada) associados às chamadas das funções assíncronas.

b) Serviço de alocação dinâmica de DCBs

Para prover uma padronização nas chamadas da interface e ao mesmo tempo obter uma maneira de compatibilizar softwares com versões diferentes, foi adotado um mecanismo de passagem de parâmetros do usuário de forma única e bem definida. São os DCBs (Data Control Block) estruturas que contêm os parâmetros das chamadas de serviços da API, os quais são tratados diferenciadamente, de chamada a chamada, dependendo do tipo de informação que carregam.

O número de parâmetros pode variar livremente de uma chamada para outra sem se alterar a "fachada" da chamada de função da API, mantendo-se assim a sua padronização.

São colocados em DCBs todos os parâmetros opcionais e condicionais que um serviço possa suportar.

A API tem uma função de suporte que o usuário pode usar para pedir à alocação e a inicialização dos DCBs, tanto os de entrada, como os de saída. Chama-se **DYNAMIC INITIALIZE DCB**.

Com essa função, percebe-se que a API se coloca entre o usuário e o S.O também quanto à alocação de espaço de memória. Além da interface verificar se há disponibilidade de memória e, se houver, fazer a alocação para aquele determinado tipo de estrutura, ela ainda verifica, parâmetro a parâmetro, que valores "defaults" ou que valores iniciais o DCB deve levar.

A área comum de dados partilhada por AP e API é gerenciada pelo usuário. Ele fica responsável pela desalocação do espaço

utilizado, ou seja, ele utiliza a função específica da linguagem de programação de sua aplicação para liberar o espaço de memória que não mais lhe interessa.

c) Serviço de tradução de código de erro

Quando um usuário faz uma chamada de função à API, ele espera, do retorno da função, um código que indique se o serviço foi executado corretamente ou se houve um erro. Esse código de retorno de função é do tipo RETURN-CODE e terá o valor 0 (zero) se a função foi desempenhada sem erro, e um determinado valor, diferente de zero, para cada um dos erros possíveis.

A API pode ser chamada pelo usuário para traduzir esse "código" de erro numa "cadeia de caracteres" que o torne legível ao usuário.

A função de suporte da API que presta esse serviço ao usuário chama-se **GET PRINTABLE ERROR**.

d) Serviço de notificação de liberação de recursos

Os recursos alocados a uma determinada conexão feita entre APs não são ilimitados, pois estão sujeitos aos recursos locais do provedor. Essas restrições dizem respeito à disponibilidade de enfileirar e processar pedidos de serviços do usuário. As restrições de recursos referem-se ao mecanismo de troca de mensagens no modelo de interação entre AP's. Não havendo espaço nas filas, significa também não haver capacidade de processamento de pedido naquele instante. Porém, essa restrição pode acabar à medida que os pedidos de serviço vão sendo processados.

A API fornece mais uma função de suporte ao usuário, para avisá-lo do término da restrição de recursos dentro de uma conexão.

O usuário deve fazer o pedido através da função **NOTE WHEN CLEARED** após uma chamada de função que tenha no código de retorno um "NO CONNECTION RESOURCE". Um nome de evento deve estar associado à função para identificar o usuário solicitante, e a identificação da conexão da qual o usuário espera liberação de recursos também deve ser passado como parâmetro da função.

O usuário pode usar as funções de suporte WAIT e NOTE para se liberar durante a espera de recursos.

2.6.2. SERVIÇOS DE GERENCIAMENTO DE CONEXÃO

Essa classe de serviços concerne às etapas de estabelecimento e término de associação entre os AP's comunicantes. Já foi dito que a API é genérica e poderá suportar através destes serviços, associações múltiplas de AP's englobando serviços de aplicações diversos. Os serviços de Gerenciamento de Conexão estão presentes em toda e qualquer API.[1] São essenciais, já que os atuais protocolos de aplicação OSI são todos orientados a conexão. A especificação dos aspectos de estabelecimento de associação estão contidos em [M4], e seguem as regras do modelo geral da API e a descrição da arquitetura da camada de aplicação da OSI. Detalhes de implementação dos serviços de gerenciamento de conexão são abordados em [T1].

Descreve-se agora sucintamente as etapas de um processo aplicativo envolvendo esses serviços.

O modelo de interação da API, em conformidade com o modelo de referência da ISO, define como usuário dos serviços da API, um Processo de Aplicação ou Programa Aplicativo (AP), que reside no sistema aberto de rede.

A qualquer instante de seu processamento, o AP estabelece uma invocação de AE o que significa que esse AP passa a existir para a rede nesse instante. A função da API que o AP deve usar para estabelecer uma invocação de AE, é a AE_ACTIVATION. Esta função necessita, como parâmetro de entrada, o nome de uma AE, que a identifique. Por isso, antes de uma AE ser invocada (ativada) ela deve estar registrada no diretório da rede.

---

[1] No documento [M4] o termo "conexão" é usado indiscriminadamente para referenciar uma associação de aplicação. Conserva-se neste texto o termo "Gerenciamento de Conexão" aplicando-o às atividades de controle de associação.

Registro de uma AE

Para que uma entidade de aplicação (AE) torne-se um membro de uma rede alguns requisitos devem ser observados:

- um endereço de apresentação deve ser atribuído à entidade, através do qual ela poderá ser acessada por qualquer outro membro da rede;
- seus títulos devem constar no diretório (uma AE pode ter nenhum, um só ou múltiplos títulos);
- as características de uma AE, que são conhecidas e manipuladas pelo gerenciamento do ACSE, devem estar bem definidas;
- os recursos e capacidades da AE devem ser estabelecidos previamente com os parâmetros de restrições

Todos esses requisitos fazem parte das informações estáticas sobre uma AE registradas no diretório. A implementação das funções de "registro de AE" envolvem aspectos locais. Uma vez disponível o protocolo de Serviço de Diretório (DS), as funções de registro, de gerenciamento dos atributos e de controle de acesso estarão em conformidade com o protocolo.

Ativação de uma AE

Foi mencionado, no ítem anterior, que uma AE deve ser invocada para tornar-se um elemento ativo na rede, ou seja, para prover/obter serviços de rede.

A invocação de uma AE é a sua declaração de existência e disponibilidade aos usuários de rede. Uma invocação de AE dispara as funções de gerenciamento do sistema local que atuarão sobre a AE com o objetivo de manutenção das propriedades dinâmicas da mesma.

A função da API que, executa o procedimento de ativação da AE é a função do alto-nível **AE_ACTIVATION**.

Define-se, nesta etapa, um ponto de acesso aos serviços de rede, porém é necessário, a partir daí, estabelecer uma "ligação" fim

a fim com outra(s) AE(s) ativa(s). Entra-se na fase de estabelecimento de associação.

Estabelecimento de Associação entre AE's

Após uma AE ser ativada, o usuário da invocação pode explicitamente solicitar uma associação com outra AE ou declarar-se receptivo a indicações de associações enviadas por outras AE's.

No primeiro caso, o usuário da invocação de uma AE, o AP, deve fazer uso da chamada da função **CONNECT**, futuramente mapeada num CONNECT_request.

No segundo caso, a AE declara sua permissão em receber indicações de associação (CONNECT_indication) através da chamada da função **LISTEN**. Para declarar que não tem mais interesse em receber qualquer indicação de associação, para aquela invocação de AE, o AP insere um **STOP LISTEN**.

Como resultado de um pedido de associação, onde o ACSE e a ASE se cooperam,um identificador de associação (*connection_id*) é atribuído à associação entre AE's ativas [12].

O identificador de associação será parâmetro obrigatório nos pedidos de serviços subsequentes.

Reinicialização de uma Invocação de AE

É muito desejável e útil a capacidade de manter viva uma invocação de AE, caso ocorra, durante o tempo de execução, falha irrecuperável do sistema. Para tal, exigem-se procedimentos dependentes do ambiente local, não especificados em ISO/OSI.

Porém, uma invocação de AE pode contar com uma função que solicita um procedimento de "limpeza drástica" ou reinicialização da invocação. Ao usar a função de alto-nível **AE_RESET**, o usuário da invocação de AE está solicitando abortos para todas as associações (já estabelecidas ou pendentes) dentro daquela invocação e o "stop listen" de futuras indicações de associações. A invocação de AE retorna ao seu estado inicial, justamente após ela ter se tornado ativa.

Término de Associação entre AE's

Uma associação existente numa invocação de AE pode terminar normalmente, quando o término é negociado entre os usuários ou abruptamente, quando uma associação é terminada sem negociação.

O término normal de associação envolve um serviço confirmado onde a API usa o par de primitivas de requisição e confirmação. O AP que deseja terminar a associação, faz a chamada da função de baixo-nível **RELEASE REQUEST** e aguarda uma confirmação positiva ou negativa de sua solicitação.

Já um término anormal ou abrupto ocorre quando o usuário envia uma requisição de aborto ou quando recebe uma indicação de aborto do provedor. A requisição do usuário é feita pela função **ABORT**. O recebimento de indicação de aborto traz dois significados: um aborto gerado pelo usuário par (A_ABORT_indication) ou um aborto gerado pelo Provedor (AP_ABORT_indication).

A Preparação para Recebimento de Indicações

Um AE deve permitir receber indicações, seja atuando num papel Cliente (recebimento de indicações de Abort, Reject, Conclude, Information_Report e Unsolicited_Status) ou especialmente num papel Servidor, no qual provê serviços de diversos tipos.

A função que permite o recebimento de indicações por parte da AE é a **INDICATION RECEIVE**.

A Desativação de uma AE

O procedimento reverso de uma ativação de AE é a sua desativação. A função que o AP utiliza para esse objetivo é a **APPLICATION ENTITY DEACTIVATION**. O identificador da invocação da AE deve ser passado como parâmetro de entrada explícito. Como consequência dessa chamada de função, nenhum serviço mais poderá ser solicitado através dessa invocação, pois o ae-label tornar-se-á inválido.

2.6.3. SERVIÇOS ESPECIFICOS DA APLICAÇÃO

É através desses serviços específicos que o AP desempenha sua atividade total. Numa invocação de um AP (ativação de uma tarefa qualquer), uma entidade de aplicação é ativada e a partir daí, serviços relativos aos Elementos de Serviços de Aplicação (ASE's) envolvidos são requeridos/recebidos pela AE.

A característica de modularidade da API pode ser exemplificada aqui. Dependendo do tipo de dispositivo que a interface atende, nem todas as **classes de serviços** existentes no sistema de aplicação serão necessárias, já que o conjunto total de serviços especificados objetivam atender a qualquer aplicação possível num ambiente aberto. Por exemplo, a API terá mais funções implementadas naquele dispositivo mais complexo e com maiores capacidades funcionais, e menos funções naqueles que implementam apenas os serviços restritos às suas capacidades funcionais.

Os serviços para a Aplicação MMS serão mencionados no capítulo 4.

2.7. CONCLUSÕES

O modelo de arquitetura da interface consiste de módulos funcionais que podem ser vistos como entidades com atribuições independentes.

A abordagem modular da interface de Aplicação (API) procura refletir a realidade dos sistemas de aplicação onde dispositivos e softwares aplicativos variam em complexidade e niveis de funcionalidade. A interação entre os módulos objetivam a realização, de forma cooperativa, dos serviços da API em favor do usuário (AP).

A API assume diversas atividades dentre as quais a de gerenciar as múltiplas invocações das entidades de aplicação (AE) e suas múltiplas associações.

A API ainda provê outras capacidades funcionais como: suporte a operações síncronas e assíncronas, filtragem de indicações, gerência de recursos locais.

A biblioteca de funções da API pode ser configurada de forma modular, adequando-se às necessidades da aplicação.

CAPÍTULO 3

ASPECTOS DE ESPECIFICAÇÃO DE IMPLEMENTAÇÃO DA API PARA O MMS NO SISDI-MAP

Foi apresentado no capítulo anterior o modelo de uma API genérica, e sua especificação para atender ao interfaceamento de programas aplicativos com vários sistemas de comunicação existentes no âmbito das aplicações.

Neste capítulo apresentam-se os aspectos da especificação do sistema de comunicação para controle de manufatura, o MMS (Manufacturing Message Specification), e as particularidades de especificação da API para este protocolo.

No SISDI-MAP, o protocolo MMS é atualmente o representante único dos elementos de serviços de aplicação específicos (SASE). Outros SASE's deverão ser incorporados ao SISDI-MAP sem grandes alterações no sistema. A seção 3.2 descreve o SISDI-MAP e mostra como a API-MMS está integrada nele.

3.1. O MMS

Um usuário (AP) solicita serviços da API, através de chamadas às funções existentes na biblioteca da API. Se o serviço solicitado se refere ao conjunto de serviços MMS, a API solicitará diretamente serviços MMS através de suas primitivas de serviços, inserindo-os na máquina de protocolo MMS. Os serviços MMS, por sua vez, são providos à API, pela máquina de protocolo MMS, fazendo uso dos serviços do ACSE e dos serviços de Apresentação. O protocolo MMS mapeia a primitiva recebida da API na PDU correspondente. O exemplo da figura 3.1 mostra a transformação dos dados entre AP, API e MMS.

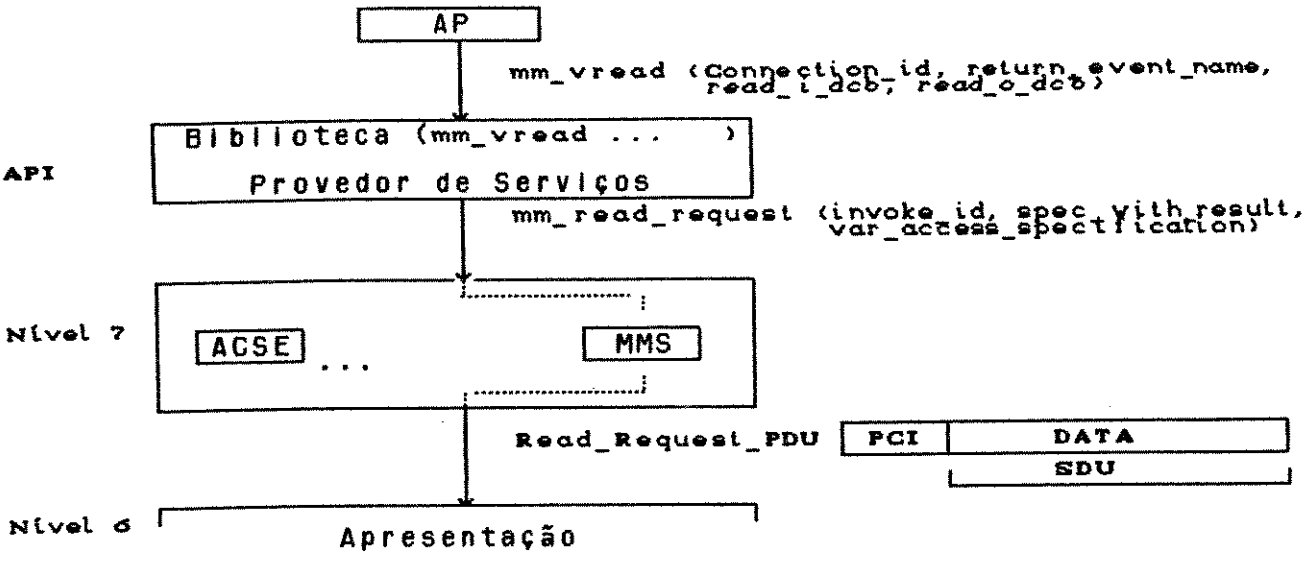


Figura 3.1 - Transformação dos dados entre AP, API e Protocolo MMS

Os serviços MMS dizem respeito à comunicação e ao interprocessamento de informações entre dispositivos programáveis do ambiente industrial. O protocolo MMS, apoiado por outros protocolos, contribui para a manufatura integrada por computador, ou seja, o CIM, onde os elementos interagem entre si com o objetivo de contribuir com suas atividades distintas para a realização de uma atividade de propósito geral que as engloba [L5].

Através de **modelos abstratos** é possível a representação da interconexão de dispositivos heterogêneos e a comunicação entre eles, com as suas propriedades estáticas e dinâmicas.

As propriedades estáticas dizem respeito a regras, localização, nomeação, representação, formatos, etc.

As propriedades dinâmicas dizem respeito a atividades, comportamentos, estados, eventos e interação.

Numa visão mais externa podemos observar, num ambiente de controle e monitoração fabril, diversos nodos interconectados e que neles estão sendo realizadas as tarefas distribuídas de um sistema aberto real. Nesses nodos estão implementados os protocolos

necessários à inter-operação do sistema. No esquema da figura 3.2 pode-se observar os vários elementos integrantes do universo-rede, e a abstração embutida em cada elemento ou sub-elemento de forma a retratar seu comportamento visível pelo usuário. Aqui estão representados: objetos virtuais, (variáveis, funções, etc), dispositivos virtuais, entidades abstratas, interfaces, protocolos, tarefas...

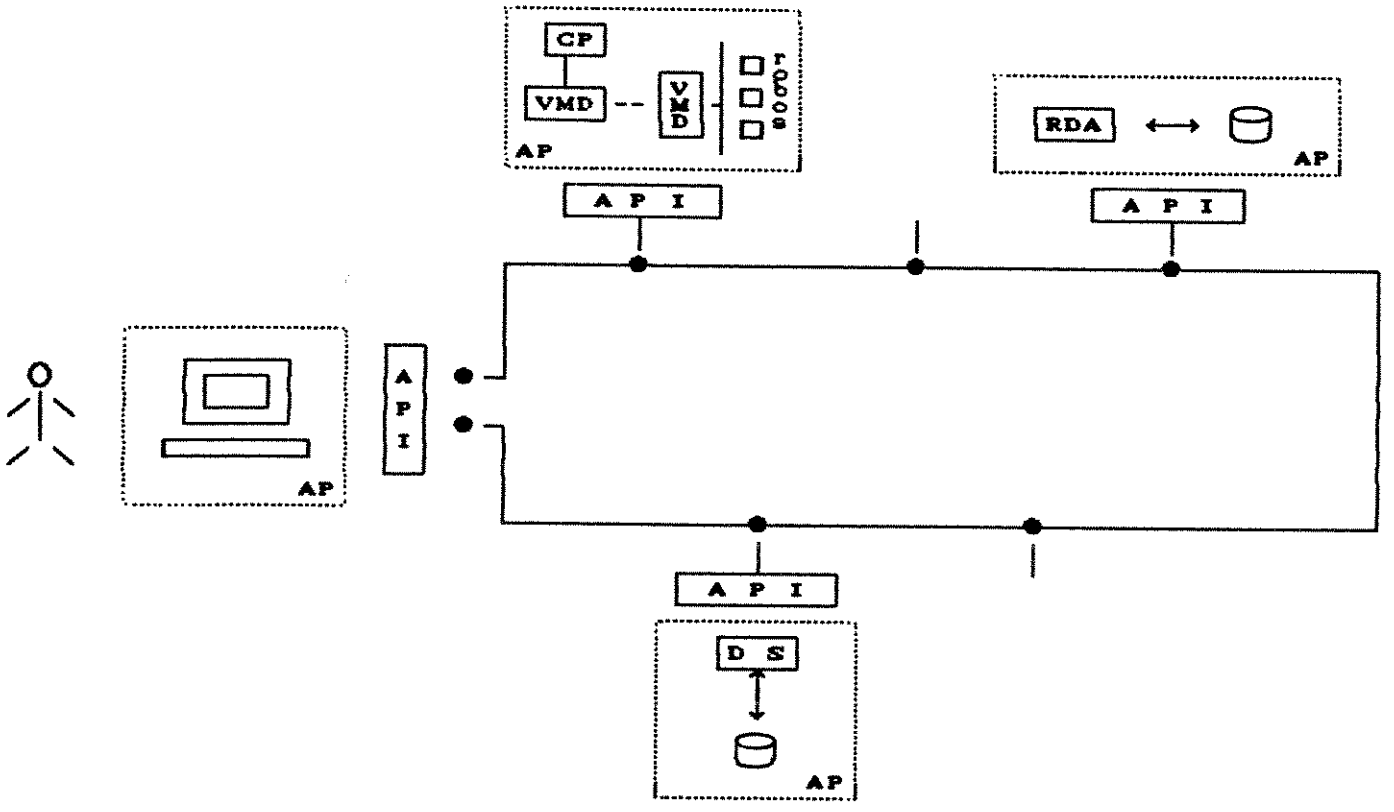


Figura 3.2 - Modelo da rede MAP e interação entre elementos.

