UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

UM MODELO DE INTERFACE DE APLICAÇÃO PARA SISTEMAS DE COMUNICAÇÃO

Por: Edmundo Roberto Mauro Madeira
Orientador: Prof. Dr. Manuel de Jesus Mendes

Tese apresentada à Faculdade de Engenharia Elétrica da UNICAMP, como parte dos requisitos exigidos para a obtenção do título de DOUTOR EM ENGENHARIA ELÉTRICA

Outubro de 1991

UM MODELO DE INTERFACE DE APLICAÇÃO

PARA SISTEMAS DE COMUNICAÇÃO

Edmundo Roberto Mauro Madeira

Outubro de 1991

Ao Lucas,
pelo princípio da onipotência do desejo.

À Adélia,
pela concretização dos meus desejos.

AGRADECIMENTOS

Ao Mendes, pelos desafios propostos e instigante presença que, acredito, continuará após a defesa desta tese.

Ao grupo SISDI-MAP, em especial à Verônica, pela colaboração. E ainda, as muitas contribuições recebidas do amigo Jayme.

À Adélia que, ao partilhar a vida comigo, partilhou as luzes e sombras desta tese.

RESUMO

Este trabalho discute o conceito de APIs ("Application Program Interfaces"), ou seja, Interfaces de Aplicação, para Sistemas de Comunicação. A partir desta discussão, são construídas APIs para diversos protocolos de aplicação, e é proposto um Modelo Geral conceitual.

É também apresentada a implementação da API para o Protocolo MMS ("Manufacturing Message Specification") no SISDI-MAP ("Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP"), desenvolvida no Departamento de Computação e Automação Industrial da Faculdade de Engenharia Elétrica da UNICAMP.

Conceitos existentes em Sistemas de Comunicação relacionados com o de API são analisados: "Socket" do Sistema UNIX, Agente de Usuário dos Protocolos X-400 (correio eletrônico) e de Serviço de Diretório, e "Binding" e Operação Remota do Protocolo ROSE ("Remote Operations Service Element"). Finalmente, é proposto um Modelo de Apresentação de API, que reflete estes conceitos e o Modelo Geral.

ABSTRACT

This work analyzes the concept of API ("Application Program Interface") for communication systems. From this discussion, APIs for several Application Protocols are built, and a conceptual General Model is proposed.

This work also presents the API implementation for the MMS ("Manufacturing Message Specification") Protocol in the SISDI-MAP, a Didactic System for the MAP ("Manufacturing Automation Protocol") Project, developed in the Department of Computing and Industrial Automation of the Faculty of Electrical Engineering at UNICAMP.

Some concepts of communication systems related to the API concept are analyzed: Sockets from the UNIX System, User Agent from the X-400 (eletronic mail) and Directory Service Protocols, and Binding and Remote Operation from the ROSE ("Remote Operations Service Element") Protocol. Finally, a Presentation Model of API is proposed, which considers these analyzed concepts and the General Model presented.

RÉSUMÉ

Dans ce travail nous présentons un modèle général d'une API ("Application Program Interface") pour le logiciel de communication. Quelques aspects du modèle pour plusieurs protocoles d'application sont analysés.

Une implantation de API pour le protocole MMS ("Manufacturing Message Specification"), developpée a l'UNICAMP, est aussi presentée.

ÍNDICE

TABELA DE SIGLAS

ACSE	Association Control Service Element
AE	Application Entity
AI	Application Interface
AP	Application Process
APCF	Application Process Control Function
A-PCI	Application Protocol Control Information
A-PDU	Application Protocol Data Unit
API	Application Program Interface
ASE	Application Service Element
AU	Access Unit
CASE	Common Association Service Element
CCITT	Comité Consultatif International de Télégraphique et Téléphonique
CCR	Commitment Concurrency and Recovery
CEP	Connection End Point
CGMIF	Computer Graphics Metafile Interchange Format
CLP	Controlador Lógico Programável
CM-API	Context Management - Application Program Interface
CNC	Comando Numérico Computadorizado
CSP	Confirmed Service Provider
DCB	Data Control Block
DIB	Directory Information Base
DIS	Draft International Standard
DP	Draft Proposal
DS	Directory Service
DSA	Directory Service Agent
DUA	Directory User Agent
FDT	Formal Description Technique
FTAM	File Transfer Access and Management
GKS	Graphical Kernel System
HLSP	High Level Service Provider

TABELA DE SIGLAS

IF	Interaction Function
IFA	Indication Filter and Arbitrator
IGES	Initial Graphics Exchange Specification
IP	Indication Processing
IS	International Standard
ISO	International Standards Organization
LIB	Library
LLC	Logical Link Control
LOTUS	Language of Temporal Ordering Specification
MACF	Multiple Association Control Function
MAECF	Multiple Application Entity Controlling Function
MAP	Manufacturing Automation Protocol
MAPCF	Multiple Application Process Control Function
MASE	Message Administration Service Element
MHS	Message Handling System
MMS	Manufacturing Message Specification
MOTIS	Message Oriented Text Interchange Systems
MRSE	Message Retrieval Service Element
MS	Message Store
MSSE	Message Submission Service Element
MTA	Message Transfer Agent
MTS	Message Transfer System
NM	Network Management
ODIF	Office Document Interchange Format
OSI	Open Systems Interconnection
PCI	Protocol Control Information
PDIF	Product Data Interchange Format
PDU	Protocol Data Unit
PM	Protocol Machine
P-SAP	Presentation Service Access Point
PSP	Primitive Service Provider
RAOCF	Reception Association Object Control Function
RDA	Remote Database Access
RM-OSI	Reference Model - Open Systems Interconnection
ROSE	Remote Operations Service Element
RTSE	Reliable Transfer Service Element