Quando dois APs se comunicam, seus comportamentos são determinados sob os aspectos selecionados no esquema conceitual pre-estabelecido e compartilhado por ambos; implementa-se em cada AP toda a abrangência do esquema conceitual a que ele é destinado, com seus requisitos e suas regras.

A partir daí, conecta-se ao AP, um ou mais dispositivos virtuais (VMD), com todos os seus domínios e funções. Quando o

processo de aplicação é ativado, ou melhor, quando ocorre uma instância do AP, ocorre também a invocação da entidade abstrada do tipo VMD. Dentro do domínio do VMD, todos os recursos (serviços) a ele vinculado naquela instância, passam a ser acessíveis pelo usuário, representado pela instância do AP.

O MMS define um modelo abstrato abrangente que incorpora as interações entre esses elementos, e representa suas funcionalidades de maneira visível ao usuário da interação, na forma de requisitos procedurais, através dos quais os elementos possam obter a execução de serviços uns dos outros, adotando-se uma relação cliente/servidor.

3.1.1. MODELO ABSTRATO DO MMS

Pretende-se agora verificar as especificidades do MMS dentro do ambiente OSI somente para identificar aspectos relevantes ao relacionamento API-MMS. Refira-se à especificação do MMS, seus serviços e seu protocolo em [I6] e [I7]. O MMS foi também objeto de teses de mestrado em cujas referências se encontra abordagens bem distintas, de modelamento e implementação [T3] e [T5]. O MMS deve:

- prover serviços que permitam interações de processo aplicativos (AP's), (esses serviços devem atender as atividades de associação entre AP's chamador e chamado).

- Definir um comportamento externo dos servidores de aplicação, ou seja, um modelo VMD de dispositivo virtual. O termo MMS-Server é aplicado, nesse sentido, para descrever a percepção externa de comportamento do dispositivo dentro de uma instância particular de um pedido de serviço feita pelo usuário (MMS-Client).

- Fornecer formas, regras e representações de todos os serviços disponíveis, para que sejam utilizadas adequadamente pelo usuário dos serviços.

- Definir objetos com atributos que constituam a entidade

abstrata a qual é afetada pelas operações desencadeadas pelos serviços MMS, (ou seja, os serviços MMS podem modificar as características dinâmicas desses objetos dentro de uma instância do objeto).

Uma descrição rápida sobre cada um desses aspectos é dada em seguida.

3.1.1.1. INTERAÇÃO DE PROCESSOS APLICATIVOS

Um AP é a representação abstrata dos aspectos funcionais de um sistema real. APs devem se comunicar com outros APs, na realização de uma tarefa distribuída.

O MMS deve dispor de serviços para a etapa inicial de troca de informações entre os APs para que, posteriormente, estes possam usar suas funções de processamento de forma cooperativa e compatível. Os serviços de gerenciamento de contexto estão voltados para estes aspectos (é conveniente lembrar aqui o relacionamento estreito desses serviços com os serviços de gerenciamento de conexão da API).

É também necessário diferenciar entre os serviços que requerem uma confirmação do par remoto e aqueles que não requerem, como, por exemplo, os de verificação de estado, de diagnóstico, etc.

Para permitir total interação dos APs e induzí-los a desempenhar alguma ação, o MMS associa um tipo de primitiva de serviço (*request*, *indication*, *response-positive*, *response-negative*, *confirm-positive*, *confirm-negative*) àquela situação específica.

A API corresponde ao modelo de interação MMS, provendo serviços dos tipos requisitantes e respondedores, fazendo uso de suas funções de provedor de serviços confirmados, não-confirmados, etc., conforme o modelo geral descrito no capítulo anterior.

O Provedor de Serviços de Primitivas exerce o papel de manipulador de primitivas MMS enviando-as e recebendo-as para/da Máquina de Protocolo MMS.

3.1.1.2. O DISPOSITIVO VIRTUAL DE MANUFATURA

A especificação do MMS-draft 6 introduz um novo conceito de abstração de dispositivos, chamado de VMD.

O VMD expressa os elementos estruturais e os objetos (descritos no ítem seguinte) envolvidos no dispositivo real.

Um AP conterá um ou mais VMDs. Por exemplo, um nodo MMS ao qual estão conectados vários dispositivos de manufatura, poderá ser implementado como um único AP, contendo vários VMDs, ou por vários APs, cada um contendo um VMD distinto [L1].

Um certo VMD pode ser endereçado por várias instâncias do AP, uma vez que ele pode ser compartilhado por seus vários clientes. Os clientes do VMD irão ter a visão de um dispositivo conectado a eles individualmente.

Os recursos virtuais dos VMDs são mapeados nos recursos físicos num nível inferior e um mecanismo existente nos APs os torna acessíveis através dos VMDs. Assim, os clientes dos vários VMDs podem necessitar de coordenação dos seus acessos a um simples recurso real, que pode ser obtida, associando-se a cada VMD, um controle de semáforo virtual. Os semáforos virtuais possibilitam a abstração independente dos VMDs sob a visão dos usuários de serviços MMS, e para tal, dispõe-se de serviços de gerenciamento de semaféros. Espera-se que se possa atribuir à API funções relacionadas ao mecanismo de mapeamento dos recursos virtuais nos recursos físicos do VMD, liberando o AP dessa atividade.

Ainda sobre os VMDs, a estes estão vinculados os conceitos de domínios definidos como um subconjunto dos recursos do VMD alocados de forma estática ou dinâmica. Por exemplo, num controlador numérico, os domínios devem representar tanto os recursos físicos - como memória, dispositivos de E/S - como também uma parte da funcionalidade total do controlador. Objetos manipuláveis pelo MMS podem ter suas funcionalidades restritas ao escopo do domínio ao qual pertencem.

No modelo do VMD, o elemento que exprime a utilização dos recursos do mesmo, ou dos seus domínios é a função executiva. Nela estão incluídas os aspectos de operação do VMD.

A função executiva está associada ao P-SAP, através do qual o usuário AP acessa as capacidades operacionais do VMD.

É necessário que a API implemente as funções que suportem o conceito de VMD, que representarão as necessidades do usuário de obter sua identificação, informações sobre as capacidades do dispositivo, seu estado de operação, etc.

Uma estrutura simplificada de um VMD está representada na figura 3.3.

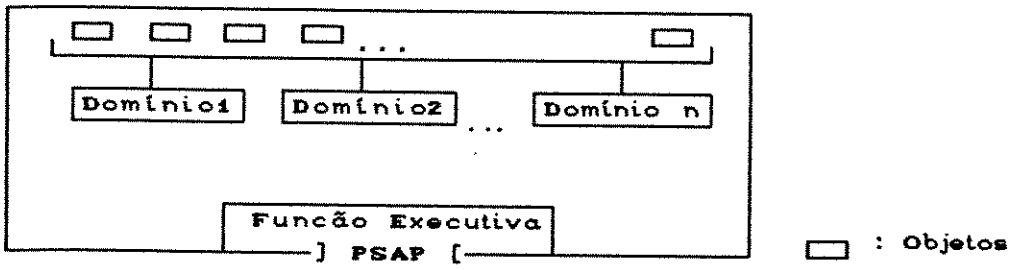


Figura 3.3 - Modelo simplificado de um VMD

3.1.1.3. OBJETOS, ATRIBUTOS E OPERAÇÕES SOBRE OBJETOS

O MMS faz uso da técnica de modelamento abstrato de objetos para descrever o relacionamento dos dispositivos MMS com os serviços MMS.

Os serviços MMS atuam sobre um VMD de forma a produzir transações em seus objetos.

Quando um VMD recebe primitivas de serviços, através de sua função executiva, ele "checa" a validade do acesso aos objetos e cria uma transação que governa o processamento relacionado com o serviço.

O MMS define uma certa quantidade de objetos (variáveis, eventos, semáforos, invocação de programas, etc). Cada um deles representa uma entidade abstrata que exibe certas características que podem ser afetadas pelos serviços e operações do MMS, criando-se uma instância do objeto. Cada instância do objeto deve ser unicamente identificada entre todos os objetos.

Associados a essas instâncias do objeto estão os atributos de identificação, endereços, operações permitidas, etc. Alguns desses

atributos são condicionais, outros opcionais. Um atributo, ou alguns deles numa forma combinada o identificam unicamente numa instância específica: são os atributos-chaves.

No MMS, os objetos são sintaticamente definidos do seguinte modo:

Objeto: (nome do Objeto)

Atributo chave: (nome do atributo)
:
Atributo chave: (nome do atributo)
Atributo (nome do atributo)
.
.
Atributo (nome do atributo)
Condição: (expressão de condição)
Atributo (nome do atributo)

OBS: . Atributos podem estar aninhados, para representar uma hierarquia entre eles.
. Condições podem ser expressas em qualquer lugar dentro da lista de atributos, com a convenção de que todos os atributos sujeitos à condição estejam embutidos na condição.

Objetos MMS são normalmente referenciados por nomes. O nome de um objeto deve ser único dentro do escopo de existência do objeto. No MMS, eles podem ser definidos dentro de 3 escopos:

. escopo específico de Associação de Aplicação (AA specific)
. escopo da instância do VMD
. escopo do domínio específico no VMD.

Na seção 3.2.3 (Tipos de dados da API-MMS) apresenta-se a estrutura do nome de objeto, o qual ocorre frequentemente na especificação de serviços MMS e cuja API-MMS faz referências em seus serviços.

As instâncias dos objetos, identificadas por nomes, são classificadas no MMS sob os aspectos funcionais que desempenham na aplicação. As classes dos objetos MMS abrangem: variáveis, semáforos, eventos, "journal" ou journal, domínios, invocação de programas e estação de operador. Nem sempre eles existem em todos os 3 escopos. A tabela 3.1 mostra as classes dos objetos MMS e seus respectivos escopos de atuação.

O modelo abstrato de objetos é ideal para uma manipulação com linguagem orientada a objetos. A linguagem C, por exemplo, conta atualmente com uma extensão, denominada C^{++} que trata elementos de forma global, significando um único objeto, com sua identificação, seus atributos e operações.

Acredita-se que numa versão futura da API o tratamento dos objetos MMS venha a ser introduzido e uma implementação mais facilitada decorra dessa alteração.

CLASSES DOS OBJETOS	ESCOPOS		
	VMD	DOMAIN	A-ASSOCIATION
Objetos Variável nomeada	x	x	x
Objetos Acesso espalhado	x	x	x
Objetos Lista_de_variáveis nomeada	x	x	x
Objetos Tipo nomeado	x	x	x
Objetos Semáforos	x	x	
Objetos Condição de eventos	x	x	x
Objetos Ação de evetos	x	x	x
Objetos Jornal	x	x	
Objetos Domínio	x		
Objetos Invocação de Programa	x		
Objetos Estação de operador	x		

Tabela 3.1 – Classes dos objetos MMS e seus escopos de operação

3.1.1. A DESCRIÇÃO DOS SERVIÇOS E REQUISITOS DE CONFORMIDADE

O MMS presta serviços ao usuário (AP) através de primitivas. As primitivas carregam toda a informação relacionada aos serviços por intermédio dos parâmetros associados. Os parâmetros são descritos em termos de tipos e de uso em cada primitiva especificada.

Os requisitos estáticos de uma implementação de serviços MMS devem ser considerados. Eles são chamados de conformidade estática. A conformidade estática descreve os requisitos de suporte aos tipos de dados das primitivas, bem como outras capacidades dos objetos MMS.

Os Blocos Construtivos de Conformidade (CBB - Conformance Building Blocks), implementam a conformidade estática, ou seja, descrevem o inter-relacionamento dos serviços MMS. Há dois tipos de relacionamento e eles estão representados pelos CBBs verticais e CBBs horizontais. O relacionamento descrito no CBB vertical diz respeito à relação dos serviços em termos de suas funções sobre um objeto MMS, e o relacionamento descrito no CBB horizontal descreve os serviços relacionados através deparâmetros (parâmetros que aparecem em vários serviços).

Os usuários MMS negociam quais serviços irão usar dentro de uma associação, atra s dos blocos de conformidade.

Todos esses aspectos de descrição de serviços MMS - primitivas envolvidas (*request*, *indication*, *response* e *confirmation*), parâmetros (obrigatórios, opcionais, condicionais), e CBBs (horizontais, verticais - são representados, no documento [16], através de tabelas. As tabelas contém até 6 colunas, contendo: o nome do parâmetro, sua ocorrência nos 4 tipos de primitivas e uma coluna para a especificação de CBB. Os parâmetros do serviço são listrados, um a um, horizontalmente na tabela.

Na especificação atual da API não é feita nenhuma referência à negociação de classes de conformidade. Contudo, entende-se que as funções API mantêm também um inter-relacionamento de serviços, que algumas funções dependem de parâmetros fornecidos por outras funções que os requisitos de conformidade, existem implicitamente, para uma API particular.

3.1.2. OS SERVIÇOS DE MMS

O draft 6 do Projeto EIA [16] descreve vários serviços fornecidos pelo MMS.

Neste trabalho mencionam-se, através de uma tabela as diversas classes de serviços MMS e descrevem-se especialmente os serviços de acesso a variáveis, que são os serviços escolhidos para efeito de implementação da API-MMS, por estarem frequentemente presentes num sistema de manufatura. A tabela 3.2 relaciona as classes de serviços, serviços permitidos e um pequeno comentário sobre os serviços pertencentes à respectiva classe [L1].

CLASSE DE SERVIÇO	SERVIÇOS	COMENTÁRIOS
Gerenciamento de Contexto	Iniciate Conclude Abort Cancel Reject	Esses serviços, em colaboração com o ACSE, permitem iniciar, terminar e abortar a comunicação c/ o usuário MMS-par, cancelar pedidos de serviços e receber notificação de erros de protocolos
Suporte de VMD	Status Unsolicited Status Get Name List Identify Rename	Operam sobre o VMD. Fornecem status e outras informações sobre os objetos do VMD. Permite renomear um objeto
Gerenciamento de Domínio	Iniciate Download Sequence Down Load Sequence Terminate Download sequence Iniciate Upload Sequence Upload Sequence Terminate Upload Sequence Request Domain Download Request Domain Upload Load Domain Content Store Domain Content Delete Domain Get Domain Attributes Obatin File Alter Event Enrollment Event Notification Acknowledge Event Notification Get Alarm Summary Get Alarm Enrollment Summary Attach to Event Condition Modifier	Esses serviços permitem manipular domínios: atribuir-lhes recursos específicos (parte dos recursos total do VMD); prover-lhe opção de compartilhamento numa comunicação multi-usuários, onde existem invocações de programas simultâneas; obter seu estado (*LOADING*, *READY*, *IN-USE*, *NON-EXISTENT*, etc.).
Gerenciamento de Jornal	Read Journal Write Journal Initialize Journal Report Journal Status	Esses serviços dizem respeito a recuperação e armazenagem de informações vinculadas a ocorrência de eventos num VMD

CLASSE DE SERVIÇO	SERVIÇOS	COMENTÁRIOS
Gerenciamento de Invocação de Programa	Create Program Invocation Delete Program Invocation Start Stop Resume Reset Kill Get Program Invocation Attributes	Permitem criar e deletar invocações de programas no VMD; Após criados, mudar seus estados, que são: *IDLE, STARTING, RUNNING, STOPPING, STOPPED, RESUMING E UNRUNNABLE*
Acesso a Variáveis	Read Write Information Report Get Variable Acess Attributes Define Named Variable Define Scattered Access Get Scattered Access Get Scattered Access Attributes Define Named Variable Define Scattered Access Get Scattered Access Attributes Delete Variable Access Define named Variable List Get Named Variable List Define Named Type Get Named Type Attributes Delete Name Type	Esses serviços permitem operações sobre objetos que descrevem variáveis existentes no VMD
Gerenciamento de Semáforos	Take Control Relinsh Control Define Semaphoro Delete Semaphoro Report Semaphoro Status Report Pool Semaphoro Status Report Semaphoro Entry Status	Engloba serviços de sincronização de aplicações MMS, de forma a controlar e coordenar acessos a recursos de VMD compartilhados pelos APs
Comunicação de Operadores	Input Ouput	São serviços de comunicação do VMD com estação operadora para entrada e saída de dados

CLASSE DE SERVIÇO	SERVIÇOS	COMENTÁRIOS
Gerenciamento de Eventos	Define Event Condition Delete Event Contion Get Event Contition Attributes Report Event Condition Status Alter Event Conditon Monitoring Trigger Event Define Event Action Delete Event Action Get Event Action Attributes Report Event Action Status Define Event Enrollment Delete Event Enrollment Get Event Enrollment Report Event Enrollment Status	No VMD, podem ser incluídos objetos que descrevem eventos. Esses Objetos são manipuláveis pelo AP de forma a permitir-lhe gerenciar eventos. Envolvem informação de eventos, execução de serviços MMS sob ocorrências de eventos, notificação a usuários sobre transições de eventos, etc.
Gerenciamento de Arquivos	File Open File Read File Close File Rename File Delete File Directory	Permitem a manipulação de arquivos virtuais vinculados a um VMD. Os arquivos podem conter: programas e/ou dados

Tabela 3.2 - Os serviços MMS

3.1.2.1. OS SERVIÇOS DE ACESSO A VARIÁVEIS

O modelo do VMD pode ser extendido para revelar a existência de objetos que descrevam variáveis tipadas associadas à operação do VMD. Os serviços de acesso a variáveis são utilizados para criar, manipular e deletar esses objetos que embutem uma abstração das variáveis, sua sintaxe, sua faixa de valores possíveis e sua representação física durante seu tempo de vida. Descrevem o mapeamento entre os objetos virtuais (variáveis abstratas) e as variáveis reais, vinculadas à implementação concreta do ambiente.

Os objetos-variáveis ou variáveis MMS representam as variáveis reais do VMD sob dois níveis distintos de abstração:

objetos do tipo variável nomeada e objetos do tipo variável não-nomeada.

A abstração representada pelo primeiro está relacionada com a aplicação que acessa e manipula a variável real. Sua existência corresponde à existência (tempo de vida) da aplicação, do VMD ou do domínio a qual pertence. O acesso a elas é feito usando um nome determinado pela aplicação (seu endereço não necessariamente deve ser conhecido).

Já a abstração representada pelos objetos do tipo variável não-nomeada é mais próxima dos aspectos dependentes da implementação do sistema real. Seu acesso se faz usando endereços específicos do VMD, portanto sua existência depende da existência do dispositivo ou do VMD.

Sob essas representações abstratas, a manipulação de variáveis existentes no servidor (VMD) torna-se possível a qualquer cliente MMS, mesmo quando existe uma grande diferença de implementação entre os dois ambientes, seja devido a idades ou complexidade.

O MMS fornece serviços que operam sobre os objetos de acesso a variáveis que:

- permitem a definição de objetos do tipo nomeado em termos de objetos do tipo não-nomeado,
- auxiliam na manipulação de objetos complexos a partir de objetos elementares,
- providenciam acesso a múltiplas variáveis MMS dispersas, na forma de uma única lista ou estrutura e
- permitem acesso a elementos individuais que compõem estruturas mais complexas (acesso parcial).

Essas facilidades constituem no modelo do VMD a cinco (5) objetos de acesso a variáveis sobre os quais os serviços de acesso operam:

1- objeto variável-nomeada,
2- objeto variável não-nomeada,

3- objeto acesso disperso,
4- objeto lista nomeada de variáveis
5- objeto tipo nomeado.

A API-MMS, por sua vez, dispõe de algumas funções auxiliares para o tratamento de acesso a variáveis. O cliente MMS serve-se destas funções antes (criação ou montagem) e depois (interpretação) de uma chamada de função que corresponda a um serviço MMS propriamente dito (função principal). Por exemplo, o usuário cliente-MMS pode fazer várias chamadas à função auxiliar da API **"new-type"** para construir um tipo MMS complexo e depois do tipo já construído pedir um serviço MMS **"Define Named Type"** que cria um objeto tipo-nomeado no VMD, e a partir daí a descrição desse tipo MMS pode ser referenciado através de um nome, por qualquer usuário existente na rede.