TABELA DE SIGLAS

SACF	Single Association Control Function
SAO	Single Association Object
SAP	Service Access Point
SASE	Specific Application Service Element
SDU	Service Data Unit
SISDI-MAP	Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP
SP-API	Specific - Application Program Interface
TAOCF	Transmission Association Object Control Function
TCP/IP	Transmission Control Protocol / Internet Protocol
TLI	Transport Layer Interface
TOP	Technical and Office Protocol
TP	Transaction Processing
UA	User Agent
UDP	User Datagram Protocol
UE	User Element
VDRDB	Virtual Device / Real Device Binding
VMD	Virtual Manufacturing Device
VT	Virtual Terminal

INTRODUÇÃO

MOTIVAÇÃO:

Muitos de nós, da área da computação, já passamos pela experiência de desejar trocar um microcomputador de sala, e constatar que a tomada da nova sala não é compatível com o "plug" deste microcomputador. Então, saímos à procura de um adaptador para o "plug", o que muitas vêzes nos custa, além do aborrecimento, muito tempo. Nem sempre encontramos o adaptador conveniente. Algumas vêzes, tendo o "plug" três pinos e a tomada apenas dois contatos, arrumamos um adaptador que usa apenas dois pinos deste "plug", desconsiderando o terceiro, que é o pino terra. Desta forma, não aproveitamos todos os recursos que o microcomputador proporciona, como no exemplo, o efetivo isolamento deste microcomputador.

Quando deseja-se que um programa de aplicação (AP), que utiliza um sistema de comunicação, migre de um ambiente para outro, com um sistema de comunicação similar, defronta-se com um problema semelhante. É necessário ter uma interface entre o AP e o novo sistema de comunicação, para realizar as conversões requeridas. Portanto, é necessário escrever esta nova interface.

Contudo, constata-se o fato que os APs, que utilizam sistemas de comunicação, fazem normalmente muitas chamadas a estes sistemas em comparação às chamadas de funções próprias de seu processamento. Isto aliado ao fato de que a construção desta interface pode não ser simples, porque o novo ambiente pode ser muito diferente, leva à conclusão que em alguns casos é mais econômico reescrever o AP, do que construir uma nova interface. Além disso, como no exemplo inicial, a construção da interface pode desconsiderar funcionalidades do AP ou do sistema de comunicação, se a interface for simplificada. Escrever uma interface própria para o novo ambiente, ou reescrever o AP ?

As duas soluções requerem trabalho, e não são muito elegantes. Uma proposta satisfatória é apresentada, por exemplo, no Projeto

MAP/TOP ("Manufacturing Automation Protocol"/"Technical and Office Protocol"), e consiste em definir uma Interface de Programa de Aplicação (API) genérica [MAP88b], como interface entre o AP e o sistema de comunicação. A API é genérica, no sentido que o seu modelo é geral independente de qual protocolo de aplicação que a API suporte, e no sentido que a API independe de qual sistema operacional que o computador possua, a menos de poucas características bem definidas.

Com esta solução, pode-se resolver o problema da migração de um AP, que utiliza um sistema de comunicação, de um ambiente para outro: é suficiente, que os dois sistemas possuam a API definida pelo Projeto MAP/TOP. Dois problemas podem ser levantados:

a - o novo hospedeiro pode ter um sistema operacional diferente do antigo;

b - o novo sistema de comunicação, apesar de possuir os mesmos protocolos de aplicação, pode apresentar algumas diferenças em relação ao sistema de comunicação antigo.

A API resolve estes dois problemas da seguinte maneira:

- quanto ao primeiro, as duas APIs são diferentes em relação a um módulo da API, que trata de sua interface com o sistema operacional, enquanto que nos outros módulos as duas APIs são iguais;

- quanto ao segundo, as duas APIs podem ser diferentes no tocante às mensagens enviadas/recebidas para o/do sistema de comunicação, mas são iguais em relação à sua interface com o AP, ou seja, a API fornece aos APs as mesmas funções de serviços com os mesmos parâmetros. As diferenças nas mensagens podem ser causadas por diferentes versões dos protocolos de aplicação nos dois sistemas de comunicação.

CONTEXTO DO TRABALHO:

a- Além do problema de portabilidade citado anteriormente, um outro ponto importante na análise de sistemas de comunicação é que estes apresentam normalmente protocolos de aplicação, que oferecem serviços complexos, apesar de terem máquinas de estados simples. Este fato contrasta com o que geralmente ocorre nos protocolos das camadas

mais inferiores do Modelo de Referência para Interconexão de Sistemas Abertos da ISO ("International Standards Organization") RM-OSI/ISO, que possuem máquinas de estados mais complexas e serviços mais simples. A complexidade dos serviços de aplicação ocorre por serem estes em grande número e por seus parâmetros poderem ser de tipos estruturados, em diversos protocolos de aplicação, como por exemplo, nos Protocolos MMS ("Manufacturing Message Specification") e FTAM ("File Transfer Access and Management").

Além disso, atualmente, é comum uma aplicação (AP) requerer diversos protocolos de aplicação de uma maneira interrelacionada. Por exemplo, uma aplicação, ao necessitar os serviços do Protocolo de Acesso às Bases de Dados Remotas, pode requerer também os serviços do Protocolo de Processamento de Transações Distribuídas, se as Bases de Dados são distribuídas.