No próximo ítem, será visto o modelo da API-MMS, suas funções e requisitos ante o modelo MMS aqui descrito.

3.2. A API-MMS

O modelo da API-MMS [M5] incorpora um modelo de interação entre usuários MMS (MMS-requisitante e MMS-respondedor), com o objetivo de troca de informações para posterior tratamento das mesmas de forma cooperante. Os usuários MMS são os processos aplicativos implementados, localmente ou remotamente, nos diversos dispositivos programáveis. Deve-se permitir que os serviços MMS, disponíveis aos processos aplicativos, possam residir num processo separado daquele que o solicita. Em inúmeras aplicações é importante que se execute serviços remotamente (carga de programa, gerenciamento de eventos, ou simplesmente obter informações periódicas de jornal) o que torna necessário um suporte de sistema operacional que proveja um sistema multi-tarefa para comunicação.

A API entra com um papel contributivo muito importante nesse ambiente: ela simplifica as etapas de estabelecimento de comunicação; interfaceia-se entre o usuário e o sistema operacional local; desempenha funções auxiliares que diminuem a necessidade de conhecimento, pelo usuário, da complexa máquina que constitui um

protocolo de aplicação; e formata para o usuário, etapa por etapa, as primitivas de serviço que ele precisa utilizar.

É importante observar que o usuário dos serviços MMS deve conhecer todo o funcionamento do sistema de comunicação e os tipos de mensagens que trafegam entre processos aplicativos, bem como ter em mente o modelo virtual de um dispositivo programável que fará parte da interação inter-processo, para poder então utilizar (solicitanto ou respondendo) os serviços que se façam necessários.

A API-MMS simplemente coloca à disposição do usuário as funções que ele necessitará, de forma padronizada, tornando mais amigável a utilização do protocolo MMS. No AP estarão todas as operações (controles, regras, restrições) associadas a aplicação dentro do universo de heterogeneidade existente num ambiente aberto real.

3.2.1. O MMS COMO PARTE DE UMA AE

A ISO define uma (N)-Entidade como um elemento ativo dentro de um (N)-subsistema que englobe um conjunto de capacidades definidas para a (N)-camada e a qual é identificada sem ambiguidades.

Na camada de aplicação existem vários ASE's associados a AE's que serão utilizados convenientemente pelo AP para realizar sua aplicação. Estando ativo, o programa de aplicação deve utilizar pelo menos o ACSE. No caso de usar o MMS, as chamadas de funções da API-MMS fazem parte de uma AE invocada previamente pela AP.

O contexto sob o qual associações de AE devem ser mantidas, é negociado através de chamadas de funções específicas da API. A API mapeia os parâmetros dessas funções (gerencimento de conexão) nos serviços do MMS responsáveis pelo gerenciamento de contexto de aplicação.

Na versão atual do MAP (3.0), num contexto de aplicação pode-se ter apenas um ASE ativo num determinado tempo. O nome do contexto (ou do ASE) é passado como parâmetro do pedido de associação. Durante a fase de negociação é feita a verificação da legitimidade das informações específicas do ASE considerando suas capacidades e requisitos. Para o MMS, essas informações tratam de diversos

aspectos como: tamanho de mensagens permitidas para cada uma das partes comunicantes; capacidade de envio de pedidos, tanto de Request de Initiate, como de outros serviços subsequentes; e capacidade de suporte a tamanhos de mensagens de Responses positivos. Também deverá ser considerada sua capacidade de armazenar serviços pendentes de selecionar o conjunto dos serviços que poderão ser utilizados naquela associação, através da negociação de CBBs.

A AE passará a representar, a partir daí, os serviços MMS disponíveis naquele AP.

3.2.2. AS ESTRUTURAS DAS FUNÇÕES DA API-MMS

A chamada de funções da biblioteca da API-MMS cairá em 4 tipos de funções desempenhadas pela interface:

1º as de gerenciamento de conexão (sempre existente em qualquer API-MMS),
2º as de suporte da API (interface entre AP e SO)
3º as de suporte aos serviços MMS
4º as dos serviços MMS

A API-MMS segue a estrutura de definições de funções do modelo genérico já apresentado.

Os parâmetros dessas funções, os quais carregam as informações do usuário, estão dispostos de maneira padronizada. Os parâmetros obrigatórios de entrada aparecem sempre antes dos não obrigatórios, e estes, tanto obrigatórios como não-obrigatórios, sempre precedem os parâmetros de saída.

O formato das chamadas de funções é obtido com a utilização dos DCB's (Bloco de Controle de Dados). A utilização de DCB's visa versões futuras, que podem vir a expandir o número de parâmetros opcionais ou condicionais. Colocando-os dentro desses DCB's, a padronização da chamada é conservada, pois a aparência externa da chamada não irá mudar e, portanto, a interface não sofrerá nenhuma modificação nas suas declarações de funções.

O formato das funções que constam na biblioteca da API segue

esse padrão:

Nome_função (parâmetros_de_entrada_explícito, DCB_de_entrada, DCB_de_saída, parâmetros_de_saída_explícitos).

Tomando como exemplo a função de pedido de associação da API, e de leitura de variável tem-se:

MM_CONNECT (ae_label, return_event_name, called_ae_name, *connect_i_dcb*, *connect_o_dcb*, connection_id)
MM_VREAD (connection_id, return_event_name, *read_i_dcb*, *read_o_dcb*)[1]

Os parâmetros de entrada explícitos que antecedem os de entrada implícitos, estão inerentemente relacionados com o usuário e são certamente verificados e manipulados. Já os de entrada, contidos no DCB, não são óbvios ao usuário. Eles podem variar de chamada a chamada e seus valores, os quais variam frequentemente, vão revelar sua funcionalidade.

Para ficar claro que tipo de parâmetro os DCB contém explicam-se alguns argumentos do DCB de entrada da função CONNECT:

Input DCB:

CALLED_PRESENTATION_ADDRESS	(pode estar ausente e seu valor será provido pelo serviço de diretório, através do "called_ae_name", que é obrigatório)
NUMBER_OF_RETRY	(é variável . O usuário pede para tentar a requisicão de "connect" n vezes. Zero é o default)
CONTEXT_NAME	(vai depender de qual ASE está envolvido no pedido de associação)
ASE_SPECIFIC_ASSO_REQ_INFORMATION	(não se sabe à priori que estrutura esse parâmetro. Ela vai diferir consideravelmente de ASE para ASE)

[1] Os parâmetros em negrito são do tipo obrigatório.

Já o parâmetro connection_id é um parâmetro explícito e tem um significado óbvio para o usuário da chamada da função: na função CONNECT ele espera uma identificação daquela associação que está solicitando e na função VREAD esse identificador diz a que associação esse pedido de serviço pertence.

Para concluir o assunto sobre funções da API-MMS, apresenta-se, de forma sintética, uma tabela contendo:

- os vários tipos de funções existentes na biblioteca relacionados aos serviços prestados pela interface,
- exemplos de funções de cada tipo.

Dentre as várias "funções principais de Serviços MMS", devido à grande quantidade de funções, relacionam-se apenas algumas, separando-as por classe de serviço MMS a que pertencem (tabela 3.3).

TIPO DE FUNÇÃO	*NOME DA FUNÇÃO*
Funções de Gerenciamento de Conexão	mm_aeactivation mm_deactivation mm_connect :
Funções de Suporte a API	mm_vait mm_note mm_dinitializeDCB :
Funções principais de Servicos MMS	mm_status mm_get_name_list mm_identify : mm_create_program_invocation mm_start mm_stop : mm_vread mm_vwrite mm_ireport mm_dnvariable :
Funções auxiliares de Servicos MMS	mm_nrestriciton mm_ntype mm_npartialaccess :

Tabela 3.3 - As funções da API-MMS

Para fins de implementação, foi dado enfoque às funções relativas aos serviços de acesso a variáveis que serão discutidos no capítulo 4, abrangendo, desta lista, todas as funções auxiliares referentes a esses referidos serviços e as três principais de acesso a variáveis: Read, Write, Information Report.

3.2.3. OS TIPOS DE DADOS DA API-MMS

Um AP fará referências e manipulará ao longo de sua execução, os diversos tipos que descrevem os objetos MMS.

Esses tipos não devem ser dependentes de nenhuma linguagem de alto nível particular, já que devem manter uma abstração de modo que possam ser implementados em qualquer linguagem, sob qualquer suporte de S.O. (tipos estruturados, cadeias de caractres e vetores devem ser suportados de alguma forma, independente da linguagem usada na programação da interface).

A API define certos tipos de dados e utiliza outros já definidos e exportados (via arquivo INCLUDE) da aplicação em questão, no caso MMS.

Prossegue-se agora mostrando os tipos que foram definidos no MMS, mas antes citam-se as regras observadas na especificação quanto às nomeações de tipos de dados.

Não há nenhuma convenção particular para nomear parâmetros, contudo, dentro de uma aplicação, os parâmetros com a mesma semântica e mesma sintaxe, devem ter o mesmo nome. Esses mesmos nomes devem ocorrer durante toda a aplicação.

Os tipos de cada DCB devem ser identificados de forma a representar se são de entrada ou se são de entrada/saída. Então a convenção inclui um delimitador que distingue os dois tipos: são as letras "i" (para entrada) e "o", (para entrada/saída).

Como a API é genérica, as regras para definição de nomes de funções, de nomes de primitivas, de erros e de DCBs seguem mais ou menos o mesmo estilo. Usa-se nesses nomes duas letras identificadoras que se referem ao elemento de serviço associado: FT para FTAM, MM para MMS, JT para JTM e assim por diante.

Quanto aos nomes de função, por causa das diferenças de linguagens de programação, eles se baseiam no padrão definido para aquela linguagem adotada. Cada linguagem possui um padrão de nomes. Assim, em todas as bibliotecas de funções das interfaces específicas da linguagem de programação, o nome de uma determinada função tem sua origem comum no padrão de nome assinalado àquela função. Os nomes devem ser únicos em seus 6 primeiros caracteres e devem ter a

seguinte forma:

ssdxxx[x]*

ss ⟶ identificador do serviço

d ⟶ delimitador (depende da linguagem)

xxx ⟶ parte única do nome

[x]* ⟶ parte complementar do nome (não precisa ser única)

Exemplo: mm_dvname (Define Variable Name)

As regras gerais seguidas para elaborar o "Binding" para a linguagem C estão baseadas no padrão proposto pelo Draft ANSI C Language e nas recomendações da especificação do modelo genérico de interface. [M2].

Tipos Primitivos de dados API-MMS

As definições de tipos de dados primitivos da interface API-MMS devem corresponder às seguintes definições, mostradas através da tabela 3.4.

Nome do Tipo	Descrição	Comentário
uint 8	inteiro não sinalizado	Faixa: 0 a $2^8 - 1$
uint 16	inteiro não sinalizado de 16 bits	Faixa: 0 a $2^{16} - 1$
uint 32	inteiro não sinalizado de 32 bits	Faixa: 0 a $2^{32} - 1$
sint 8	inteiro sinalizado de 8 bits	Faixa: (2^7) a $(2^7 - 1)$
sint 16	inteiro sinalizado de 16 bits	Faixa: $-(2^{15})$ a $(2^{15}-1)$
sint 32	inteiro sinalizado de 32 bits	Faixa: $-(2^{31})$ a $(2^{31}-1)$
bool 32	valor logico: TRUE ou FALSE	FALSE = 0 , TRUE <> 0
octet	string de 8 bits(byte)	Tipo Char

Tabela 3.4 - Tipos Primitivos da API-MMS

Tipos dos Parâmetros Universais

Os parâmetros comuns a quase todas as chamadas de funções são: connection_id, return_event_name e return_code.

1. **Connection_id:**

   Identificador da associação da aplicação sob a qual os serviços MMS serão solicitados.

   Tipo: uint 16

2. **Return_event_name:**

   parâmetro de entrada das funções que têm capacidade de operar síncrona e assincronamente. Se a chamada for feita de maneira síncrona, esse valor é zero.

   Tipo: EMTLEN (nome local dado a um evento)

   Obs: EMTLEN é o tipo uint32

3. **Return_code**

   Valor de retorno de uma chamada de função.

   Tipo: sint 32

   Obs: Nas funções que só podem ser chamadas como síncronas, esse parâmetro não existe na chamada. É o caso das funções auxiliares da API-MMS. No caso, o valor da unção deve ser transformado (cashing) para o tipo Return_code.

   Ex: Return_code mm_ntype (tipe_tag, new_type, inout_dcb)

Outros Parâmetros

Como foi dito, o parâmetro return_code representa o diagnóstico do provedor de serviço e indica os erros gerais e específicos de um serviço quando esse é submetido à rede. Quando o return_code indicar um erro específico do serviço (listados na cláusula 17 do documento MMS [16]), um outro parâmetro de saída do usuário, normalmente embutido no DCB de saída, conterá informações

adicionais acerca do erro. Esse parâmetro é do tipo **Error_block** (Error_type, no draft 6). É um tipo estruturado e sua representação em ASN.1[2] encontra-se no documento MMS [17].

A título de ilustração, descreve-se a estrutura do tipo Error_block em ASN.1, que, por razões de simplicidade, está abreviada:

```
Error_block ::= SEQUENCE
    {
      error class  CHOICE
           {
              "error codes"     IMPLICIT INTEGER
                 .
                 .
                 .
           }
   additionalDescription          Printable String     OPTIONAL
   ServiceSpecificInformation     CHOICE
           {
                 .
                 .
                 .
           } OPTIONAL
    }
```

Além do tipo "Error_block", o MMS tem outros tipos comuns que são referenciados em vários serviços. Já foram mostrados os tipos inteiros a que o MMS se refere como Integer 8, Integer 16, Integer 32, Unsigned 8, Unsigned 16 e Unsigned 32.

Há ainda o tipo **Identifier** que é definido como sendo do tipo **Visible String**, que por sua vez é uma cadeia de caracteres.

Outro tipo bastante usado é o **Object_Name**. No MMS ele está representado em ASN.1, da seguinte forma:

---

[2] A especificação da ASN.1 e as regras de codificação estão descritas em [18] e [19], respectivamente.

```
Object Name ::=  CHOICE
      { vmd_specific          [0] IMPLICIT  Identifier,
        domain_specific       [1] IMPLICIT SEQUENCE
              {
                domainID      Identifier,
                itemID        Identifier
              }
        aa_specific           [2] IMPLICT Identifer
       }
```

Por fim, há um outro tipo de dado muito usado nos serviços MMS que é o tipo **Address**.

O tipo Address, em ASN.1, está no seguinte formato:

```
Address ::=  CHOICE
     {
       numeric_address      [0] IMPLICIT Integer 32;
       symbolic_address     [1] IMPLICIT Visible String;
       unconstrained        [2] IMPLICIT OctetString;
     }
```

Uma outra forma de se abstrair das particularidades dos tipos, consiste na sua definição como tipos exportados. Os tipos exportados usados na API são fornecidos pelo projetista da aplicação, via arquivos (por exemplo arquivos INCLUDE) em forma de caixa-preta.

A API declara esses tipos e os manipula seguindo algumas regras, que restringem as operações permitidas sobre esses tipos. O fato é que eles já são predefinidos e as regras de compatibilidade devem ser mantidas ao manipulá-los.

Que variáveis têm esses tipos?

No MMS existem variáveis complexas e que são representadas nas diversas formas (arranjos ou listas, estruturas simples, arranjos de estruturas, estruturas auto-referenciadas ou recursivas.

Com o objetivo de obter uma maneira elegante de manusear parâmetros recursivos sem definir uma implementação específica, definiram-se alguns tipos que são referências para os tipos

pré-definidos (exportados).

O valor da variável de um tipo desses é, na verdade, uma referência a uma estrutura de dados mais complicada dentro da interface, ou seja, é um ponteiro para a estrutura.

Na API-MMS, esses tipos são referenciados nas funções auxiliares, que dão suporte ao usuário para construir tipos mais complexos e, no retorno, interpretá-los.

Por exemplo, o serviço de acesso a variáveis necessita desse suporte da API. Para ilustrar, a tabela 3.5 mostra os tipos MMS, o nome do parâmetro corresponde atribuído pela especificação MMS [16] e as funções auxiliares da API-MMS que as utilizam, ou como parâmetro de entrada ou como parâmetro de saída.

TIPO	PARÂMETRO MMS	FUNÇÕES AUXILIARES
MMS_component	Structure	New component Interpret component New type Interpret type Make list Extract list element
MMS_restriction	Select_Access	New restriction Interpret restriction New partial access Interpret partial access
MMS_partialaccess	Alternate Access	New partial Access Interpret partial access Make list Extract list element New scattered access Interpret scattered access New variable specification Interpret variable specification
MMS_list	Structure Alternate Acces Scattered Access Discription Data Access Result Write Result	Make list Extract element liste (Obs: não se menciona as funções que manipulam as listas com elementos desses tipos, pois já foram citadas nos respectivos tipos)
MMS_type	Type Specification	New type Interpret type New component Interpret component New variable specification Interpret variable specification New association Interperet association
MMS_varspecification	Variable Specification	New variable specification Interpret variable specification New scattered access Interpret scattered access New variable access Interpret variable access

TIPO	PARÂMETRO MMS	FUNÇÕES AUXILIARES
MMS_vaccessspeci-tion	Variable access specification	New variable access Interpret variable access New association Interpret association
MMS_scattaccess	Scattered access description	New scattered access Interpret scattered access Make list Extract list element New variable specification Interpret variable specification
MMS_data	Data	New data Interpret data Make list Extract list element New association Interpret association

Tabela 3.5 - Relação dos tipos MMS dos Serviços de Acesso a Variáveis, nome do parâmetro MMS correspondente a funções auxiliares que os utilizam.

3.3. O SISDI-MAP

3.3.1. OBJETIVO DO SISDI-MAP

O SISDI-MAP (Sistema Didático-MAP) é um projeto com fins didáticos que incorpora protocolos de aplicação, interfaces de aplicação, processos aplicativos e uma interface de operação d sistema e será utilizado como ferramenta de estudo e observaçã comportamento de uma rede local tipo MAP [A1].

O sistema foi especificado de forma conjunta r SISDI-MAP e, atualmente, está definido para suportar o r Juntamente com o ACSE,para a aplicação de comur Posteriormente ele virá a suportar outros protor

como por exemplo, o FTAM, o RDA, o ROSE, bem como o NM, o DS, e outras interfaces para essas aplicações.

O SISDI-MAP foi estruturado de forma hierárquica, seguindo o modelo ISO, e portanto permitirá sua expansão gradativa na incorporação dos níveis inferiores de protocolo (figura 3.4), à medida que forem sendo implementados, sendo a simulação transferida para o próximo nível adjacente inferior (N-1). Na versão atual, a simulação encontra-se na camada 6 (Proc-Presentation). Os protocolos de Apresentação e de Sessão encontram-se em fase de implementação.

O sistema conta com o suporte de um núcleo tempo real que se caracteriza como um sistema multi-tarefa, permitindo realizar aplicações reais de tempo-real.

3.3.2. PRINCÍPIOS DA ESPECIFICAÇÃO DO SISDI-MAP

Cada um dos componentes do sistema (protocolo MMS, protocolo ACSE, interface API-MMS, processos aplicativos, simulador nível 6 e programas aplicativos) foi especificado de forma a constituírem módulos autônomos (processos), com suas funcionalidades distintas, seus recursos independentes (filas, portas de comunicação, etc) e numa estrutura de operação multi-usuário.

A comunicação entre as tarefas se deve dar via portas unidirecionais dedicadas. O controle das tarefas e sua intercomunicação são efetuados pelo núcleo [T2]. O núcleo fornece primitivas para criação e escalonamento dos processos, criação das portas de comunicação, tratamento de sincronização e comunicação entre processos e gerenciamento de memória (cada processo tem sua própria área de memória).

3.3.3. PRINCÍPIOS DE OPERAÇÃO DO SISDI-MAP

Uma interface de operação foi definida para permitir a utilização de forma amigável por parte do operador, através de "menus", janelas e gráficos [T6]. Ela será responsável pela criação das tarefas e provê o gerenciamento do ambiente durante sua execução.