Portanto, a camada de aplicação pode não oferecer seus serviços aos APs de uma maneira simples. Por outro lado, o AP normalmente deseja que os serviços dos sistemas de comunicação sejam oferecidos de uma maneira simples. Portanto, existe uma lacuna ("gap") a ser diminuída.

Se o sistema de comunicação possuir a API, esta pode suavizar esta lacuna existente. Contudo, o AP ainda deverá manipular serviços complexos, pois afinal, o AP é que tem as diretrizes da comunicação que o mesmo deseja efetuar. Contudo, a API pode oferecer ao AP os serviços de rede de uma maneira amigável ("friendly"). Este trabalho propõe maneiras de suavizar a lacuna comentada, a partir de adições de novas funcionalidades à API;

b- Nos últimos anos houve a definição de diversos protocolos de aplicação: MMS, FTAM, VTP ("Virtual Terminal Protocol"), ACSE ("Association Control Service Element"), CCR ("Commitment Concurrency and Recovery"), TP ("Transaction Processing"), RDA ("Remote Database Access"), DS ("Directory Service") e NM ("Network Management") entre outros, apontando uma "explosão" da camada de aplicação, em contrapartida às outras camadas, que possuindo poucas alternativas de protocolos, têm nestes protocolos as suas estabilidades. Portanto, a API deve refletir a existência de diversos protocolos de aplicação,

não somente dos que já se tornaram padrões internacionais a algum tempo, como por exemplo, os Protocolos MMS e FTAM, mas também daqueles que ainda não se tornaram, como por exemplo, o Protocolo TP.

Para tal, esta tese propõe:

- A API deve ser genérica, no sentido que seu Modelo possa suportar qualquer um destes protocolos de aplicação;

- A API deve ser única, no sentido que se uma aplicação (AP), para serviços típicos de comunicação, desejar invocar serviços de diversos protocolos de aplicação interrelacionados, este AP deve utilizar serviços de uma única API. Esta API deve ter a capacidade de invocar primitivas dos diversos protocolos em questão;

c- O conceito de API (Interface específica pela qual um AP pode acessar um Protocolo de Comunicação) pode ser extrapolado para outras camadas além da de aplicação. A interface "socket" do Sistema UNIX é um exemplo de API para a camada de transporte. Este trabalho sugere a existência de APIs para as camadas de aplicação, transporte e enlace.

QUESTIONAMENTOS:

A partir desta definição de API, esta tese pretende discutir o conceito de Interface de Aplicação. O Projeto MAP/TOP, ao definir a API para obter portabilidade dos APs, introduziu nela outras funcionalidades para facilitar a programação dos APs. Quais são estas funcionalidades ? Quais outras funcionalidades poderiam ser introduzidas ? Como estas funcionalidades são tratadas nas especificações da ISO ("International Standards Organization") ?

Esta tese propõe modelos conceituais de APIs, e analisa os blocos funcionais que compõem estes modelos. Como seriam os modelos de API para os diversos protocolos de aplicação, tais como MMS ("Manufacturing Message Specification"), FTAM ("File Transfer Access and Management"), TP ("Transaction Processing") e RDA ("Remote Database Access") ? Teriam modelos similares ? Como seria um modelo geral de API ?

Esta tese pretende também propor maneiras de implementação. Qual a metodologia de desenvolvimento e implementação de uma API ? Como

seria a sua estrutura de tarefas e procedimentos ? Como seria a sincronização e a comunicação entre as tarefas da API e do sistema de comunicação ? Quais os algoritmos principais da API ? Quais seriam suas estruturas de dados? Como seria sua interface com o sistema operacional ?

Apresenta-se também uma implementação de API para o protocolo de aplicação MMS no SISDI-MAP - Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP - desenvolvido na FEE - Faculdade de Engenharia Elétrica, que serve como base prática para as discussões conceituais.

Esta tese pretende discutir alguns conceitos relacionados e/ou similares ao conceito de API. Quais são estes conceitos ? Quais as vantagens e desvantagens destes conceitos ?

As conclusões sobre as características da API são obtidas da implementação no SISDI-MAP. É vantajoso ter uma API ? Esta oferece vantagens, mas aumenta a complexidade do sistema de comunicação. Como simplificá-la ?

CONTEÚDO:

No capítulo 1 discutem-se inicialmente os conceitos e a terminologia OSI/ISO ("Open Systems Interconnection / International Standards Organization"), com relação aos APs e à camada de aplicação. A partir destes conceitos, introduz-se o conceito de API no ambiente OSI/ISO, mostrando as funcionalidades da API, e como podem se enquadrar nos conceitos OSI/ISO. Além disto, é comentada a utilização de serviços de resposta em sistemas de comunicação.

Na segunda parte deste capítulo (seção 1.3) são analisados os conceitos e a terminologia das APIs do Projeto MAP/TOP. Especificações do MAP/TOP são apresentadas resumidamente. Das especificações, apenas os aspectos fundamentais e os que são pertinentes a este trabalho são analisados. Como as especificações apresentam dúvidas e ambiguidades, interpretações dos conceitos são dadas para exemplos em sistemas abertos.

Ao final, este capítulo descreve as funcionalidades adicionais

que poderiam ser acrescentadas à API do Projeto MAP/TOP (seção 1.4). Desta forma, pode-se construir Modelos Conceituais de API, compostos por blocos funcionais que refletem as funcionalidades discutidas. São construídos Modelos de APIs para os diversos protocolos de aplicação, tais como MMS, FTAM e RDA. É proposto um Modelo Geral de API. É discutido o que muda no Modelo Geral, para ser utilizado por estações clientes e servidoras, pois a maioria dos protocolos de aplicação são assimétricos, sendo constituídos de estações clientes e servidoras.