Sinais de controle serão emitidos pelos diversos processos de forma a permitir que o usuário do sistema obtenha informações sobre as diversas etapas do processamento de um serviço, bem como a introdução de erros de comunicação, provendo situações de exceção.

A passagem de informação entre os processos, que na especificação ISO é efetuada através de primitivas de serviços associados aos protocolos, no SISDI-MAP será implementada através de um buffer ou bloco de memória que contêm os parâmetros necessários a serem passados e um endereço para esse bloco é colocado na porta de comunicação especificada para essa comunicação.

Assim, o formato de mensagens é único em cada via (porta) e deve estar dentro do limite máximo especificado nessa comunicação particular. Isso permite um controle a nível de sistema (através de definições de valores máximos) dos tamanhos de mensagens (PCI + SDU) entre camadas.

Outra vantagem da passagem de informação entre processos via buffer é a minimização de cópias de dados entre as camadas. A concatenação das áreas de memória através de ponteiros, formará um bloco único contendo a mensagem completa [L4].

A colocação e retirada de mensagens deve seguir modelo FIFO. A figura 3.4 [A1] permite visualização clara do modelo de comunicação no SISDI-MAP. Foi incorporada à figura a representação dos conceitos de endereçamento e nomeações da ISO [I2].

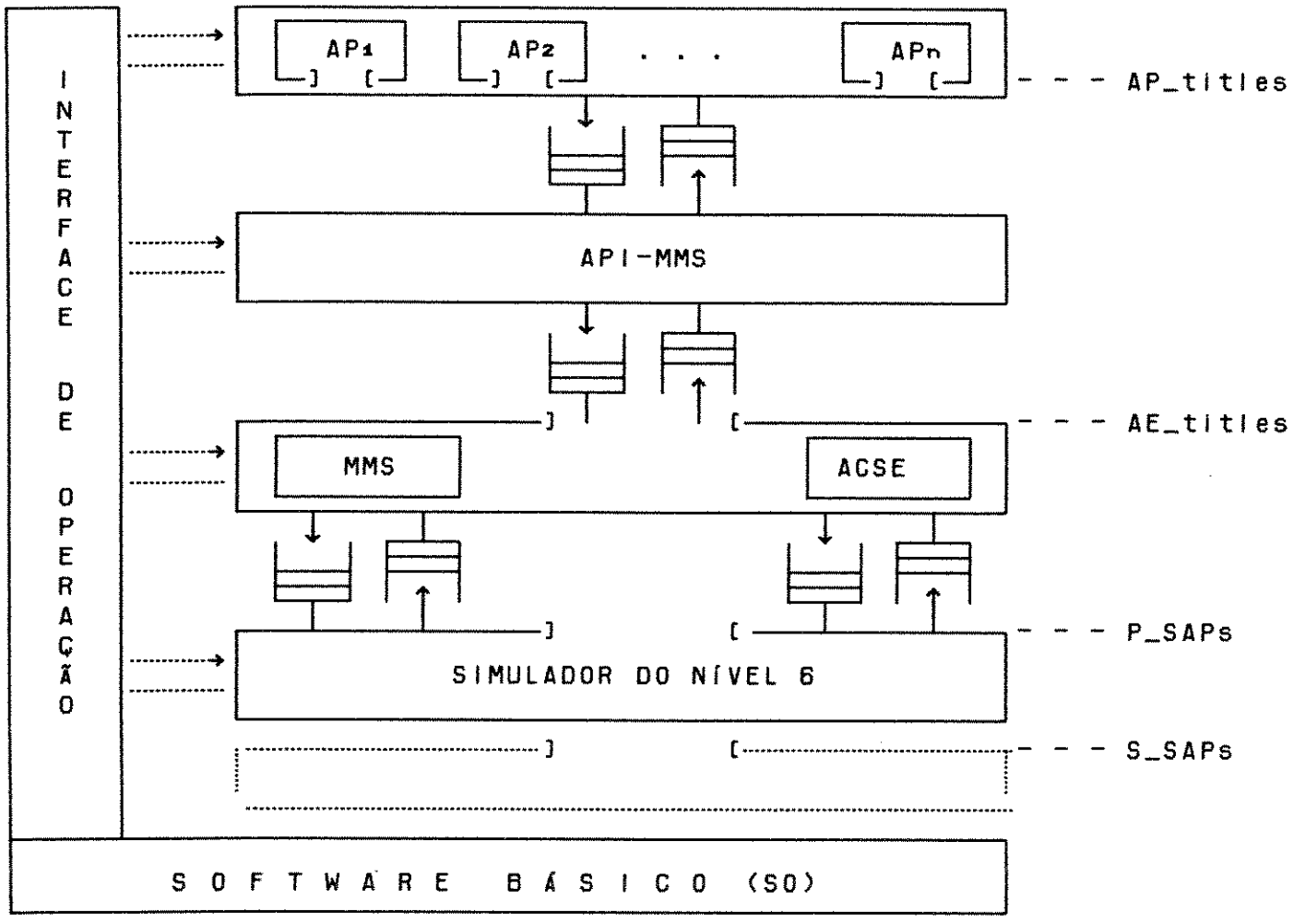


Figura 3.4 - Comunicação entre processos no SISDI-MAP

3.4. CONCLUSÕES

Segundo [L7], a especificação de um sistema deve atender aos requisitos e objetivo do sistema.

O SISDI-MAP foi especificado de forma a manter coerência com o modelo hierárquico ISO/OSI. Os processos que compõem o sistema tem suas funcionalidades independentes e estão dispostos hierarquicamente. A comunicação entre eles se dá através de mensagens que, associadas às portas de comunicação, traduzem o conceito de SAP's.

Observa-se na especificação da API-MMS a compatibilidade com o modelo geral da interface de aplicação e, ao mesmo tempo, ao modelo do sistema de comunicação MMS.

São especificadas as funções de cada módulo componente do modelo, bem como o fluxo de informação entre eles.

No entanto, na versão atual, a especificação da API-MMS restringe-se a uma interface mestre, essencialmente requisitante. Futuras versões deverão surgir para atender às especificações de um servidor-MMS, ou seja, de uma API essencialmente respondedora.

As funções da interface neste caso estariam mais voltadas ao modelamento de um VMD, ao controle de acesso compartilhado e às operações atômicas sobre objetos compartilhados.

CAPÍTULO 4

ASPECTOS DE IMPLEMENTAÇÃO DA API-MMS NO SISDI-MAP

A especificação atual da API-MMS está dirigida para o MMS - draft 5 [I4] e [I5]. Devido a abstração e a falta de clareza em ambas especificações foram encontradas muitas dificuldades nas etapas de desenvolvimento da implementação.

Foi elaborado o primeiro esboço de algumas funções da API-MMS para a versão do MMS-draft 5 para a qual foi especificada, e suas respectivas estruturas de dados foram definidas [A3].

Com o surgimento do projeto SISDI-MAP [A1], onde o protocolo MMS contava com sua nova versão - draft 6 [I6] e [I7], um esforço no sentido de compatibilizar as versões diferentes de MMS e API, passou a ser primordial.

A estrutura de dados sofreu alterações, principalmente devido ao surgimento, na versão atual do MMS, de um novos conceitos para o MMS, dirigido ao modelamento de objetos, classes de objetos, operações sobre objetos e que substituiu o anterior voltado a um modelo procedural.

Para exemplificar as alterações sofridas nas especificações de argumentos para um serviço de leitura de variável, vale comparar as sintaxes abstratas das primitivas read_request de ambas especificações:

Draft 5:

```
ReadRequest ::= CHOICE
        {
          singleAccess        Variable Access,
          scatteredAccess     [5] IMPLICIT SEQUENCE OF VariableAccess
        }
```

Draft 6:

```
ReadRequest ::= SEQUENCE
        {
          specificationWithResult          [0] IMPLICIT BOOLEAN
                                               DEFAULT FALSE,
          VariableAccessSpecification      [1] VariableAccessSpecification
        }
```

Nesse capítulo acompanha-se a trajetória da implementação, mostram-se as adaptações sofridas nos parâmetros e estruturas de dados e compara-se, sempre que possível, com a versão anterior da API para o MMS draft 5.

4.1. ESCOPO DE IMPLEMENTAÇÃO

As funções definidas para a API-MMS,na versão atual, foram orientadas para abranger apenas um **subconjunto dos serviços MMS**. As chamadas de funções da API têm sua relevância na utilização dos serviços MMS por parte dos programas de aplicação (AP's). Estes devem fazer uso coerente das funções dentro do contexto e objetivo de sua aplicação.

Um exemplo de utilização do sistema de comunicação, num ambiente fabril, é mostrado em [A1].

Detalhando um pouco mais a seqüência de funções necessárias a uma tarefa particular, citam-se algumas funções principais a serem utilizadas.

- ativação da entidade de aplicação (APPL. ENTITY ACTIVATION)
- estabelecimento de associação com outra AE ativa (CONNECT)
- obtenção de identificação (IDENTIFY)

  Obs: na identificação estão expressos todos os recursos associados ao dispositivo
- carga de um programa num dispositivo (FILE DOWNLOAD)
- execução do programa (START)
- leitura de uma variável (READ VARIABLE)
- deleção de uma variável (DELETE VARIABLE NAME)
- término de associação (CONCLUDE)
- desativação da Entidade de Aplicação (AE DEACTIVATION)

Diante dessa aplicação percebe-se que há uma seqüência coerente de operações: primeiro, uma AE deve estar ativa antes de negociar associações; depois um VMD deve ser devidamente ativado e identificado para posteriores operações; em seguida, um programa deve ser carregado antes de ser executado; posteriormente objetos devem ser

definidos e identificados antes de serem manipulados e assim por diante. As funções de "fechamento" também só terão validades caso tenha havido a função correspondente de "abertura" das operações. EX: um DELETE só pode ser aplicado num objeto existente; um término qualquer de operação é aplicável a uma operação ativa.

Com o objetivo de definir o **escopo de implementação** desse trabalho, são analisados agora alguns aspectos do projeto de implementação, mostrando as limitações e as considerações adotadas.

4.1.1. SUBCONJUNTO DE SERVIÇOS MMS

Em sua globalidade, os serviços MMS permitem a comunicação e a interoperação dos diversos dispositivos virtuais (VMD's) existentes num ambiente fabril automatizado, gerenciando contextos de associações, provendo suporte aos VMD's, seus domínios e capacidades, gerenciando carga e invocações de programas, acesso a variáveis, gerenciando eventos e semáforos e até comunicação com operador e gerenciamento de "logs" ou jornais.

No entanto, em ambientes reais, nem sempre todos esses serviços são suportados em todas as estações. Por sua vez a especificação da API em seu estágio atual define apenas um subconjunto dos serviços MMS.

Para estabelecer o escopo de implementação da API, definiu-se inicialmente o conjunto dos serviços de gerenciamento de contexto [T1] e um subconjunto dos serviços de acesso a variáveis, implementado neste trabalho.

A modularidade da API permitirá facilmente o acréscimo de novas funções à medida que forem sendo implementadas.

4.1.2. FUNÇÕES REQUISITANTES X FUNÇÕES RESPONDEDORAS

Além da restrição acima, a interface especificada em [M5] é dirigida a aplicações mestres com atividades de controle e monitoração, geralmente residentes num computador hospedeiro, ou seja, interfaces para aplicações tipo "Cliente" que consistam essencialmente

de funções requisitantes.

As funções respondedoras são tratadas por esse tipo de interface de maneira indireta, restringindo-se a expô-las ao usuário.

Essas limitações conduziram a uma visualização não muito clara do papel da API no tratamento das funções, uma vez que a especificação só permite a visão unilateral da comunicação. Há uma requisição, porém não se sabe como é o procedimento dual, de resposta.

Se a interface não estiver especificada para responder automaticamente às indicações recebidas, ela terá que prover as funções respondedoras das quais o usuário poderá dispor ao receber "INDICATION RECEIVE's autorizados. Em suma, o módulo RS da Biblioteca de Funções da API está "vazio".

4.1.3. PROCEDIMENTOS DE MAPEAMENTO DOS SERVIÇOS CONFIRMADOS

Enfrentou-se dificuldades em implementar omapeamento da primitiva de confirmação nos parâmetros de saída do pedido de serviço do usuário (AP). Isso foi devido também à visão unilateral da interface. Não se dispunha de uma especificação da API-MMS do lado respondedor, recebendo indicações de serviços confirmados e respondendo positiva ou negativamente.

O módulo funcional CSP (Confirmed Service Provider) descrito no capítulo 2, teria sua funcionalidade reduzida caso não se considerasse o recebimento de confirmações, advindas das primitidas de "response" emitidas pela API par. Apesar da atual versão da API não especificar os procedimentos quanto à chegada de primitivas de confirmação, julgou-se necessário obter informações a partir do documento MMS e completar os referidos procedimentos.

Portanto tornou-se viável acrescentar à funcionalidade da API o tratamento de confirmações dos serviços solicitados.

4.1.4. TRATAMENTO DE INDICAÇÕES

Quanto ao tratamento de indicações, a especificação delimita o recebimento de indicações a apenas alguns tipos: ABORT,

REJECT, CONCLUDE, CANCEL, UNSOLICITED STATUS E INFORMATION REPORT, atendendo os requisitos essenciais de uma interface "mestre".

Observa-se que o IFA está especificado para realizar uma filtragem nas primitivas de indicações que venham da máquina de protocolo MMS, porém o tratamento das indicações filtradas é deixado em aberto. Surgiu a dúvida: Preenche-se essa lacuna?

O recebimento, por parte do usuário, de indicações por ele autorizadas pela chamada à função INDICATION RECEIVE ocorrem quando a interface não pode responder automaticamente. Para responder a um READ_indication a interface deveria contar com a função V-GET especificada no MMS-draft 6, e que requer informações adicionais de mapeamento do objeto MMS na representação local. A versão anterior do MMS não conta com esse tipo de facilidade e portanto a API afasta-se da possibilidade de realizar tal mapeamento. Quanto à primeira situação, para responder um READ_indication recebido através da função INDICATION RECEIVE, o usuário deve contar com a função respondedora READ RESPONSE contendo os parâmetros de resposta à indicação.

A API não especifica nenhuma função respondedora, na atual versão.

Considerou-se, portanto, que a interface não tem funcionalidade total para responder automaticamente às indicações de Read/Write, pois necessita de procedimentos para tratamento de indicações bem especificados. No capítulo 5 apresentam-se propostas para ampliar a funcionalidade da API-MMS neste sentido.

4.2. A IMPLEMENTAÇÃO DOS SERVIÇOS DE ACESSO A VARIÁVEIS

Como foi introduzido no capítulo anterior, o MMS define objetos representando abstrações de variáveis reais de um dispositivo e provê métodos de acesso a essas variáveis que podem ser utilizados pelas diversas aplicações existentes numa rede, desde que estejam associadas à aplicação local possuidora da variável. Os serviços MMS de acesso a variáveis são designados a atuar sobre as variáveis MMS, manipulando-as de forma conveniente. Tais serviços têm por

finalidade principal a realização de operações de leitura e escrita de variáveis. Para realizar uma operação de leitura, por exemplo, são necessárias informações que descrevam a variável, descrevam o método de acesso (se por endereço ou nome) e que associem à descrição os dados relativos à variável. Essas informações são devidamente utilizadas para converter a representação real (aplicação local) numa forma que possa ser transmitida e reconhecida pelas aplicações da rede.

A API fornece subsídios através de suas funções auxiliares na realização dessas atividades, tornando o papel do usuário menos árduo.

Já foram citadas essas funções, relacionando seus parâmetros, na tabela 3.5. Aqui apresentam-se com mais detalhes, mostrando-se as estruturas de dados envolvidas e o algoritmo da função, seguindo uma seqüência de implementação real.

4.2.1. AS FUNÇÕES AUXILIARES DE CONTRUÇÃO DE DEFINIÇÕES MMS

Com a finalidade de realizar um "Read request", diversas chamadas às funções auxiliares da API são feitas para se construir, passo a passo, a estrutura final de um dos parâmetros requeridos, no caso a especificação de acesso à variável (Var_access_specification), que transitará numa PDU até o usuário destino.

Dentre as diversas funções, pelo menos as que solicitam a construção de uma especificação de variável, a de especificação do acesso e a de definição de uma associação com endereços locais, devem ser chamadas antes de se concretizar uma chamada a uma função principal[1].

Entretanto, dependendo do tipo da variável, do método de acesso a ser utilizado e do tipo de associação a ser empregada, outras funções são utilizadas a critério do usuário, de um caso mais simples a um mais complexo.

Segue agora a descrição de cada uma das funções seguindo um mesmo estilo, para melhor documentação, contendo os seguintes ítens:

---

[1] Uma vez realizada qualquer uma das definições, esta pode ser reutilizada em pedidos de serviços posteriores.

1- Objetivo da função
2- Nome da função já no formato padronizado seguindo a especificação [M5]
3- Argumentos da função
4- Algoritmo em "C-like". Para efeito de clareza, para cada função, o parâmetro resultante terá um comentário ao lado onde se denomina o prefixo MMS e o nome do próprio parâmetro especificado no MMS-draft 6. O termo "handle" é usado para referenciar aos tipos exportados.
5- Observações são feitas oportunamente a cada descrição de função.
6- Comentários adicionais referem-se a considerações sobre a implementação adotada.

4.2.1.1. A FUNÇÃO NEW RESTRICTION

Objetivo:

Criar uma definição de restrição de acesso a ser utilizada posteriormente numa definição de acesso parcial. Exemplo: parte de um vetor ou componente de uma estrutura.

Nome da função:

mm_nrestriction

Argumentos da função:

kind_of_access_selection, restriction_i_dcb, select_access

Algorítmo:

```
testa_parâmetros
IF (erro)
   retorna (return_code=erro)  /*Parâmetro inválido*/
ELSE
   {
     Verifica_recursos
     IF (recurso não disponível)
        retorna (return_code=LOCAL_RESOURCE_PROBLEM)
     ENDIF
```

```
        Cria_definição_de_restrição
        mapeia_parâmetro_saída     /*MMS_Select_access*/
        retorna (return_code=OK)[2]
    }
ENDIF
```

Obs.:

São feitas tantas chamadas a essa função quantas forem as variáveis "restritas" a comporem uma especificação de acesso parcial (MMS-draft 5) ou alternado (MMS-draft 6).

Comentários:

Cabe no momento uma comparação de definições do tipo MMS_restriction (versão atual da API) com o tipo MMS_Select_Access (MMS-draft 6).

Nenhuma definição de parâmetro que corresponda à estrutura de um "MMS_restriction" foi encontrada no documento MMS.

Como a definição do parâmetro Partial Access Specification incorpora os parâmetros passados à função New_restriction e a própria especificação da função NEW PARTIAL ACCESS da API tem como entrada um parâmetro do tipo MMS_restriction, conduziu-se a uma definição do tipo MMS que contenha tal estrutura e que será componente da estrutura que define um "acesso alternado" ou acesso parcial (nome usado na API e MMS-draft 5): o MMS_Alternate_access.

O parâmetro Select Access definido no MMS contém informações de restrições identificadas por um dos 3 ítens: component_name, index ou index_range. Kind_of_acess_selection revela um desses três tipos.

4.2.1.2. A FUNÇÃO NEW PARTIAL ACCESS

Objetivo:

Criar uma definição de acesso parcial que será um elemento de uma lista a qual será usada na especificação de acesso a variável.

---

[2] A cada tipo de erro é associado um valor. O código de retorno OK tem valor 0 (zero).

Nome da função:

mm_npartialaccess

Argumentos da função:

kind_of_selection, partialaccess_i_dcb, alternate_access

Algorítmo:

```
testa_parâmetros
IF (erro)
   retorna (return_code=erro)  /*Parâmetro inválido*/
ELSE
   {
     Verifica_recursos
     IF (recurso não disponível)
        retorna (return_code=LOCAL_RESOURCE_PROBLEM)
     ENDIF
     Cria_definição_de_acesso_parcial
     mapeia_parâmetro_saída    /*MMS_Alternate_access*/
     retorna (return_code=OK)
   }
ENDIF
```

Obs.:

O parâmetro resultante da chamada dessa função é, na verdade, um elemento do tipo MMS_Alternate_access, pois a definição de um Alternate_access no MMS é uma lista (em ASN.1 um SEQUENCE OF [I7). Isso é realizado com chamadas a MAKE LIST, tantas quantas forem as forem as definições de acessos parciais.

Comentários:

Essa lista pode ser usado como parâmetro de entrada da função acima - NEW PARTIAL ACCESS, pois uma definição de um acesso alternado que não seja a primeira (de mais baixo nível) utiliza definições já construídas, recursivamente. Um exemplo é mostrado a seguir, onde se acompanha a construção gradativa de uma possível estrutura que defina um acesso alternado em 2 níveis. A função MAKE LIST é descrita no próximo ítem:

1ª chamada a NEW RESTRICTION
parâmetro resultante: *new_restriction1*

2ª chamada a NEW RESTRICTION
parâmetro resultante: *new_restriction2*

1ª chamada a NEW PARTIAL ACCESS
parâmetro resultante: *new_partial_access1*, contendo o parâmetro new_restriction1

1ª chamada a MAKE LIST
parâmetro resultante: *list_of_partial_access1*, contendo o parâmetro new_partialaccess1

2ª chamada a NEW PARTIAL ACCESS
parâmetro resultante: *new_partial_access2*, contendo o parâmetro list_of_partial_access1 + o parâmetro new_restriction2

2ª chamada a MAKE LIST
parâmetro resultante: *new_partial_access2*, contendo o parâmetro list_of_partial_ access1 + o parâmetro new_restriction2

A lista resultante desse exemplo contém um só elemento, e este, por sua vez, define um acesso alternado de 2 níveis.

Quando a lista contém mais de um elemento, o tipo resultante deste parâmetro é uma estrutura e os componentes tem seus tipos determinados pelas seleções especificadas, onde o nível mais interno descreve a restrição (MMS_Select Access ou MMS_restriction).

4.2.1.3. A FUNÇÃO MAKE LIST

Objetivo:

Inserir um elemento a uma lista. É passado um ponteiro para o início da lista. Caso seja nulo, significa que será criada uma nova

lista que conterá um só elemento.

Nome da função:

mm_makelist

Argumentos da função:

list, kind_of_element_list, list_i_dcb, new_list

Algorítmo:

```
testa_parâmetros
IF (erro)
   retorna (return_code=erro)/* Parâmetro inválido*/
ELSE
   {
     IF (ponteiro nulo)
        {
          verifica_recursos
          IF (recurso não disponível)
             retorna (return_code=LOCAL_RESOURCE_PROBLEM)
          ELSE
             cria nova lista
        }
     ENDIF
     insere elemento
     devolve nova lista acrescida do elemento /*MMS_list*/
     retorna (return_code=OK)
   }
ENDIF
```

Comentários:

A lista resultante é uma cópia física (não lógica) da lista original.

A API preserva a lista original, deixando para o usuário a tarefa de consumí-la. Isso foi decidido com base nos argumentos da própria função: se o usuário passa duas listas, uma como entrada e outra como saída, provavelmente necessita das 2 cópias, pois nada impediria que elementos fossem acrescidos a uma lista existente, seja qual fosse o tipo de implementação adotada.

Nessa implementação, por motivo de parametrização e facilidade de entendimento (requisitos do SISDI-MAP) adotou-se pela estrutura de vetor onde o limite de tamanho é definido pelo parâmetro

MAX_ELEMENT.

A função MAKE LIST é usada para construir qualquer lista existente na especificação do MMS. Um elemento a ser inserido na lista deve ser do mesmo tipo dos elementos já existentes.

No conjunto de serviços utilizados na classe de acesso a variáveis, do **MMS- draft 6**, encontram-se vários tipos de listas (tipos definidos como SEQUENCE OF). Relaciona-se em seguida todos eles na mesma ordem em que são encontrados no documento, e cita-se ao lado que função da API que contrói seus elementos.[3]

List of Components	-	New Component
List of Alternate Access	-	New Partial Access
List of Data	-	New Data
List of Variable	-	-
List of Scattered Access	-	New Scattered Access[4]
List of Access Result	-	-
List of Write Result	-	-

Ao se comparar com o **MMS-draft 5** para o qual a API está especificada, constata-se a incompatibilidade quase que total da API com essa versão MMS -draft 6.

No **MMS-draft 5** são citados:

- List of Variable
- List of Specifications
- List of Values
- List of Unsolicited Specifications
- List of Components Variables
- List of Unpacked Components
- List of Packed Components

---

[3] **A API** ainda introduz um novo objeto que poderá compor uma lista: List of Variable Association, cujo elemento é criado com a função: New Variable Association

[4] Para compatibilizar a **API** com o **MMS** - draft 6, definiu-se a nova função: **NEW SCATTERED ACCESS**, inexistente no documento **API [M5]**

- List of Selected Items
- List of Array Elements Values
- List of Struct. Component Values

A API na função MAKE LIST, cita apenas 4 tipos de elementos que compõem suas respectivas listas. São eles:

mms_component
mms_partial_access
mms_var_asso e
mms_data

Dispensa-se prolongar-se mais em comentários ...

4.2.1.4. A FUNÇÃO NEW COMPONENT

Objetivo:

Cria uma definição de estrutura. Uma lista dessas definições é usada na especificação de um tipo MMS (MMS-Type Specification), quando o tipo for "structure".

Nome da função:

mm_ncomponent

Argumentos da função:

kind_of_component, component_i_dcb, component_structure

Algoritmo:

```
testa_parâmetros
IF (erro)
   retorna (return_code=erro)/* Parâmetro inválido*/
ELSE
   {
     verifica_recursos
     IF (recurso não disponível)
        retorna (return_code=LOCAL_RESOURCE_PROBLEM)
     ENDIF
     cria_definição_de_componente_de_estrutura
     mapeia_parâmetro_saída    /*MMS_Structure*/
```

```
        retorna (return_code=OK)
      }
    ENDIF
```

Obs.:

Uma ou mais dessas definições compõem a lista de definições MMS_Structure posteriormente utilizada na especificação de um novo tipo com chamada(s) a NEW TYPE.

Comentários:

A definição de estrutura é feita de forma recursiva: componentes formam componentes de estruturas. Tipos elementares são usados nos componentes mais internos. Veja próxima função.

4.2.1.5. A FUNÇÃO NEW TYPE

Objetivo:

Construir uma definição de tipo MMS.

Nome da função:

mm_ntype

Argumentos da função:

kind_of_type, type_i_dcb, type_specification.

Algorítmo:

```
testa_parâmetros
IF (erro)
   retorna (return_code=erro)  /*Parâmetro inválido*/
ELSE
   {
     Verifica_recursos
     IF (recurso não disponível)
        retorna (return_code=LOCAL_RESOURCE_PROBLEM)
     ENDIF
     Cria_especificação_de_tipo
     mapeia_parâmetro_saída     /*MMS_Type*/
     retorna (return_code=OK)
   }
ENDIF
```

Obs.:

A construção do tipo MMS é feita de modo recursivo: Tipos são construídos a partir de tipos já construídos.

Os valores de kind_of_type são os seguintes: type_name, array_type, structure_type e simple_type.

Os tipos elementares MMS podem ter as seguintes classes: Boolean, Bit_string, Integer, Unsigned, Floating_point, Octet_string, Visible_string, Generalized_time, Binary_time e BCD.

Comentários:

Consegue-se criar tipos dos mais simples aos mais complexos.

4.2.1.6. A FUNÇÃO NEW VARIABLE SPECIFICATION

Objetivo:

criar uma especificação de variável, parâmetro obritatório numa definição de acesso a variável.

Nome da função:

mm_nvspecification

Argumentos da função:

Kind_of_variable, vspec_i_dcb, var_specification.

Algorítmo:

```
testa_parâmetros
IF (erro)
   retorna (return_code=erro)  /*Parâmetro inválido*/
ELSE
   {
     Verifica_recursos
     IF (ecurso não disponível)
        retorna (return_code=LOCAL_RESOURCE_PROBLEM)
     ENDIF
     Cria_especificação_de_variável
     mapeia_parâmetro_saída     /*MMS_Variable_specification*/
     retorna (return_code=OK)
   }
ENDIF
```

Obs.:

Uma especificação de variável define um dos 4 seguintes tipos de variável: named, unnamed, single ou scattered. O parâmetro kind_of_variable determina o tipo resultante.

Comentários:

A especificação de variável é a informação mínima para se definir um acesso a variável MMS. As variáveis MMS podem ser especificadas ou por nome, ou endereço, ou endereço e tipo ou, enfim, por uma definição de acesso disperso.

Uma especificação de variável pode usar a definição de especificação de acesso disperso, desde que essa definição já tenha sido construída pela função NEW SCATTERED ACCESS (Veja ítem 4.2.1.7) a qual embute uma especificação de variável. Percebe-se a construção de forma recursiva para tipos dispersos.

4.2.1.7. A FUNÇÃO NEW SCATTERED ACCESS

Objetivo:

construir uma especificação de acesso disperso. Essa definição utiliza uma definição de especificação de variável.

Nome da função:

mm-nscatteredaccess

Argumentos da função:

var_specification, scattered_i_dcb, scattered_access

Algorítmo:

```
testa_parâmetros
IF (erro)
   retorna (return_code=erro)  /*Parâmetro inválido*/
ELSE
   {
     Verifica_recursos
     IF (recurso não disponível)
        retorna (return_code=LOCAL_RESOURCE_PROBLEM)
     ENDIF
```

```
        Cria_especificação_de_acesso_espalhado
        mapeia_parâmetro_saída     /*MMS_scattered_access*/
        retorna (return_code=OK)
      }
    ENDIF
```

Obs.:

A definição de uma variável do tipo "scattered" na verdade é uma combinação de várias outras variáveis (qualquer tipo) em forma de uma lista. Opcionalmente um acesso alternado pode ser empregado sobre uma especificação de variável. A definição de um acesso alternado foi descrito em 4.2.1.2.

Comentérios:

Essa função teve que ser criada e definida para se obter a compatibilidade da API com o protocolo MMS-draft 6, pois para o draft 5 ela não existia.

4.2.1.8. A FUNÇÃO NEW VARIABLE ACCESS

Objetivo:

Criar uma especificação de acesso a variável necessária a realização de leituras e escritas de variáveis através da rede.

Argumentos da função:

kind_of_access, vaccess_spec_i_dcb, var_access_specification

Algoritmo:

```
testa_parâmetros
IF (erro)
   retorna (return_code=erro)  /*Parâmetro inválido*/
ELSE
   {
     Verifica_recursos
     IF (recurso não disponível)
        retorna (return_code=LOCAL_RESOURCE_PROBLEM)
     ENDIF
     Cria_especificação_de_acesso
     mapeia_parâmetro_saída     /*MMS_var_access-specification*/
```

```
            retorna (return_code=OK)
        }
    ENDIF
```

Obs.:

É a especificação de acesso a variável que é levada dentro de uma PDU ao usuário MMS destino no caso de uma leitura/escrita/relatório de informação. Esta definição é requerida como parâmetro de entrada nas requisições dos serviços acima e contém uma das seguintes informações (CHOICE):

- uma lista contendo uma ou mais especificações de variável (construídas pela função NEW VARIABLE SPCIFICATION) definida no MMS como List of Variable.
- nome da lista que contém as especificações de variável (Essa lista deve ser nomeada utilizando-se do serviço MMS especificado para este fim: DEFINE VARIABLE NAME, no draft 5; DEFINE NAMED VARIABLE LIST, no draft 6).

Comentários:

A definição desta função foi condicionada à inexistência de uma função que provesse ao usuário informações sobre uma especificação de acesso de variável, requeridas no parâmetro Var_access_specification existente no MMS. Esse parâmetro é requerido como parâmetro de requisição dos seguintes serviços MMS: Read, Write e Information Report, portanto é necessária sua definição prévia.

O parâmetro Kind_of_access determina que tipo de informação essa definição conterá. Seus possíveis valores são: list_of_variable e variable_list_name.

Com as definições até agora descritas, é fácil visualizar o embutimento de definições noutras definições. Isso permite a construção dos tipos complexos e aninhados requeridos no MMS. A figura 4.1 ilustra uma construção completa de uma estrutura, realizada através de chamadas consecutivas, muitas vezes recursivas, às funções auxiliares de serviços de acesso a variáveis da API.

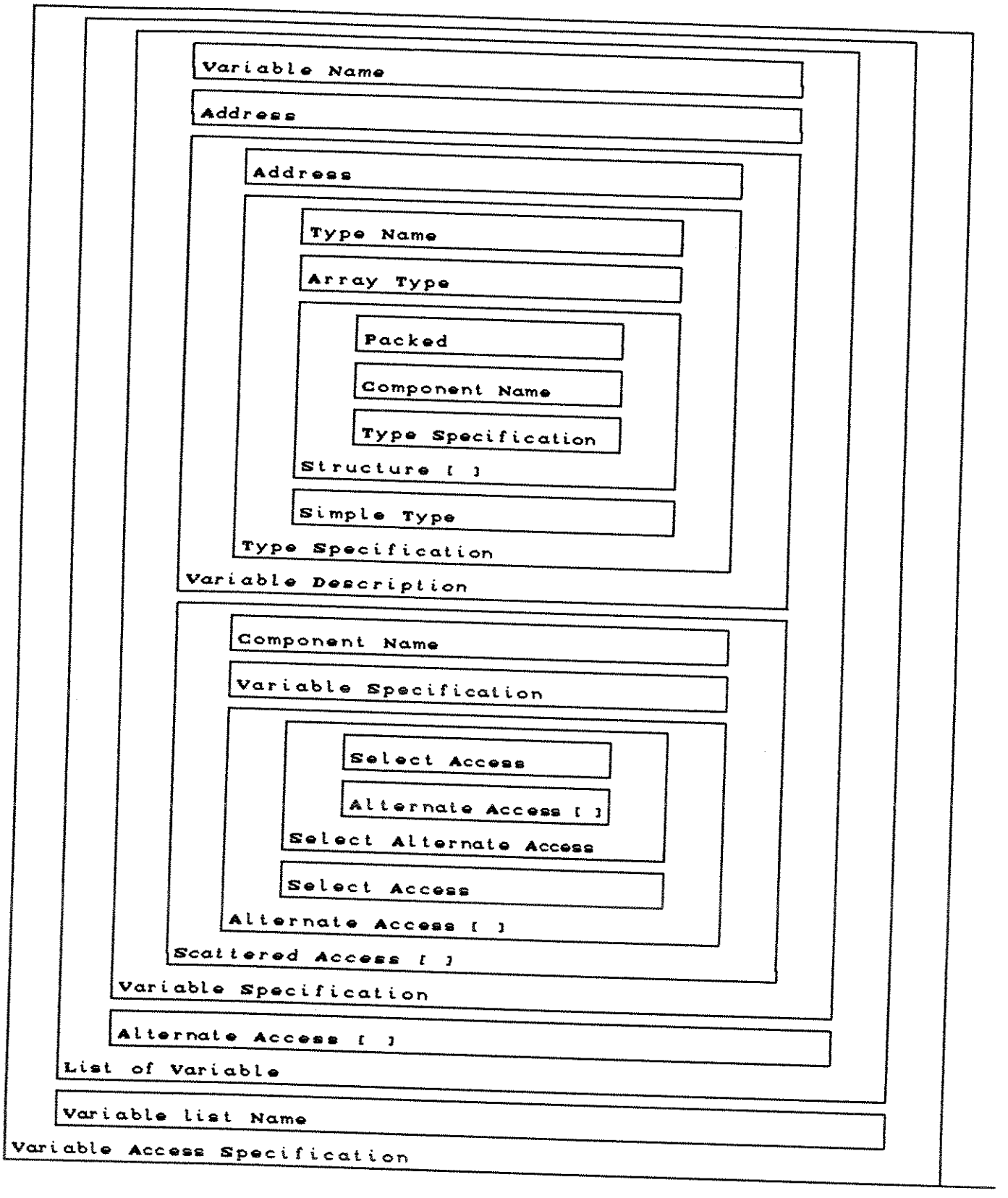


Figura 4.1 - Exemplo de definições aninhadas requeridas no MMS.

4.2.1.9. A FUNÇÃO NEW VARIABLE ASSOCIATION

Objetivo:

Especificar uma associação de variável com endereços locais para futuro mapeamento de especificação de variável MMS com a representação dos respectivos dados (MMS data).

Nome da função:

mm_nvasso

Argumentos da função:

Kind_of_asso, asso_i_dcb, var_association

Algoritmo:

```
testa_parâmetros
IF (erro)
   retorna (return_code=erro)  /*Parâmetro inválido*/
ELSE
   {
     Verifica_recursos
     IF (recurso não disponível)
        retorna (return_code=LOCAL_RESOURCE_PROBLEM)
     ENDIF
     Cria_definição_de_associação_de_variável
     mapeia_parâmetro_saída    /*Var_association*/
     retorna (return_code=OK)
   }
ENDIF
```

Obs.:

Existirá uma especificação de associação em cada usuário interessado em acessar uma variável MMS. As associações são individuais a cada especificação de acesso a variável requerida aos read/write's. Uma lista de associações é passada à API nas chamadas de funções de leitura/escrita de variáveis.

Comentários:

Uma definição de associação significa uma "ligação" entre a representação de um tipo MMS com a representação local de dados.

Esta função é na verdade a que encerra maior número de informações quanto ao mapeamento de uma representação de variável MMS (conhecida na rede) com a representação local, onde os tipos primitivos MMS são mapeados nos tipos específicos da linguagem.

A especificação da API não sugere nenhuma informação de como esse mapeamento é feito. Ao invés:

1- cita um objeto em sua definição que só tem sentido local (asso_name);

2- compõe uma definição de associação local: portanto não tem significado na rede

3- relaciona o parâmetro de saída (mms_var_asso) a um tipo que nãocorresponde (não tem equivalência) nas definições de tipos MMS (de rede) portanto, termina apresentando uma definição contendo ambiguidades e falta de especificidades intelegíveis.

Conclui-se que a API cria essa nova definição somente para propósito local. Deve existir uma definição de associação em cada nó da rede que utilize uma determinada especificação de variável de rede e ainda deve existir, na associação, uma representação local de dados MMS, como uma cópia mantida por cada usuário interessado na variável.

Como foi implementado?

Com a definição de uma nova função, New Variable Access (não corresponde à definição da API), constrói-se parte da informação (var_access_specification) que vai compor uma associação e recebe-se através dos DCB's as informações adicionais necessárias à associação (mms_data, data_access_error). Essa associação terá significado local e será usada nas operações de leitura/escrita de variáveis. Uma lista de associações é passada à API numa chamada de read/write.

Na implementação, assume-se que a API não faz o mapeamento (equivalentes ao V-GET e V-PUT do MMS-draf 6), deixando a encargo do usuário. O usuário cria a associação entre um "Var_access_specification de variável de rede" (MMS_var_access_specification) com um "Data local" (MMS_data).

A associação permite que o usuário obtenha a lista de dados que o usuário poderá processar usando as funções Extract List Element e Interpret Data para cada mms_data da lista.

4.2.1.10. A FUNÇÃO NEW DATA

Objetivo:

Criar uma representação para dados MMS.

Nome da função:

mm_ndata

Argumentos da função:

kind_of_data, data_i_dcb, new_data

Algorítmo:

```
testa_parâmetros
IF (erro)
   retorna (return_code=erro)  /*Parâmetro inválido*/
ELSE
   {
     Verifica_recursos
     IF (recurso não disponível)
        retorna (return_code=LOCAL_RESOURCE_PROBLEM)
     ENDIF
     Cria_definição_de_dados
     mapeia_parâmetro_saída     /*MMS_data*/
     retorna (return_code=OK)
   }
ENDIF
```

Obs.:

A definição de dados é recursiva, permitindo representar dados da mesma forma que um tipo MMS pode especificar: simples, array ou estruturas aninhadas. O parâmetro kind_of_data determina o tipo, assumindo os seguintes valores: structure_data, array_data, ou um dos valores representando os tipos simples (Boolean, Bit_string, etc)

Comentários:

A atual especificação da API não provê funções que mapeiem variáveis abstratas em reais. A especificação de um dado MMS, constitui de valores para os dados simples e uma lista de valores para representar dados de um vetor ou estrutura. Esses valores de dados vão ter faixas e grandezas de acordo com o tipo que representam.