A terceira parte deste capítulo (seção 1.4), ao lado dos capítulos 2 e 4, forma o corpo desta tese.

O capítulo 2 trata da implementação do Modelo Geral da API, apresentando uma proposta de implementação da API em blocos funcionais. A proposta é modular em relação às funções oferecidas pela API. São comentadas metodologias para o desenvolvimento e implementação da API e de seus blocos funcionais. A seguir, são analisadas a implementação da API em diversos ambientes multitarefas e maneiras de sincronização e comunicação entre as tarefas. Ao final, é discutida a estrutura de dados necessária para a API.

O capítulo 3 apresenta uma implementação de API, segundo o Modelo Geral proposto no capítulo 1, para os Protocolos MMS e ACSE ("Association Control Service Element"), pertencente ao SISDI-MAP. Além do SISDI-MAP, são analisados aspectos de implementação da API: algoritmos principais, estruturas de dados, utilização dos recursos do sistema operacional hospedeiro, sincronismo e assincronismo das chamadas de funções da API, funções auxiliares e alocação de memória, entre outros.

O capítulo 4 descreve alguns conceitos existentes em sistemas de comunicação relacionados com o conceito de API. São analisados os conceitos de "sockets" do sistema operacional UNIX [STE90], Agente de Usuário dos Protocolos X-400 (Correio Eletrônico) ([ISO10021] e [CCITT400]) e de Serviço de Diretório [ISO9594], e "binding" do Protocolo ROSE ("Remote Operations Service Element") [ISO9072]. Vantagens e desvantagens destes conceitos, em relação ao conceito de API, são comentadas. É proposto um Modelo de Apresentação de API, ou seja, um Modelo que define a visão que o AP (Processo de Aplicação)

tem dos serviços oferecidos pela API. Este Modelo é baseado nas discussões dos capítulos anteriores, e na tendência atual dos sistemas de comunicação utilizarem os conceitos comentados neste capítulo.

Ao final, são apresentadas as conclusões deste trabalho, as sugestões para estudo futuro e as tendências dos sistemas de comunicação em relação às APIs. A análise de APIs baseia-se atualmente na discussão de especificações, pois não se conhecem outras implementações em detalhes. As conclusões são baseadas no SISDI-MAP.

O apêndice A apresenta alguns comentários sobre o Projeto MAP/TOP e sobre o Protocolo MMS, pois a API é um conceito MAP/TOP, e o SISDI-MAP implementa o Protocolo MMS. Contudo, não é objetivo deste texto descrever o Projeto MAP/TOP e o Protocolo MMS. No apêndice A pode-se encontrar referências bibliográficas sobre ambos. O apêndice B apresenta as definições de primitivas de requisição, resposta, indicação e confirmação. Estas definições do Modelo OSI/ISO serão muito utilizadas no texto.

O apêndice C apresenta resumidamente a API do Projeto MAP/TOP para o Protocolo FTAM, enquanto que o apêndice D para a Comunicação Privada. O apêndice E apresenta as funções da API do Projeto MAP/TOP para o Protocolo MMS.

O conteúdo deste trabalho originou as seguintes publicações:

[MAD88] MADEIRA, E. R. M. e MENDES, M. J.
"Interface de Programas de Aplicação para o Protocolo MMS (RS-511)"
III CONAI - Congresso Nacional de Automação Industrial, São Paulo, Setembro 1988

[PAG89] PAGLIONI, A. J.; JACINTHO, D. C. A.; MADEIRA, E. R. M.; FERNANDES, I. A.; CORRÊA, J. N.; LIMA, J. M. S.; ZABEU, M. C.; SOUSA, V. L. P. e MENDES, M. J.
"SISDI-MAP: Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP"
Seminário Franco-Brasileiro em Sistemas Informáticos Distri-

buidos, Florianópolis - SC, Setembro 1989

[MAD90a] MADEIRA, E. R. M.; CORRÊA, J. N.; SOUSA, V. L. P. e MENDES, M. J.
"Modelagem de Interfaces de Aplicação e Exemplo de Implementação para o Protocolo MMS"
8o Simpósio Brasileiro de Redes de Computadores, Campinas, Abril 1990

[MAD90b] MADEIRA, E. R. M. e MENDES, M. J.
"Application Interface Model for Communication Software and an MMS Protocol Implementation Example"
Colloque International - CIM90 - Productique et Integrations, Bordeaux, França, Junho 1990

[COR90b] CORRÊA, J. N.; SOUSA, V. L. P., MADEIRA, E. R. M. e MENDES, M. J.
"Aspectos de Programação e Uso do MMS"
I Seminário sobre Redes de Comunicação Industrial - SOBRACON, São Paulo, Setembro 1990

[MAD90c] MADEIRA, E. R. M. e MENDES, M. J.
"An Application Interface Model for Communication Software"
IEEE Global Telecommunications Conference - GLOBECOM'90
San Diego, Estados Unidos, Dezembro 1990

e as notas de apresentação:

[MAD90d] MADEIRA, E. R. M.
"An Application Interface Model for Communication Systems"
Notas de Apresentação - GMD/FOKUS
Berlim, Alemanha, Dezembro 1990

CAPÍTULO 1

MODELAGEM DE INTERFACES DE APLICAÇÃO (APIs)

1.1 - CAMADA DE APLICAÇÃO E PROCESSO APLICATIVO

A discussão sobre a camada de aplicação e os processos aplicativos, apresentada a seguir, é baseada nas especificações da ISO ("International Standards Organization") [ISO1494] e [ISO9545], que descrevem a estrutura interna da camada de aplicação. A especificação [ISO9545] é a sucessora de [ISO1494].