Para efeito de teste da implementação, foi considerado que esses valores de dados são passados, via API, pelo usuário, através do DCB de entrada (data_i_dcb) e a função mm_ndata simplesmente mapeia esses valores na estrutura MMS_data.

4.2.2. AS FUNÇÕES AUXILIARES DE INTERPRETAÇÃO DE DEFINIÇÕES MMS

Do mesmo modo que estruturas complexas foram definidas, passo a passo, pelo usuário (AP) utilizando-se das funções de construção, estas são decompostas ou interpretadas, passo a passo, pelo usuário par (AP remoto) através das funções de interpretação. Há uma simetria entre esses dois tipos de funções e o usuário deve utilizá-las da mesma forma simétrica que o usuário construtor, conforme figura 4.2.

AP LOCAL	**AP REMOTO**
Antes da função principal:	Ao chegar a primitiva indication:
7. mm_new_asso	1. mm_interpret_asso
6. mm_makelist	2. mm_extract_list_element
5. mm_new_access	3. mm_interpret_access
4. mm_new_var_spec	4. mm_interpret_var_spec
3. mm_makelist	5. mm_extract_list_element
2. mm_new_partial access	6. mm_interpret_partial access
1. mm_new_restriction	7. mm_interpret_restriction

Figura 4.2 - Simetria das funções auxiliares da API

São interpretadas as associações, os acessos, as especificações de variáveis, os tipos, as seleções e restrições. Dados em formato MMS também são interpretados via função de interpretação. Listas construídas via MAKE LIST têm seus elementos extraídos via EXTRACT LIST ELEMENT em ordem simétrica à inserção: o último elemento inserido será o primeiro a ser extraído.

Como fica enfadonho repetir todas as descrições de funções de interpretação, sem trazer novidades nas estruturas manipuladas, citam-se todas as funções na ordem inversa à de construção (ítem 4.2.1), mostrando a simetria e consistência da utilização:

INTERPRET DATA REPRESENTATION
INTERPRET ASSOCIATION
INTERPRET VARIABLE ACCESS
INTERPRET SCATTERED ACCESS
INTERPRET VARIABLE SPECIFICATION
INTERPRET TYPE SPECIFICATION
INTERPRET COMPONENT
EXTRACT LIST ELEMENT
INTERPRET PARTIAL ACCESS
INTERPRET RESTRICTION

Para ilustrar a utilização coerente dessas funções, para o mesmo exemplo utilizado na seção 4.2.1.2, mostra-se a sequência de chamadas da API_par às funções de interpretação:

1ª chamada a EXTRACT LIST ELEMENT
parâmetro resultante: um elemento do tipo Alternate Access especificando uma seleção do tipo acesso "alternado"

1ª chamada a INTERPRET PARTIAL ACCESS
parâmetro resultante: uma descrição de restrição e uma lista (porque nessa definição o tipo de seleção de acesso é alternado kind_of_access_selection = select alternate access)

2ª chamada a EXTRACT LIST ELEMENT
parâmetro resultante: um elemento do tipo Alternate Access especificando uma selação de "acesso simples"

2ª chamada a INTERPRET PARTIAL ACCESS
parâmetro resultante: uma descrição de restrição kind_of_access_selection = select access

1ª chamada a INTERPRET RESTRICTION
parâmetro resultante: uma especificação de índice, faixa de índice ou identificação de componente.

2ª chamada a INTERPRET RESTRICTION
parâmetro resultante: idem.

Obviamente nada impede a mudança de seqüência de chamadas quando o resultado de uma não implica na outra: questão de opção do usuário.

4.2.3. A FUNÇÃO DE LEITURA DE VARIÁVEL: READ VARIABLE

As funções principais da API requerem um tratamento diferenciado das funções auxiliares, discutidas no item anterior, devido, principalmente, a essas serem sensíveis ao contexto enquanto as auxiliares são livres de contexto.

Um pedido de leitura de variável, por exemplo, será aceito somente se o mesmo for feito dentro do contexto negociado e estabelecido entre os pares comunicantes. Para satisfazer essa exigência, são negociados nos CBB's as operações possíveis e os atributos dos objetos MMS em questão. O protocolo MMS faz as devidas averiguações nas primitivas à luz do contexto previamente estabelecido. Para a realização do serviço de leitura de variável, por parte da API_MMS, solicitado pelo AP através da chamada de função READ VARIABLE (mm_vread), considera-se que o AP tenha solicitado antes o estabelecimento de associação, utilizando-se das funções de gerenciamento de contexto providas pela API [T1].

Outra característica das funções auxiliares é que elas não requerem da API procedimentos do Provedor de Serviços de Rede. Ao

invés, a API só realiza os procedimentos referentes à Biblioteca de Funções, pois elas não fluem pela rede.

Já as funções principais da API são tratadas primeiramente a nível de biblioteca de funções, tomando, posteriormente, uma direção a um dos módulos funcionais do Provedor de Serviços de API (HLSP, CSP, IFA OU PSP), de acordo com o tipo de serviço solicitado pelo usuário.

A especificação da API permite assincronismo na comunicação interna de interação entre seus módulos funcionais, pois estes mantem suas funcionalidades independentes. Porém, visando o SISDI-MAP, adotou-se, por implementar os módulos funcionais como procedimentos, os quais são chamados com passagem de parâmetros.

A partir da chegada do pedido de serviço de READ VARIABLE, a API segue os passos de tratamento de um serviço de baixo-nível, confirmado. Basicamente, o algoritmo da função mm_vread tem os seguintes passos:

- verifica, através do parâmetro connection_id a que contexto de associação pertence e se é valido
- verifica se o pedido é síncrono ou assíncrono e as condições de realizar o pedido. No caso de pedido assíncrono, não podendo atender, verifica se pode aceitá-lo como pendente (limite de serviços não atendidos pendentes)
- armazena as informações de identificação (connection_id) e eventos associados (return_event_name).
- testa parâmetros específicos da função
- obtem identificador para o pedido (invoke_id)
- mapeia os parâmetros passados pelo usuário na primitivado MMS Read-request
- armazena informações referentes ao usuário pelo *invoke_id*
- e envia à máquina de protocolo (PM)...

Em qualquer etapa, na falta de requisitos ou condições de progredir, a API retorna ao usuário um código de erro que retrate o ocorrido.

Não havendo erro, após o envio da primitiva, se o serviço está sendo realizado de modo assíncrono, a API sinalisa o envio e aguarda pela confirmação desse pedido, com os parâmetros de saída do

usuário armazenados e devidamente identificados pelo invoke_id.

... Na chegada da primitiva de confirmação para esse serviço:

- verifica se o serviço continua pendente à espera de confirmação (pode ter havido um ABORT ou REJECT), pelo connection_id
- recupera informações do usuário pelo *invoke_id*
- mapeia os parâmetros da primitiva de confirmação do MMS (Read_confirm) nos parâmetros de saída do usuário que estavam armazenados.
- retorna resultados ao usuário.

Para melhor acompanhar toda a seqüência de operações numa leitura de variável, em ambos os lados da comunicação, descreve-se, identificando os pares comunicantes como A (solicitante) e B (respondedor), e acompanhado o diagrama temporal da figura 4.3, a seqüência de funções auxiliares envolvidas e as primitivas MMS que fluem entre ambos. As operações que não estão esclarecidas na especificação da API, são colocadas em parênteses, assumindo-se que é de responsabilidade do usuário prover essas informações (numa versão futura da API, essas funções podem ser passadas à mesma, permitindo, inclusive, a resposta automática referida em 4.1.2):

1- O usuário A, que solicita uma leitura de variável deve ter construído à priori a definição do acesso a variável de maneira a representar a variável MMS. A definição é reconhecida pela API-A e pela MP (Máquina de Protocolo) local e deverá ser interpretada no par destino (usuário B) para que o mesmo responda à solicitação corretamente com dados MMS os quais fluirão no sentido contrário pela rede, como retorno à solicitação de A.

O parâmetro do tipo MMS Var_access_specification resultante será posteriormente utilizado como parâmetro de entrada da primitiva Read_request no passo 4.

2- O usuário A cria uma associação ("ligação") entre a lista de especificações de variáveis contidas na especificação de acesso e as representações locais de dados.

É necessário que o usuário A aloque à priori a área de

armazenamento dos dados de resposta (list_of_access_result).

3- O usuário A passa a lista de associações ou nome da lista, embutidas numa especificação de acesso à variável, devidamente associada às especificações de dados locais, na chamada de READ VARIABLE. Isso permite a leitura de variáveis múltiplas numa só chamada.

4- A API mapeia a lista ou nome da lista na primitiva de Read_request.

Na implementação, devido ao surgimento de um novo parâmetro de entrada numa primitiva de read_request (specification With Result), no MMS-draft 6 optou-se por atribuir-lhe o valor FALSE e colocá-lo num DCB de entrada (read_i_dcb), juntamente com o parâmetro Variable Access Specification.

5- Ao chegar a indicação de read ao destino B, a API-B verifica se tem condições para receber essa indicação, averigua as condições de resposta automática e se não está programada para isso, verifica se há autorização do usuário para receber indicações (uma função de INDDICATION RECEIVE já deve ter sido inserida pelo usuário para declarar sua permissão).

6- No caso da API-B não responder automaticamente à solicitação, o usuário faz as devidas interpretações utilizando-se das funções auxiliares.

7- São feitos os mapeamento (V-Get). As operações de leituras são feitas atômicamente: para cada variável não acessada, um data_access_error é atribuído.

8- O usuário B deve construir uma especificação de dados MMS associada à especificação de variáveis, fazer a "ligação" entre ambos e torná-la disponível ao solicitante. A função New Data é chamada tantas vezes quantas forem os dados solicitados.

9- O usuário B cria sua lista de associação de variável, contendo os dados. (Refira-se à seção 4.2.1.9).

10- Uma primitiva de Read_response é construída pela API-B,a partir da lista de associações mapeando-se no parâmetro List_of_access_result da primitiva.

11- Na chegada da confirmação, a API-B verifica as condições (se não houver término de associação) e recebe a confirmação.

12- Processa o parâmetro da primitiva da confirmação. Se for negativa, mapeia o código de erro no parâmetro "erro_block" do usuário, do DCB de saída (read_o_dcb). Se a confirmação for positiva, mapeia o parâmetro list_of_access_result da primitiva nos parâmetros da associação fornecido pelo usuário.

13- A cada componente da lista list_of_access_result, um parâmetro de resultado (success_result) determinará se a operação de read foi bem sucedida - TRUE, ou mal sucedida - FALSE. Neste caso, data_access_erro assume um dos possíveis erros ocorridos.

14- Se o usuário recebe os dados em forma de lista, estes são extraídos e interpretados um a um, usando as funções auxiliares EXTRACT LIST e INTERPRER DATA.

A operação de leitura (ou escrita) de variável implica em aspectos locais de ambos os sistemas comunicantes. Aspectos como a manutenção de contexto e gerenciamento de eventos estão envolvidos durante todo o ciclo até o fechamento da operação completa.

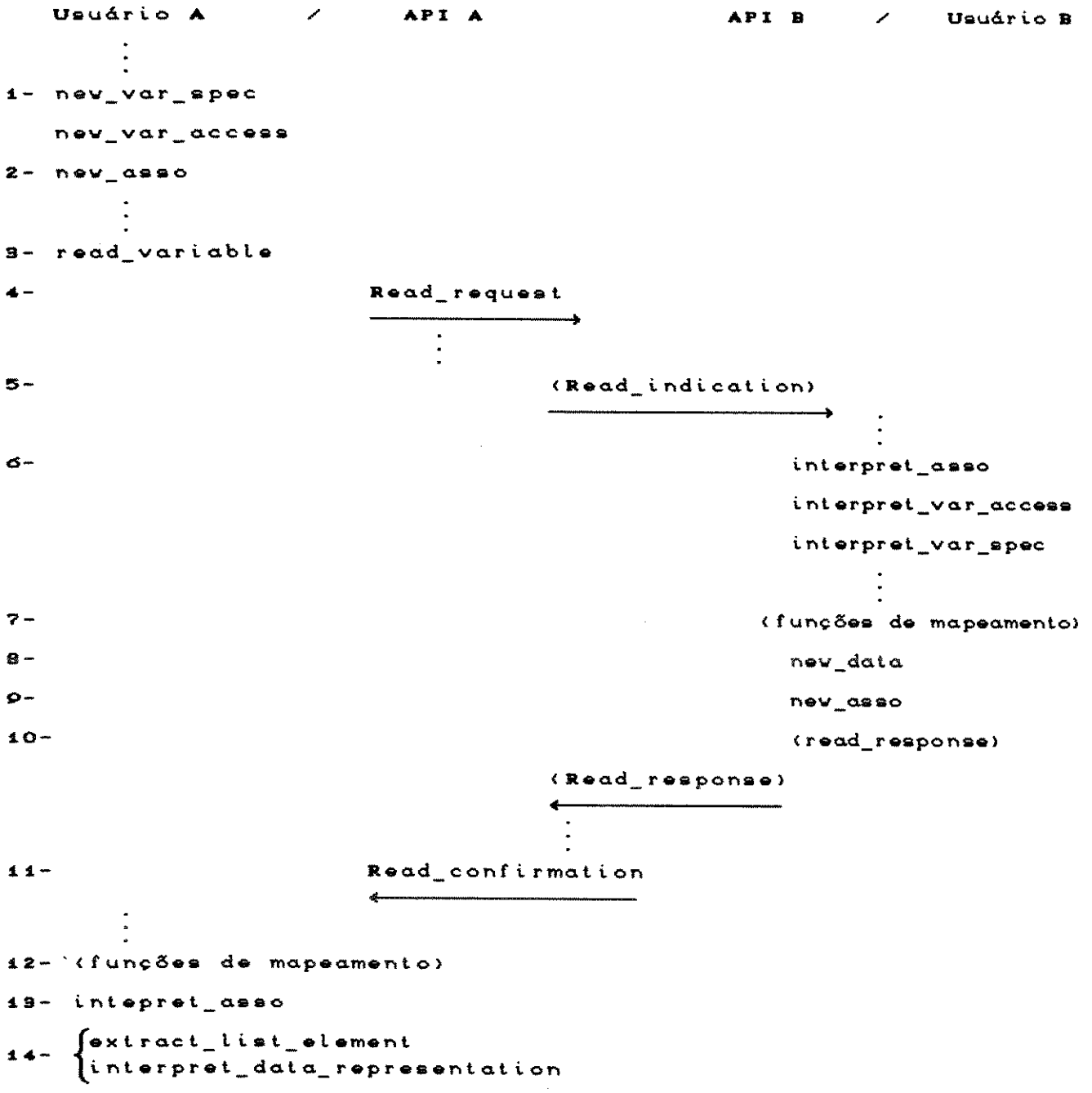


Figura 4.3 - Diagrama temporal de uma operação de leitura de variável.

4.3. A INTEGRAÇÃO DA API NO SISDI-MAP

Visando integrar a API no SISDI-MAP, alguns aspectos foram abordados sobre particularidades de implementação:

4.3.1. COMUNICAÇÃO E SINCRONIZAÇÃO ENTRE PROCESSOS

A comunicação entre os processos dá-se através de "portas" ou "caixas-postais". A própria comunicação via portas provê um meio de sincronizar processos. O acesso dos vários AP's a uma mesma porta da API é garantida utilizando-se semáforos. Foram igualmente utilizados semáforos como mecanismo de sincronização entre AP's e API. As definições de portas e semáforos foram feitas conforme especificações do núcleo. Portas e semáforos são identificados e alocados estaticamente no processo.

Através de uma porta fluem mensagens de um só tipo de estrutura. Isso levou à adaptação do formato de chamada de função da biblioteca da API a um formato único, de tamanho pré-definido contendo os parâmetros da chamada de função.

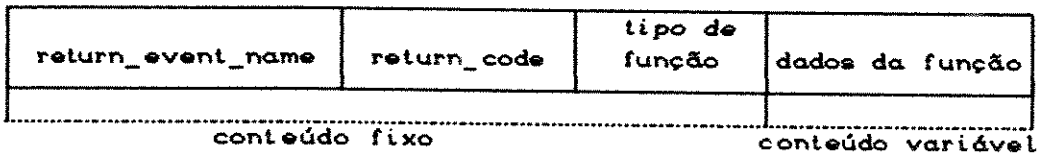
É o próprio endereço da mensagem que é colocado na porta.

As chamadas do núcleo para colocar e retirar mensagens das porta são, respectivamente: **envia_mensagem ()** e **recebe_mensagem ()**.

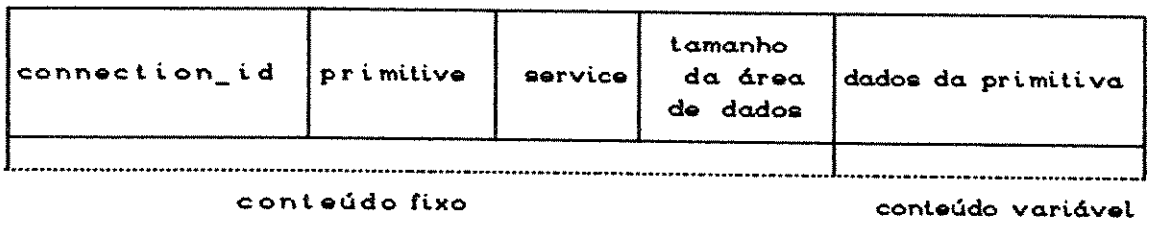
4.3.2. FORMATOS DE MENSAGENS

Como a API se comunica com o processo de aplicação (AP) e com o protocolo MMS, há, portanto dois tipos de mensagens fluindo nas duas portas: a mensagem API-AP e a mensagem API-MMS, respectivamente.

Em ambas as mensagens há um conteúdo fixo e um conteúdo variável de tamanho conhecido, conforme a figura 4.4.



a) Mensagem API-AP



b) Mensagens API-MMS

Figura 4.4 - Formatos de mensagens: a) mensagem API-AP
b) mensagem API-MMS

No caso da mensagem API-AP, os parâmetros comuns a todas as chamadas de funções são alocados na parte fixa da mensagem. Já a parte variável irá ser composta dos parâmetros específicos de cada função.

O mesmo acontece com a mensagem API-MMS em relação às primitivas que transitam entre ambos os processos.

As definições das estruturas das duas mensagens, em C, são mostradas a seguir:

```
Typedef struct {
        Return_event_name                   return_event_name;
        Return_code                         return_code;
        enum Function_type                  função;
        union {
                Mm_aeactivation             mm_aeactivation;
                Mm_connect                  mm_connect;
                :

                Mm_vread                    mm_vread;
                Mm_vwrite                   mm_vwrite;
                Mm_ireport                  mm_ireport;
                :
        } function_type_union;
} Block_api_ap;
```

```
Typedef struct {
                Connection_id                   connection_id;
                enum Primitive_type             primitive;
                enum Service_type               service;
                sint 16                         data_area_lenght;
                union {
                        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;
```

Portanto, a estrutura da chamada de função mm_vread, agora contida na estrutura da mensagem API-AP, passou a ter o seguinte formato:

```
Typedef struct {
                Connection_id                   connection_id;
                Vread_i_dcb_type                vread_i_dcb;
                Vread_o_dcb_type                vread_o_dcb;
              } Mm_vread;
```

O projeto do software segue um modelo do tipo "Data_Structure Driven" [L7], em que os procedimentos são selecionados de acordo com a estrutura do dado processado.

Em suma, o processo API consiste num "laço" (loop) infinito à procura de mensagens em suas portas. Como há dois tipos de mensagem (**mensagem API-AP** e **mensagem API-MMS**), a API resume-se em dois

procedimentos principais:

- Procedimento de tratamento das mensagens API-AP (Trata_ap)
- Procedimento de tratamento das mensagens API-MMS (Trata_mms)

Em "trata_ap" analisa-se o **tipo de função** e, dependendo deste, chama-se o procedimento apropriado. No caso da implementação dos serviços de acesso a variáveis, as funções auxiliares tem um tratamento mais simplificado do que a função Read Variable, que, por se referir a um serviço confirmado, requer procedimentos de serviços confirmados, como: alocação prévia de recursos e enfileiramento (armazenamento) da mensagem do usuário para espera de confirmação, geração de identificação única de chamada confirmada, etc.