Com a finalidade de descrever o modelo conceitual da camada de aplicação, a especificação [ISO9545] define termos (ou utiliza termos de outros documentos) estritamente relacionados com a camada, como por exemplo: Entidade de Aplicação (AE), Tipo de Entidade de Aplicação, Invocação de Entidade de Aplicação, Elemento de Serviço de Aplicação (ASE), Associação de Aplicação, Contexto de Aplicação, Definição de Contexto de Aplicação, Nome de Contexto de Aplicação e Elemento de Serviço de Controle de Associação (ACSE).

Além destes termos utilizados, a especificação define termos relacionados com o processo de aplicação: Tipo de Processo de Aplicação, Processo de Aplicação (AP) e Invocação de Processo de Aplicação.

Nesta seção pretende-se comentar estes conceitos, analisando suas definições, e interpretando os seus significados para exemplos em sistemas abertos. Nas outras seções deste capítulo e nos demais capítulos utilizam-se estes conceitos.

Um Tipo de Processo de Aplicação é uma especificação de um conjunto particular de funções de processamento de informação. O Processo de Aplicação é uma instância de um Tipo de Processo de Aplicação num sistema aberto real particular. A Invocação de Processo de Aplicação é uma instância de um Processo de Aplicação num sistema aberto real, para realizar suas funções para uma ocasião específica de

processamento de informação.

Uma Entidade de Aplicação (AE) é uma instância de um Tipo de Entidade de Aplicação. Um Tipo de Entidade de Aplicação é uma especificação de um conjunto particular de funções da camada de aplicação. Um exemplo de Tipo de Entidade de Aplicação é aquele em que a AE possui os protocolos MMS ("Manufacturing Message Specification") e ACSE ("Association Control Service Element"). Cada instância deste tipo é uma AE.

Uma Invocação de Entidade de Aplicação realiza as funções de parte ou de todas as capacidades de uma Entidade de Aplicação para uma ocasião particular de comunicação. Para o exemplo citado, uma Invocação de Processo Aplicativo pode invocar a AE (criando uma Invocação de AE) para acessar o protocolo MMS. De certa maneira, a Invocação de AE representa a Invocação de Processo Aplicativo nos aspectos relativos à comunicação.

Os Elementos de Seviço de Aplicação (ASEs) são os componentes de uma Entidade de Aplicação. No exemplo, os protocolos MMS e ACSE são exemplos de ASEs.

Uma Associação de Aplicação é uma relação cooperativa entre duas Invocações de Entidades de Aplicação. Esta relação é baseada na troca de A-PCIs (Application Protocol Control Information), usando o serviço de apresentação. No exemplo, a Invocação de Processo Aplicativo, usando a Invocação de AE, pode estabelecer diversas Associações de Aplicação para comunicar com diversas Invocações de Processos de Aplicação em computadores remotos, utilizando o protocolo MMS.

O Contexto de Aplicação é o conjunto de regras que governa o comportamento de duas Invocações de AE numa Associação de Aplicação. A Definição de Contexto de Aplicação é a descrição estática de um Contexto de Aplicação numa linguagem. O Nome de Contexto de Aplicação é o nome que identifica uma Definição de Contexto de Aplicação de modo não ambíguo. No exemplo, o Nome de Contexto de Aplicação das associações poderia ser "MMS".

O Elemento de Serviço de Controle de Associação é um ASE, que provê meios para o estabelecimento e o término de todas as Associações de Aplicação . No exemplo, o Elemento de Serviço de Controle de

Associação é o protocolo ACSE.

EXEMPLO:

Para exemplificar os termos definidos anteriormente, considera-se o caso simples de uma célula flexível da manufatura, como ilustra a figura 1.1.

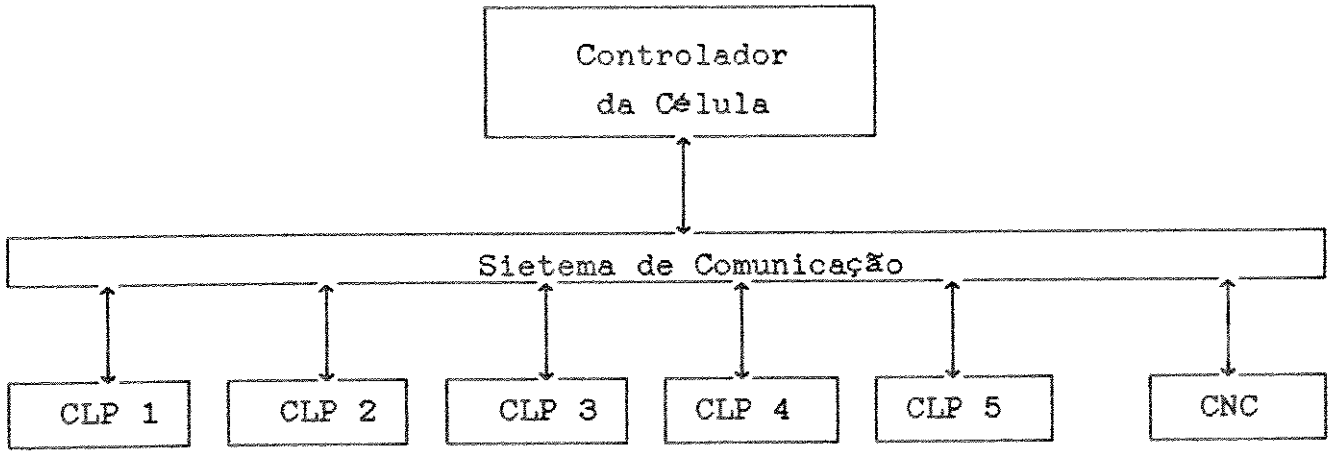


Figura 1.1 - Célula Flexível da Manufatura

A célula é composta por diversos dispositivos: Estação Controladora de Célula, Comando Numérico (CNC) e Controladores Lógicos Programáveis (CLPs). Tem-se tarefas diferentes nos vários dispositivos.