Já em trata_mms, o **tipo de primitiva** inserida, (*indication* ou *confirm*) conduz aos procedimentos apropriados: Trata_indication ou trata_confirm.

Por sua vez, em trata_confirm, analisa-se o **tipo do serviço** (serviços de gerenciamento de contexto ou serviço MMS confirmados).

Depois, analisa-se o **nome do serviço MMS confirmado**. No caso de ser "read", o procedimento de confirmação do serviço Read Variable é selecionado.

Os algoritmos gerais da API foram feitos pelo grupo SISDI-MAP do processo API, e estão descritos, em pseudo-código, em [A2].

Para a versão atual do SISDI-MAP, definido num modelo simplificado, a ser executado num IBM-PC/DOS, estabeleceu-se, por economia de recursos de máquina, que cada processo teria apenas uma instância. Portanto cada processo deve manter um controle interno das atividades de solicitante e respondedor.

A API realiza esse controle, guardando o contexto de cada um dos elementos comunicantes de forma unicamente identificável.

Outro aspecto abordado na implementação do SISDI-MAP diz respeito à prioridade dos processos. Foi adotada a atribuição de prioridades iguais a todos, com exceção do processo PROC_OP (Interface de Operação do Usuário) devido ao próprio objetivo do sistema.

4.4 CONCLUSÕES

A implementação da API foi realizada de forma modular. Os módulos funcionais que constituem a API_MMS, foram implementados como procedimentos ou "funções" (como são chamados em C).

O projeto do software seguiu um modelo orientado a estrutura de dados, com refinamentos sucessivos das tarefas internas, guiando-se a uma execução sequencial.

A estrutura global resume-se em dois grandes procedimentos que são selecionados conforme o tipo de mensagem recebida em suas portas: mensagem API-AP e mensagem API-MMS. O refinamento das funções mais internas prossegue de acordo com a estrutura dos dados que vão sendo analisados.

Para viabilizar o serviço de acesso a variáveis, diversas considerações foram assumidas, como definição de novas funções auxiliares e alterações nos parâmetros do usuário.

A ferramenta turbo-C propiciou de forma considerável na implementação e teste do software, com seus recursos de "DEBUG", "BREAK POINTS", "BUILD", "PROJECT" e outros, proporcionando versatilidade e flexibilidade em ambas etapas de desenvolvimento do software.

CAPÍTULO 5

CONSIDERAÇÕES GERAIS, CONCLUSÕES FINAIS E SUGESTÕES

Durante o desenvolvimento do trabalho, procurou-se observar os aspectos relacionados à concepção do Modelo OSI, ao atendimento dos requisitos de padronização e ao compromisso especificação/implementação.

Procura-se destacar aqueles mais significativos, comentando-os e sugerindo idéias.

5.1. PADRONIZAÇÃO E MODELO OSI

Na mesma trilha seguida para padronização dos sistemas de comunicação pela ISO, o projeto MAP/TOP contribuiu com a padronização de uma interface entre software aplicativo e serviços de rede.

Pôde-se observar que a especificação da API extende a abordagem do modelo hierárquico da ISO, constituindo-se de um provedor de serviços de rede. Alguns conceitos e definições da ISO, como identificações, entidades, relação usuário-provedor puderam ser bem visualizados, apesar de outros não estarem bem esclarecidos, devido ao modelo substancialmente abstrato adotado nas especificações dos sistemas de comunicações, bem como à interface, dentre eles, os conceitos relacionados a endereçamentos. Como discussão desses aspectos, sugere-se o artigo [A4], o qual trata de diversas abordagens de implementação de sistemas OSI.

A inclusão da API entre AP e protocolo de aplicação sugere a ampliação da arquitetura RM-OSI, com todas as implicações do acréscimo de mais uma "camada", em termos de gerenciamento, overhead, tempo de resposta. Porém, do ponto de vista do usuário (AP), é importante que se transfira tarefas (através de pedidos de serviços) a outros, com a finalidade de se liberar para outras atividades. O essencial é não haver incompatibilidades entre os elementos adjacentes.

5.2 ABRANGÊNCIA DO MODELO DA INTERFACE

O modelo geral apresentado é constituído de blocosfuncionais independentes e, portanto, atende ao requisito demodularidade, essencial em sistemas heterogêneos.

Para a aplicação MMS, para a qual foi implementada, onde o modelo Cliente-Servidor sugere complexidades distintas, a API comprova que pode ser modelada de acordo com a aplicação, evitando sobrecargas e desperdícios de capacidades e recursos.

Nesta implementação particular, foram selecionados dentre os serviços da interface, os serviços referentes somente ao protoccolo de aplicação MMS e, dentre estes, aqueles referentes ao gerenciamento de conexão [T1] e um subconjunto dos serviços de acesso a variáveis. A critério do implementador, outras funções de serviços MMS podem ser incorporadas facilmente, num crescente aumento de complexidade.

Como sugestão de continuidade do trabalho, poderiam ser implementados novos serviços do mesmo sistema de comunicação, MMS, ou de outros sistemas de aplicação, como, por exemplo, DS, RDA, FTAM.

O trabalho de E.R.M.M [T7], que está sendo desenvolvido em paralelo a esta tese aborda o modelo geral de API e incorpora novos módulos funcionais necessários para se alcançar a funcionalidade global da API tanto para o sistema MMS como para outros protocolos de aplicação.

5.3 DESEMPENHO AP/API E ASPECTOS LOCAIS

O AP terá seu desempenho comprometido com a funcionalidade da API. Atribuindo mais funções à interface, desde que se forneça os recursos necessários, esta absorve mais atividades, até então de responsabilidade do usuário. É preciso perceber até que ponto se pode repassar atribuições à interface sem que haja degradação no desempenho do AP, que espera por respostas dos serviços solicitados.

Uma observação importante diz respeito à construção das definições dos tipos MMS. Nessa tarefa, a API parece comportar-se como uma mera mapeadora. Mas, a construção dessas estruturas, passo a

passo, irão trazer benefícios quando, numa implementação real, isso for uma tarefa repetitiva: estruturas simples já definidas irão compor, posteriormente, as estruturas mais complexas, nas mais diversas formas combinadas. A proposta de padronização da interface conduz a geração de AP's padronizados, o que leva a utilização, de forma amigável, dos protocolos de comunicação.

Considerando que a especificação atual da API deixa emaberto algumas atribuições de atividades, conclui-se que o AP as assume, enquanto as mesmas não forem atribuídas à API.

5.4 EXTENSÃO DAS FUNCIONALIDADES DA API

Foi decidido atribuir ao AP certos procedimentos por falta de clareza ou de uma especificação exata. Um exemplo é o mapeamento de variáveis abstratas na representação de máquina. Como primeira versão da API, estabeleceu-se que o usuário mantem essas variáveis sob seu controle e a API não exerce operações diretas sobre as mesmas. Em futuras versões, essas funções poderão ser delegadas à API, e esta terá autonomia de acesso e o fará sem interferência do AP.

No tocante à implementação dessas extenções, a interface foi implementada de forma a garantir a inclusão de novos módulos funcionais e de novas funções de biblioteca que acompanhem essas extensões.

Na versão atual da interface, observa-se a limitação nas funções respondedoras. Resposta automática é um aspecto imprescindível a ser incorporado a uma interface do tipo servidora.

5.5 COMPATIBILIZAÇÃO API-MMS

Eram previstas as dificuldades em concretizar a compatibilização da especificação da interface para o MMS-draft 5 com o protocolo MMS-draft 6. Houve necessidade de se definir novas funções para a API, definições foram revistas na nova versão do MMS, sob o enfoque de um modelo de MMS-servidor, num VMD. Por sua vez, a especificação da API está dirigida a uma aplicação mestre, essencialmente requisitante.

Conclusão: A API é deficiente em funções respondedoras e necessitará de funções que acionem as atividades de um VMD-servidor para interfacear-se como tal.

5.6 INTERAÇÃO DA API NO SISDI-MAP

O SISDI-MAP foi projetado para conter processos distintos, atribuindo-lhes funções independentes mas interagindo entre si.

AP, API e MMS são processos distintos que executam suas atividades num contexto multitarefa, mantendo suas autonomias e suas particularidades.

Visando o objetivo final do sistema didático, a seqüência de execução de um determinado serviço pode ser visualizado com o auxílio de um outro processo independente e autônomo: o processo de interface de Operação do usuário (Proc-OP).

Como extensão da implementação da API no SISDI-MAP, a interação API-interface de Operação poderá ser implementada de tal forma que permita diferentes tipos de informação serem transmitidos entre ambos os processos para fins de visualização das atividades internas da API.

5.7 COMUNICAÇÃO API-AP

A comunicação API com AP é feita no SISDI-MAP de duas maneiras distintas: através da execução de um programa de aplicação ou interativamente, através da interface de operação. Em ambos os casos, a API desempenha sua tarefa de maneira independente, retornando ao usuário apenas no final da atividade ou sinalizando-o por códigos de retorno quando da falta de sucesso.

Uma sugestão de continuidade de trabalho seria de se desenvolver programas aplicativos seguindo normas estabelecidas em "Companion Standards", incorporando-lhes as representações de VMD, as funções associadas a um equipamento real; as definições de programas, eventos, semáforos e variáveis relacionadas ao VMD, permitindo versatilidade de configuração e implementação com abordagens diferentes.

5.8 O TESTE DE IMPLEMENTAÇÃO DA API

Não se investiu numa interface para testes interativos da API, com janelas, menus, etc., devido ao fato de se ter especificado para o SISDI-MAP uma interface de operação dosistema SISDI-MAP. Essas atribuições e mais algumas de gerenciamento de sistema são objetos de trabalho de outra tese [T6].

Utilizando-se desta interface, o teste da API poderia ser feito de modo conversacional, inserindo entradas em tempo de execução e visualizando seus resultados na tela.

Porém, o SISDI-MAP também foi especificado para ter processos aplicativos, que, independentemente, fazem chamadas à API durante sua execução.

Optou-se, para fins de teste, em simular um AP, que de forma interativa com a API, fornece suas entradas, definidas estaticamente no programa. As saídas podem ser visualizadas na tela em tempo de execução (foram inseridas, funções de escrita no programa simulador-AP) podendo, opcionalmente, serem jogadas num arquivo em disco ou serem impressas.

Para efeito didático, ambas as formas de execução são apreciáveis.

O teste da interface foi acompanhado dos recursos da própria ferramenta utilizada na implementação: o TURBO-C [F2], [F3], que fornece um ambiente integrado com subsídios de depuração, execução interativa e outros.

5.9 OUTRAS FORMAS DE PROJETAR INTERFACES

Um outro aspecto que poderia ser abordado no tocante à simplicidade de implementação de interfaces é de modelá-la segundo outro modelo de projetos de software, por exemplo: OOD.

Um projeto orientado a objetos caracteriza-se dela definição de **objetos** e **operações**. Objetos provêem mecanismos de representar domínios de informações, enquanto as operações descrevem o processamento associado com o domínio de informações [L7].

Num sistema aberto, consideram-se os objetos endereçáveis,

onde operações são efetuadas sobre estes objetos.

É interessante ainda analisar os ganhos com interfaces projetadas sob outros modelos e verificar o impacto na produtividade de um programador de API's.

5.10 BENEFÍCIOS FINAIS DA INTERFACE

Não resta dúvida que benefícios são alcançados com a incorporação de uma interface entre AP e sistema de comunicação. Particularmente para interface MMS, a redução da complexidade de um AP e de seu "overhead" resultante, são aspectos claramente visíveis de serem alcançados. A padronização exigida na implementação resulta no aprendizado e versatilidade de utilização por parte dos implementadores.

As chamadas de funções de forma facilitada garantem a transportabilidade de sistemas a sistemas, sem grandes riscos de perda de funcionalidade, permitindo a integração dos sistemas abertos.

REFERÊNCIAS BIBLIOGRÁFICAS

DOCUMENTOS ISO:

[I1] ISO/DIS 7498 "IPS-OSI: Basic Reference Model". Outubro/1982.

[I2] ISO/IS 7498-3 "IPS-OSI: Basic Reference Model - Naming and Addressing", Março/1989.

[I3] ISO/DIS 8649/2 "Service Definition for Common Application Service Elements - Association Control". Abril, 1986.

[I4] ISO/DP Proposed EIA RS-511 "Manufacturing Message Specification. Part 1: Service Definition". Draft 5. Junho, 1986.

[I5] ISO/DP Proposed EIA RS-511 1393A "Manufacturing Message Specification. Part 2: Protocol Specification". Draft 5. Junho, 1986

[I5] ISO/DIS 9506 "Manufacturing Message Specification. Part 1: Service Specification". Draft 6. Maio, 1987.

[I6] ISO/DIS 9506 "Manufacturing Message Specification. Part 2: Protocol Specification" Draft 6. Maio, 1987.

[I7] ISO/DIS 8824 "IP-OSI: Specification of Abstract Syntax Notation One (ASN.1)". Agosto, 1986.

[I8] ISO/DIS 8825 "IP-OSI: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ANS.1)". Agosto, 1986.

DOCUMENTOS MAP:

[M1] GM - "Manufacturing Automation Protocol" Versão 3.0. Julho, 1987.

[M2] GM-MAP/TOP - "Application Program Interface - Model Description and Interface Specification Requirements". Dezembro, 1987.

[M3] GM-MAP/TOP - "Application Program Interface - Application Interface Support Functions". Março, 1987.

[M4] GM-MAP/TOP - "Application Program Interface - Connection Management Interface Specification". Novembro, 1987.

[M5] GM-MAP/TOP - "Application Program Interface - MMS Application Interface Specification". Novembro, 1987.

LIVROS TEXTOS:

[L1] Mendes, M.J. "Comunicação Fabril e o Projeto MAP/TOP". Escola Brasileira Argentina de Informática - IV EBAI, 1989.

[L2] Moura, J.A.B; Sauvé, J.P.; Glozza, W.F.; Araújo, J.F.M. "Redes Locais de Computadores - Protocolos de Alto Nível. Avaliação de Desempenho". McGraw-Hill, 1986.

[L3] Glosa, W.F.; Araújo, J.F.M.; Moura, J.A.B.; Sauvé, J.P. "Redes Locais de Computadores - Tecnologia e Aplicação". McGraw-Hill, 1986.

[L4] Halsall, Fre. "Data Communications, Computer Networks and OSI". Addison-Wesley, 1988.

[L5] Dwyer, John; Ioannou, Adrian. "MAP and TOP Advanced Manufcturing Communications. Korgan Page, 1987.

[L6] Tanenbaum, Andrew S. "Computer Networks" Prentice Hall, 1987.

[L7] Pressman, R. "Software Engineering: A Practioner's Approach". McGraw-Hill, 1986.

ARTIGOS:

[A1] Paglioni, A.; Jacintho, D.C.A.; Madeira, E.R.M.; Fernandes, I.A.; Mendes, M.J.; Correa, J.N.; Lima, J.M.S.; Zabeu, M.C.; Sousa, V.L.P. "SISDI-MAP: Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP". Seminário Franco-Brasileiro em Sistemas Distribuídos. Florianópolis, setembro/1989.

[A2] Madeira, E.R.M; Correa, J.N.; Sousa, V.L.P.; Mendes, M.J. "Modelamento de Interfaces de Aplicação e Exemplo de Implementação para protocolo MMS". Anais do SBRC. Trabalho aceito a ser publicado em Campinas, em outubro/1990.

[A3] Madeira, E.R.M.; Mendes, M.J. "Interface de Programas de Aplicação para o Protocolo MMS (RS-511)". Anais do 3º. CONAI. São Paulo, setembro/1988.

[A4] Svobodova, L.; "Implementing OSI Systems", IEEE Journal on selected areas in communications, vol. 7, no. 7, setembro/1989

[A5] Madeira, E.R.M., Mendes, M. J.; "The Application Interface Model for Communication Software and an MMS Protocol Implementation

Example", Productique & Integrations CIM - Integration Aspects -, Bordeaux - France, June/1990

TESES:

[T1] Correa, J.N. "Aspectos de Implementação da Interface dos Programas de Aplicação para o Protocolo MMS e seus Padrões Associados: Gerenciamento de Conexão e exemplo de aplicação", Tese de Mestrado em fase de conclusão - UNICAMP/FEE/DCA

[T2] Zabeu, M.C. "Um Modelo para Configuração de Núcleos para Tempo Real", Campinas - UNICAMP/FEE/DCA. Dezembro, 1989.

[T3] Jacintho, D.C.A. "Aspectos de especificação e implementação da Estrutura de Mensagens de um Sistema Didático do Protocolo MMS", Campinas - UNICAMP/FEE/DCA. Dezembro, 1989.

[T4] Páglioni, A.J.
Tese de mestrado em andamento

[T5] Fernandes, I.A.
Tese de mestrado em andamento

[T6] Lima, J.M.S.
Tese de mestrado em andamento

[T7] Madeira, E.R.M.
Tese de doutoramento em andamento

FERRAMENTAS:

[F1] Kernighan & Ritchie. "The C Programming Language". Addison Wesley, 1984.

[F2] Borland International Inc. "Turbo C-User's Guide". IBM Version, 1987.