O conjunto destas tarefas numa estação forma um Tipo de Processo de Aplicação. Cada tarefa, em particular, corresponde a um Processo de Aplicação. A Estação Controladora da Célula pode ter uma tarefa para monitorar um CLP, e uma tarefa para monitorar o CNC. O conjunto destas duas tarefas forma o Tipo de Processo de Aplicação da Estação Controladora da Célula. Cada uma das duas tarefas corresponde a um Processo de Aplicação (AP). Portanto, cada estação possui um ou mais Processos de Aplicação.

Numa estação, um Processo de Aplicação pode ser ativado uma ou mais vêzes. Cada instância do Processo de Aplicação, assim obtida,

corresponde a uma Invocação do Processo de Aplicação. A Estação Controladora da Célula pode, por exemplo, para algum tipo de aplicação, ativar o Processo de Aplicação de monitoração do CLP duas vêzes, obtendo duas instâncias do Processo de Aplicação, uma instância para monitorar o CLP-1, e a outra para monitorar o CLP-2. Portanto, tem-se duas Invocações do Processo de Aplicação de monitoração de CLP na Estação Controladora da Célula.

Uma outra interpretação para os conceitos mencionados anteriormente é que Processos de Aplicação de mesmo tipo tenham o mesmo conjunto específico de funções. Suponha-se no exemplo, que o CLP-3 seja servidor do CLP-1, e que o CLP-4 seja servidor do CLP-2. A tarefa no CLP-1 para monitorar seu servidor pode ser igual à tarefa no CLP-2 para monitorar o servidor dele. Portanto, estas duas tarefas têm mesmo Tipo de Processo de Aplicação, sendo cada tarefa um Processo de Aplicação. Se o CLP-1 também é cliente do CLP-5, o CLP-1 pode utilizar a mesma tarefa (Processo de Aplicação) para controlar o CLP-3 e o CLP-5, podendo criar uma instância da tarefa para cada monitoramento. Cada instância é uma Invocação do Processo de Aplicação.

A primeira interpretação é a apresentada em [ISO1494], e a segunda em [ISO9545]. Este trabalho utiliza a segunda interpretação.

RELAÇÃO ENTRE INVOCAÇÃO DE AP E INVOCAÇÃO DE AE:

Uma Invocação de Processo de Aplicação pode ter uma ou mais, ou não ter, Invocações de Entidades de Aplicação de mesmo tipo, ou de tipos diferentes, e é desta forma que se realizam as interações entre Invocações de Processos de Aplicação num ambiente OSI. Para qualquer Invocação de AE existe uma, e somente uma, Invocação de AP associada (Figura 1.2).

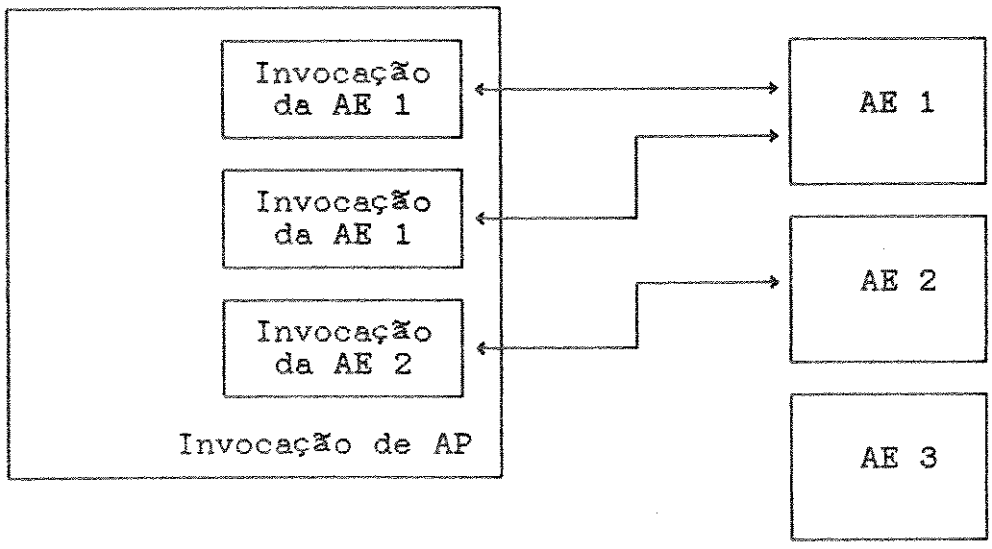


Figura 1.2 - Relação entre Invocação de AP e Invocação de AE

Uma mesma Invocação de AP pode invocar mais de uma vez uma mesma AE, em algumas situações:

a - Para realizar tarefas de comunicação logicamente independentes;

b - O número de Associações de Aplicação por Invocação de AE e o número de Invocações por AE são limitados em função dos recursos disponíveis (por exemplo, em função do tamanho das estruturas de dados) para as AEs (e mesmo para as APIs, como será visto neste capítulo). Uma Invocação de AP pode ter uma Invocação de AE, sendo que esta possui todas as associações possíveis estabelecidas. Além disso, a AE pode ter , neste momento, esta única invocação. Se esta Invocação de AP desejar ter uma nova associação, não pode tê-la na mesma Invocação de AE, pois não existe mais recurso para tal. Contudo, há recurso disponível para novas Invocações à AE. Neste caso, a Invocação de AP pode invocar uma segunda vez a AE, e obter uma nova Associação de Aplicação através dela.