[F3] Borland International Inc. "Turbo C-Reference Guide". IBM Version, 1987.

```
/*****************************************************************************
 *                                                                           *
 * ARQUIVO DE DEFINICOES DOS "TIPOS MMS" PARA O SERVICO DE ACESSO A VARIAVEIS *
 *                                                                           *
 *****************************************************************************/


/******************    DEFINICOES DOS TIPOS BASICOS - MMS   ******************/


typedef sint32                          Octet_string;
typedef sint32                          Bit_string;
typedef uint8                           Unsigned;
typedef uint8                           Integer;
typedef uint16                          Floating_point;
typedef sint32                          Visible_string;
typedef char                            Generalized_time;
typedef Boolean                         Binary_time;
typedef uint8                           BCD;

typedef char                            *Identifier;

typedef struct {
                enum Kind_of_object_name                kind_of_object_name;
                union {
                        Identifier                      vmd_specific;
                        struct {
                                Identifier              domain_ID;
                                Identifier              item_ID;
                               } domain_specific;
                        Identifier                      aa_specific;
                      } name_union;
               } Object_name;



/************************     DEFINICOES DE ENUMERADOS     ********************/


enum Kind_of_object_name            {
                                      AA_specific,
                                      Domain_specific,
                                      VMD_specific
                                    };

enum Kind_of_access                 {
                                      list_of_variable,
                                      variable_list_name
                                    };

enum Kind_of_variable               {
                                      named,
                                      unnamed,
                                      single,
```

```
                                scattered,
                                invalidated
                              };

enum Kind_of_address          {
                                numeric_address,
                                symbolic_address,
                                unconstrained
                              };

enum Kind_of_data             {
                                array_data,
                                structure_data,
                                simple_data
                              };


enum Data_access_error        {
                                invalidated_variable,
                                hardware_fault,
                                temporarily_unavailable,
                                object_access_denied
                              };


enum Kind_of_type             {
                                type_name,
                                array_type,
                                structure_type,
                                simple_type
                              };

enum Kind_of_selection        {
                                select_alternate_access,
                                select_access
                              };


enum  Kind_of_access_selection {
                                component,
                                index,
                                index_range
                              };


enum Class_type               {
                                boolean,
                                bit_string,
                                integer,
                                unssigned,
                                floating_point,
                                octet_string,
                                visible_string,
                                generalized_time,
                                binary_time,
```

```
                                  bcd
                                };


enum Kind_of_element_list         {
                                  list_of_var_element,
                                  list_of_data_element,
                                  alternate_access_element,
                                  scatt_acc_descr_element,
                                  list_of_vasso_element,
                                  list_of_access_result_element,
                                  list_of_write_result_element
                                };



/*    DEFINICOES DOS TIPOS ESTRUTURADOS - Servico de Acesso a Variaveis    */


typedef struct array_type
              {
                Boolean                          packed;
                uint32                           num_of_elements;
                struct type_specification        *element_type;
              } Array_type;


typedef struct structure_type
              {
                Boolean                          packed;
                Boolean                          component_name_valid;
                Identifier                       component_name;
                struct type_specification        *component_type;
              } Structure_type;


typedef struct {
                enum Class_type                  class;
                union {
                        Boolean                  size0;
                        Bit_string               size1;
                        Integer                  size2;
                        Unsigned                 size3;
                        Floating_point           size4;
                        Octet_string             size5;
                        Visible_string           size6;
                        Generalized_time         size7;
                        Binary_time              size8;
                        BCD                      size9;
                      } class_union;
              } Simple_type;


typedef struct type_specification
```

```
            {
              enum Kind_of_type                    kind_of_type;
              union {
                      Object_name                  type_name;
                      Array_type                   array_type;
                      Structure_type               *structure_type;
                      Simple_type                  simple_type;
                    } type_spec_union;
            } Type_specification;


typedef struct {
              enum Kind_of_address                 kind_of_address;
              union {
                      uint32                       numeric_address;
                      Visible_string               symbolic_address;
                      Octet_string                 unconstrained;
                    } addr_union;
            } Address;


typedef struct {
              Address                              address;
              Type_specification                   *type_spec;
            } Var_description;


typedef struct {
              enum  Kind_of_access_selection       kind_of_access_selection;
              union {
                      Identifier                   component;
                      uint32                       index;
                      union {
                              struct {
                                       uint32       low_index;
                                       uint32       num_of_elements;
                                     } index_range;
                              uint32                all_elements;
                            } index_range_union;
                    } aces_union;
            } Select_access;


typedef struct {
              Select_access                        *select_access;
              struct Alternate_access              *alter_access [MAX_ELEMENT];
            } Select_alternate_access;


typedef struct { enum  Kind_of_selection           kind_of_selection;
              union
                  { Select_alternate_access        select_alter_access;
                    Select_access                  *select_acces;
                  } select_acc_union;
            } Alternate_access_selection;
```

```
typedef struct Alternate_access
               {
                 Boolean                                 component_name_valid;
                 union
                    { Alternate_access_selection         unnamed;
                      struct
                           { Identifier                  component_name;
                             Alternate_access_selection  alter_acc_selection;
                           } named;
                    } alter_acc_union;
                } Alternate_access;



typedef struct {
                 Boolean                                 component_name_valid;
                 Identifier                              component_name;
                 struct var_specification                *var_spec;
                 Boolean                                 alter_access_valid;
                 Alternate_access                        *alter_access [MAX_ELEMENT];
               } Scattered_access_description;



typedef struct var_specification
               {
                 enum Kind_of_variable                   kind_of_variable;
                 union {
                         Object_name                     var_name;                       /* named */
                         Address                         var_addr;                       /* unnamed */
                         Var_description                 var_descr;                      /* single */
                         Scattered_access_description    *scatt_acc_descr [MAX_ELEMENT]; /* scattered */
                       } var_spec_union;
               } Var_specification;


typedef struct {
                 Var_specification                       *var_spec;
                 Boolean                                 alter_access_valid;
                 Alternate_access                        *alter_access [MAX_ELEMENT];
               } List_of_variable;


typedef struct {
                 enum Kind_of_access                     kind_of_access;
                 union {
                         List_of_variable                list_of_variable [MAX_ELEMENT];
                         Object_name                     list_name;
                       } var_aces_spec_union;
                } Var_access_specification;


typedef struct {
```

```
                enum Class_type                 class;
                union {
                        Boolean                 value0;
                        Bit_string              value1;
                        Integer                 value2;
                        Unsigned                value3;
                        Floating_point          value4;
                        Octet_string            value5;
                        Visible_string          value6;
                        Generalized_time        value7;
                        Binary_time             value8;
                        BCD                     value9;
                      } class_union;
              } Simple_data;


typedef struct data
              {
                enum Kind_of_data               kind_of_data;
                union {
                        struct data             *array_data [MAX_ELEMENT];
                        struct data             *structure_data;
                        Simple_data             simple_data;
                      } data_union;
              } Data;


typedef struct {
                Boolean                         success_result;
                union {
                        enum Data_access_error  data_error;
                        Data                    *data;
                      } acc_result_union;
              } Access_result;


typedef struct {
                Boolean                         success_result;
                enum Data_access_error          data_error;
              } Write_result;




/*****   ESTRUTURAS DAS PRIMITIVAS DE SERVICOS MMS - Acesso a variaveis   *****/


typedef struct {
                Boolean                         spec_with_result;
                Var_access_specification        *var_access_spec;
              } Read_req;


typedef struct {
```

```
                Var_access_specification        *var_access_spec;
                Access_result                   list_of_access_result [MAX_ELEMENT];
              } Read_pos_res;


typedef struct {
                Var_access_specification        *var_access_spec;
                Data                            *list_of_data [MAX_ELEMENT];
              } Write_req;


typedef Write_result                            Write_pos_res [MAX_ELEMENT];


typedef struct {
                Var_access_specification        *var_access_spec;
                Access_result                   list_of_access_result [MAX_ELEMENT];
              } Inforeport_req;


typedef struct {
                enum Kind_of_variable           kind_of_variable;
                union {
                        Object_name             name_attr;
                        Address                 address_attr;
                      } attr_union;
              } Gvattributes_req;


typedef struct {
                Boolean                         mms_deletable;
                Boolean                         named_public;
                Address                         address;
                Type_specification              *type_spec;
              } Gvattributes_pos_res;


typedef struct {
                Object_name                     var_name;
                Address                         var_address;
                Boolean                         type_valid;
                Type_specification              *var_type_spec;
              } Dnvariable_req;


typedef struct {
                int dummy;
              } Dnvariable_pos_res;


typedef struct {
                Object_name                     list_name;
                List_of_variable                list_of_variable [MAX_ELEMENT];
              } Dnvlist_req;
```

```
typedef struct {
                  int dummy;
                } Dnvlist_pos_res;


typedef struct {
                  Object_name                       type_name;
                  Type_specification                *type_spec;
                } Dntype_req;

typedef struct {
                  int dummy;
                } Dntype_pos_res;



/*****************************************************************************/
```

```
/*****************************************************************************
 *                                                                           *
 * ARQUIVO DE DEFINICOES DOS "TIPOS API" PARA O SERVICO DE ACESSO A VARIAVEIS *
 *                                                                           *
 *****************************************************************************/


#include "mms_arq.h"

/***********************     CONSTANTES PREDEFINIDAS     ********************/

#define FALSE                           0
#define TRUE                            1
#define NULL                            0
#define MAX_ELEMENT                     5



typedef uint16                          Return_event_name;

typedef uint16                          Return_code;

/*
typedef uint16                          Connection_id;

typedef uint16                          Invoke_id;
*/



/**********          TIPOS ESPECIFICOS DAS FUNCOES DA API        ***************/
/*                     Servicos de Acesso a Variaveis                        */

typedef enum Kind_of_asso           {
                                      asso_name,
                                      asso_info
                                    };


typedef struct {
                enum Kind_of_asso                           kind_of_asso;
                union {
                       Object_name                          asso_name;
                       struct {
                              Var_access_specification      *var_acc_spec;
                              Type_specification            *type_spec;
                              Address                       address;
                              enum Data_access_error        data_access_error;
                              Data                          *list_of_data [MAX_ELEMENT];
                              Access_result                 list_of_access_result [MAX_ELEMENT];
                              Write_result                  list_of_write_result [MAX_ELEMENT];
                             } vasso_info;
                      } vasso_union;
              } Vasso_specification;
```

```
/**********     Definicoes dos DCB's - Data Control Block's     ***************/


typedef struct {
                 Object_name                        type_name;
                 Array_type                         array_type;
                 Structure_type                     *structure_type;
                 enum Class_type                    class;
                 Boolean                            size0;
                 Bit_string                         size1;
                 Integer                            size2;
                 Unsigned                           size3;
                 Floating_point                     size4;
                 Octet_string                       size5;
                 Visible_string                     size6;
                 Generalized_time                   size7;
                 Binary_time                        size8;
                 BCD                                size9;
               } Type_o_dcb_type;


typedef struct {
                 Identifier                         component_name;
                 uint32                             index;
                 union {
                        struct {
                                 uint32             low_index;
                                 uint32             num_of_elements;
                               } index_range;
                        uint32                      all_elements;
                      } index_range_union;
               } Restriction_o_dcb_type; /* Select_access */


typedef struct {
                 Boolean                            component_name_valid;
                 Identifier                         component_name;
                 Select_access                      *select_access;
                 Alternate_access                   *list_of_alter_access [MAX_ELEMENT];
               } Partialaccess_o_dcb_type; /* Alternate_access */


typedef struct {
                 Object_name                        var_name;
                 Address                            var_addr;
                 Type_specification                 *type_spec;
                 Scattered_access_description       *scatt_acc_descr [MAX_ELEMENT];
               } Vspec_o_dcb_type;


typedef struct {
```

```
                Boolean                         component_name_valid;
                Identifier                      component_name;
                Boolean                         alter_access_valid;
                Alternate_access                *alter_access [MAX_ELEMENT];
              } Scatt_o_dcb_type;


typedef struct {
                List_of_variable                list_of_variable [MAX_ELEMENT];
                Object_name                     list_name;
              } Vaspec_o_dcb_type;


typedef struct {
                Object_name                     asso_name;
                Var_access_specification        *var_acc_spec;
                Type_specification              *type_spec;
                Address                         address;
                enum Data_access_error          data_access_error;
                Data                            *list_of_data [MAX_ELEMENT];
                Access_result                   list_of_access_result [MAX_ELEMENT];
                Write_result                    list_of_write_result [MAX_ELEMENT];
              } Vasso_o_dcb_type;


typedef struct {
                enum Class_type                 class;
                Boolean                         value0;
                Bit_string                      value1;
                Integer                         value2;
                Unsigned                        value3;
                Floating_point                  value4;
                Octet_string                    value5;
                Visible_string                  value6;
                Generalized_time                value7;
                Binary_time                     value8;
                BCD                             value9;
                Data                            *array_data [MAX_ELEMENT];
                Data                            *structure_data;
              } Data_o_dcb_type;


typedef union
              {
                Data                            *elem_list_of_data;
                Vasso_specification             *elem_list_of_vasso;
                List_of_variable                elem_list_of_variable;
                Alternate_access                *elem_alter_access;
                Scattered_access_description    *elem_scatt_acc_descrip;
                Access_result                   elem_list_of_access_result;
                Write_result                    elem_list_of_write_result;
              } MMS_element_list;


typedef struct {
```

```
                enum Kind_of_element_list           kind_of_element_list;
                int                                 indice;
                union
                     { Data                         *list_of_data [MAX_ELEMENT];
                       Vasso_specification          *list_of_vasso [MAX_ELEMENT];
                       List_of_variable             list_of_variable [MAX_ELEMENT];
                       Alternate_access             *alter_access [MAX_ELEMENT];
                       Scattered_access_description *scatt_acc_descr [MAX_ELEMENT];
                       Access_result                list_of_access_result [MAX_ELEMENT];
                       Write_result                 list_of_write_result [MAX_ELEMENT];
                     } list_union;
              } MMS_list;


typedef struct {
                enum Kind_of_element_list           kind_of_element_list;
              } List_o_dcb_type;


typedef struct {
                Boolean                             spec_with_result;
                Vasso_specification                 *var_asso_read;
              } Vread_i_dcb_type;


typedef union
             {
               Access_result                       list_of_access_result [MAX_ELEMENT];
               Error_block                         error;
              } Vread_o_dcb_type;


typedef struct {
                Vasso_specification                 *list_asso_write [MAX_ELEMENT];
              } Vwrite_i_dcb_type;


typedef union  {
                struct {
                         Boolean                    success_result;
                         enum Data_access_error     data_error;
                       } write_result [MAX_ELEMENT];
                Error_block                         error;
              } Vwrite_o_dcb_type;


typedef struct {
                Vasso_specification                 *list_asso_info [MAX_ELEMENT];
              } Ireport_i_dcb_type;


typedef struct {
                Error_block                         error;
              } Ireport_o_dcb_type;
```

```
/********    ESTRUTURAS DAS FUNCOES AUXILIARES - ACESSO A VARIAVEIS     ******/


typedef struct {
                  enum Kind_of_access_selection          kind_of_access_selection;
                  Restriction_o_dcb_type                 nrest_i_dcb;
                  Select_access                          *select_access;
                } Mm_nrestriction;


typedef struct {
                  MMS_list                               *list;
                  MMS_element_list                       *element;
                  List_o_dcb_type                        makelist_i_dcb;
                  MMS_list                               *new_list;
                } Mm_makelist;


typedef struct {
                  enum Kind_of_selection                 kind_of_selection;
                  Partialaccess_o_dcb_type               npartial_acc_i_dcb;
                  Alternate_access                       *alter_access;
                } Mm_npartialaccess;


typedef struct {
                  Var_access_specification               *var_acc_spec;
                  Scatt_o_dcb_type                       scatt_access_i_dcb;
                  Scattered_access_description           *scatt_acc_descr;
                } Mm_nscattaccess;


typedef struct {
                  enum Kind_of_type                      kind_of_type;
                  Type_o_dcb_type                        ntype_i_dcb;
                  Type_specification                     *type_spec;
                } Mm_ntype;


typedef struct {
                  enum Kind_of_variable                  kind_of_variable;
                  Vspec_o_dcb_type                       nvspec_i_dcb;
                  Var_specification                      *var_spec;
                } Mm_nvspec;


typedef struct {
                  enum Kind_of_access                    kind_of_access;
                  Vaspec_o_dcb_type                      nvaspec_i_dcb;
                  Var_access_specification               *var_acc_spec;
                } Mm_nvaccess;
```

```
typedef struct {
                enum Kind_of_asso                   kind_of_asso;
                Vasso_o_dcb_type                    nvasso_i_dcb;
                Vasso_specification                 *var_asso_spec;
              } Mm_nvasso;

typedef struct {
                enum Kind_of_data                   kind_of_data;
                Data_o_dcb_type                     ndata_i_dcb;
                Data                                *new_data;
              } Mm_ndata;

typedef struct {
                Data                                *data_handle;
                enum Kind_of_data                   *kind_of_data;
                Data_o_dcb_type                     *idata_o_dcb;
              } Mm_idata;

typedef struct {
                Vasso_specification                 *var_asso_spec;
                enum Kind_of_asso                   *kind_of_asso;
                Vasso_o_dcb_type                    *ivasso_o_dcb;
              } Mm_ivasso;

typedef struct {
                Var_access_specification            *var_acc_spec;
                enum Kind_of_access                 *kind_of_access;
                Vaspec_o_dcb_type                   *nvaspec_o_dcb;
              } Mm_ivaccess;

typedef struct {
                Var_specification                   *var_spec;
                enum Kind_of_variable               *kind_of_variable;
                Vspec_o_dcb_type                    *ivspec_o_dcb;
              } Mm_ivspec;

typedef struct {
                Type_specification                  *type_spec;
                enum Kind_of_type                   *kind_of_type;
                Type_o_dcb_type                     *itype_o_dcb;
              } Mm_itype;

typedef struct {
                Scattered_access_description        *scatt_acc_descr;
                Var_access_specification            *var_acc_spec;
                Scatt_o_dcb_type                    *scatt_access_o_dcb;
              } Mm_iscattaccess;
```

```
typedef struct {
                 enum Kind_of_selection                *kind_of_selection;
                 Partialaccess_o_dcb_type              *ipartial_acc_o_dcb;
                 Select_access                         *alter_access;
               } Mm_ipartialaccess;


typedef struct {
                 MMS_list                              *list;
                 MMS_element_list                      *element;
                 List_o_dcb_type                       *list_o_dcb;
                 MMS_list                              *new_list;
               } Mm_extractlist;


typedef struct {
                 Select_access                         *select_access;
                 enum Kind_of_access_selection         *kind_of_access_selection;
                 Restriction_o_dcb_type                *irest_o_dcb;
               } Mm_irestriction;




/******    ESTRUTURAS DAS FUNCOES de SERVICOS MMS - ACESSO A VARIAVEIS   *****/


typedef struct {
                 Connection_id                         connection_id;
                 Vread_i_dcb_type                      vread_i_dcb;
                 Vread_o_dcb_type                      *vread_o_dcb;
               } Mm_vread;


typedef struct {
                 Connection_id                         connection_id;
                 Vwrite_i_dcb_type                     vwrite_i_dcb;
                 Vwrite_o_dcb_type                     *vwrite_o_dcb;
               } Mm_vwrite;


typedef struct {
                 Connection_id                         connection_id;
                 Ireport_i_dcb_type                    ireport_i_dcb;
                 Ireport_o_dcb_type                    *ireport_o_dcb;
               } Mm_ireport;


/* * *   OUTRAS FUNCOES DE ACESSO A VARIAVEIS - IMPLEMENTACAO FUTURA    * * *

typedef struct {
```

```
                Connection_id                   connection_id;
                Object_name                     type_name;
                Type_specification              *type_spec;
                Dntype_o_dcb_type               *dntype_o_dcb;
              } Mm_dntype;


typedef struct {
                Connection_id                   connection_id;
                Object_name                     var_list_name;
                List_of_variable                *list_of_variable;
                Dnvlist_o_dcb_type              *dnvlist_o_dcb;
              } Mm_dnvlist;


typedef struct {
                Connection_id                   connection_id;
                Object_name                     var_name;
                Address                         var_address;
                Boolean                         type_valid;
                Type_specification              *var_type;
                Dnvariable_o_dcb_type           *dnvariable_o_dcb;
              } Mm_dnvariable;


typedef struct {
                Object_name                     attr_name;
                Address                         attr_address;
              } Attr_union;


typedef struct {
                Connection_id                   connection_id;
                enum Kind_of_variable           kind_of_variable;
                Attr_union                      attr_union;
                Gvattributes_o_dcb_type         *gvattributes_o_dcb;
              } Mm_gvattributes;

* * *                                                    * * */



/******                DEFINICOES JA EXISTENTES                  *****/
/******         A serem incluidas no arquivo Global_API.h        *****/


typedef enum                    {
                                  api_ap,
                                  api_mms
                                } Tipo_interface;


typedef enum Function_type {
                          mms_nrestriction, mms_irestriction,
```

```
                        mms_makelist, mms_extrlist,
                        mms_npaccess, mms_ipaccess,
                        mms_nsaccess, mms_isaccess,
                        mms_ntype, mms_itype,
                        mms_nvspec, mms_ivspec,
                        mms_nvaccess, mms_ivaccess,
                        mms_nvasso, mms_ivasso,
                        mms_ndata, mms_idata,
                        mms_vread, mms_vwrite, mms_ireport, QUALQUER
                       );


/*====================  Estrutura da Mensagem API-AP  ======================*/

typedef struct {
                Return_event_name          return_event_name;
                Return_code                return_code;
                enum Function_type         funcao;
                union {
                        Mm_nrestriction    mm_nrestriction;
                        Mm_irestriction    mm_irestriction;
                        Mm_makelist        mm_makelist;
                        Mm_extractlist     mm_extractlist;
                        Mm_npartialaccess  mm_npartialaccess;
                        Mm_ipartialaccess  mm_ipartialaccess;
                        Mm_nscattaccess    mm_nscattaccess;
                        Mm_iscattaccess    mm_iscattaccess;
                        Mm_ntype           mm_ntype;
                        Mm_itype           mm_itype;
                        Mm_nvspec          mm_nvspec;
                        Mm_ivspec          mm_ivspec;
                        Mm_nvaccess        mm_nvaccess;
                        Mm_ivaccess        mm_ivaccess;
                        Mm_nvasso          mm_nvasso;
                        Mm_ivasso          mm_ivasso;
                        Mm_ndata           mm_ndata;
                        Mm_idata           mm_idata;
                        Mm_vread           mm_vread;
                        Mm_vwrite          mm_vwrite;
                        Mm_ireport         mm_ireport;
                       } function_type_union;
             } Block_api_ap;


/*============= Estrutura da Fila de Servicos Pendentes ====================*/

typedef struct lista_pen
       {
              uint16                  ae_label;
              Block_api_ap            *mensagem_api_ap_pen;
              struct lista_pen        *next;
       };


/*================ Estrutura da Fila de Servicos Prontos ===================*/
```

```
typedef struct lista
       {
          Tipo_interface            tipo_interface;
          enum Function_type        funcao;
          Invoke_id                 invoke_id;
          union
          {
               Block_api_ap         *mensagem_api_ap;
               Block_struct         *mensagem_api_mms;
          }tipo_fila;
          struct lista              *next;
       };
```