IMPLEMENTAÇÃO DAS AEs:

Quanto à implementação das AEs, pode-se implementar um SASE

(Specific ASE), que corresponde a um protocolo específico de aplicação, numa AE ou em várias AEs, ou vários SASEs numa mesma AE.

UM PROTOCOLO - UMA AE:

É o caso do Projeto MAP/TOP, que impõe um único SASE por AE. Por exemplo, na camada de aplicação pode-se ter o MMS numa Entidade de Aplicação e o FTAM ("File Transfer Access and Management") em outra Entidade de Aplicação (Figura 1.3).

A vantagem é que tem-se a camada de aplicação estruturalmente bem construída. Para alterar um protocolo de aplicação, deve-se alterar uma única Entidade de Aplicação.

A desvantagem deste esquema é que, como uma associação de aplicação pertence a uma Invocação de AE, a Invocação de AP só pode, através desta associação, solicitar serviços de um protocolo de aplicação. Assim, no exemplo, se a Invocação de AP desejar solicitar serviços MMS e FTAM, a Invocação de AP deve invocar as duas Entidades de Aplicação, e solicitar associações distintas.

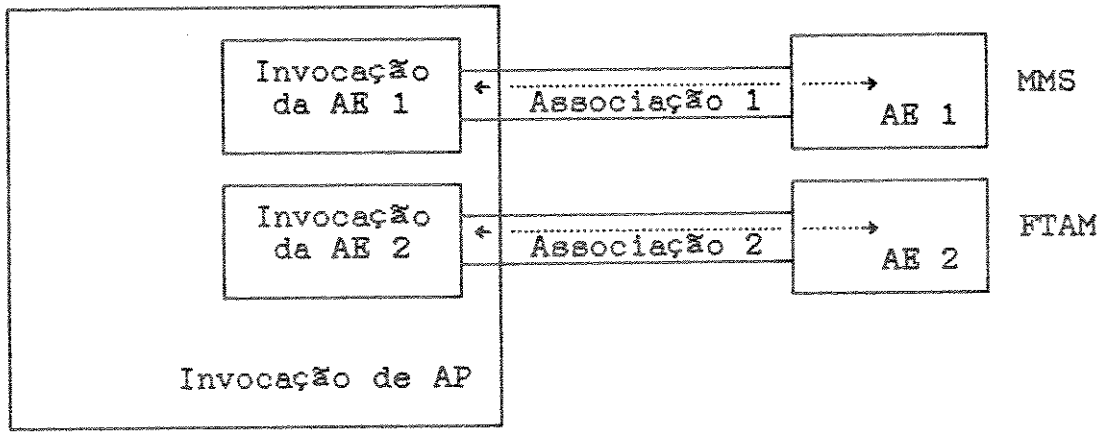


Figura 1.3 - Um Protocolo / Uma AE

UM PROTOCOLO - DIVERSAS AEs:

Com este esquema, por exemplo, o protocolo de aplicação MMS pode ser implementado através de duas AEs: uma oferecendo serviços das classes de Gerenciamento de Contexto e Acesso às Variáveis, e outra oferecendo serviços das demais classes além da classe de Gerenciamento de Contexto (Figura 1.4).

A vantagem é que em sistemas de comunicação com diferentes graus de complexidade das estações, pode-se implementar apenas parte dos protocolos de aplicação nas diversas estações. Em algumas estações pode-se ter implementado somente a AE-1, em outras apenas a AE-2 e em outras a AE-1 e a AE-2. Este esquema corresponde a uma implementação modular do protocolo de aplicação.

A desvantagem é que uma associação pertencente a uma Invocação a uma AE só pode realizar serviços daquela AE, e consequentemente, daquelas classes do protocolo de aplicação definido na AE. No exemplo, se uma Invocação de AP deseja solicitar serviços de classes distintas implementadas em AEs diferentes, a Invocação de AP deve invocar as duas AEs, e solicitar associações distintas. Como esta restrição é forte, este esquema não é normalmente utilizado.

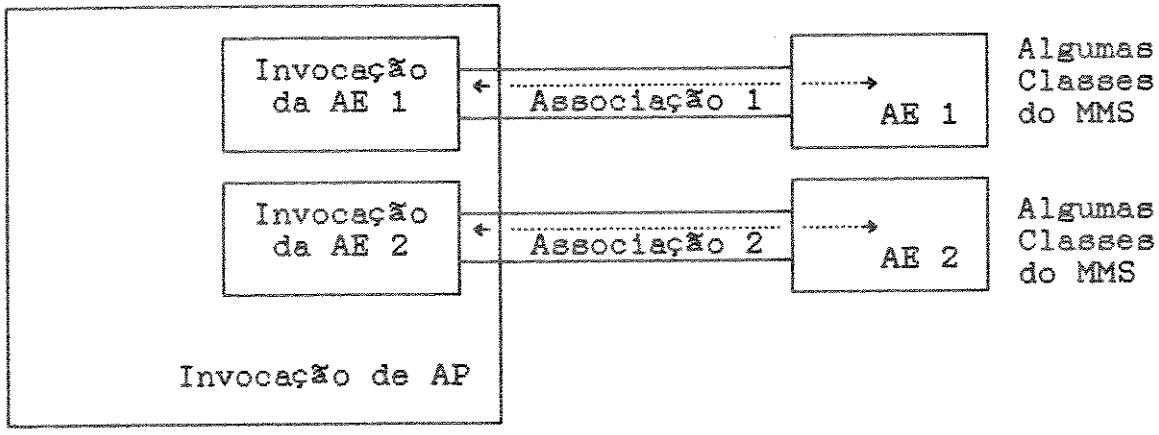


Figura 1.4 - Um Protocolo/ Diversas AEs

UMA AE - DIVERSOS PROTOCOLOS DE APLICAÇÃO

Com este esquema, por exemplo, a camada de aplicação pode ter apenas uma AE, que contém os protocolos MMS e FTAM (Figura 1.5).

A vantagem é que uma associação pertencente a uma Invocação à AE pode realizar serviços de todos os protocolos de aplicação que a AE possui. No exemplo, numa associação pode-se executar serviços MMS e FTAM.

A desvantagem deste esquema é que a Invocação do AP, quando desejar solicitar serviços de aplicação, invoca a AE, e se a Invocação do AP desejar utilizar apenas serviços de um protocolo de aplicação, assim mesmo, ele terá ativado uma AE que possui mais de um protocolo. No exemplo, se a Invocação de AP desejar utilizar seviços MMS, a Invocação de AP terá que ativar uma AE que possui o MMS e o FTAM.

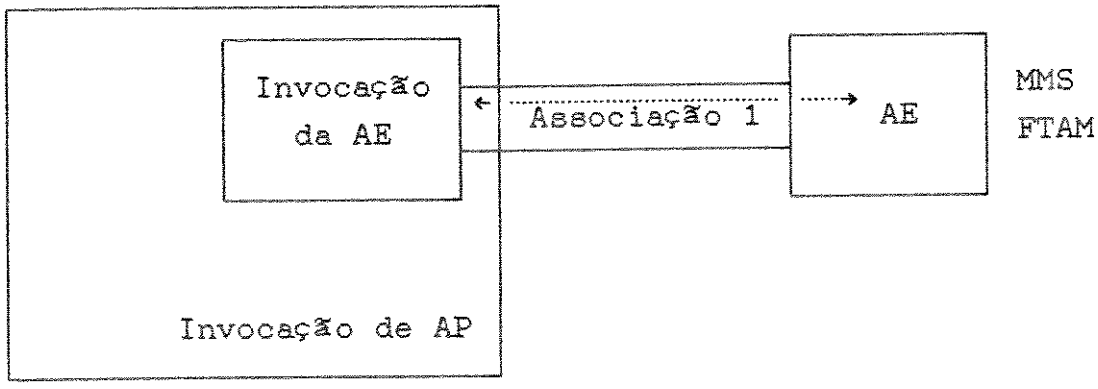


Figura 1.5 - Uma AE / Diversos Protocolos de Aplicação

Note-se que, independentemente da opção de implementação adotada, as Entidades de Aplicação deverão ter, além dos SASEs (conforme opção adotada), um ou mais CASEs ("Common Association Service Elements"), que são ASEs independentes da aplicação.

EMPREGO DO TERMO INVOCAÇÃO DE AP:

Nos demais capítulos e seções deste texto, para o conceito de Invocação de Processo de Aplicação, utiliza-se apenas o termo Processo de Aplicação. Optou-se por esta utilização, pois no jargão de Sistemas

de Computação este emprego já ocorre.

1.2 - INTERFACE DE APLICAÇÃO: ASPECTOS GERAIS

1.2.1 -DEFINIÇÃO NO AMBIENTE OSI/ISO

O documento [ISO1494] da ISO, que trata da estrutura da camada de aplicação, define uma Função de Interação (IF) pertencente ao Processo de Aplicação (AP), como responsável pela cooperação entre Invocações de Processos de Aplicação (Figura 1.6). No âmbito do AP, somente as características (serviços oferecidos) da Função de Interação interessam. Os mecanismos específicos para a realização de tais serviços não são pertinentes.

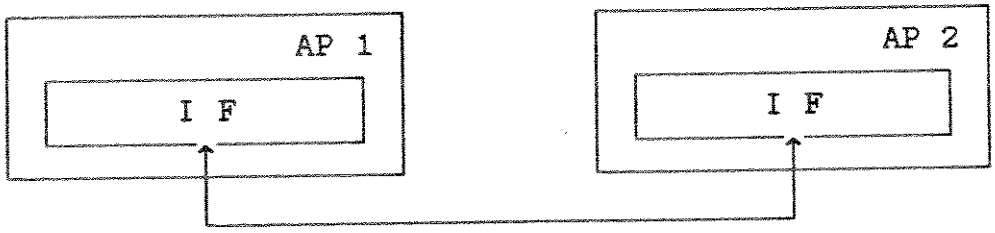


Figura 1.6 - Interação entre APs

As IFs devem oferecer serviços durante toda a associação entre as Invocações de Processos de Aplicação, incluindo:

a - Gerenciamento de conexão, se a associação for orientada à conexão (CASEs);

b - Serviços oferecidos pelos diferentes protocolos de aplicação específicos (SASEs).

Por outro lado, a ISO, no documento [ISO7498], define o UE - Elemento do Usuário (User Element) pertencente à camada de aplicação, como responsável pelo processamento de informação associado funcionalmente a um AP (Figura 1.7). Isto é representado em [ISO1498] pelo AP (através da IF), a menos dos mecanismos específicos para a realização dos serviços oferecidos pela IF. Para o UE interessa como o acesso aos protocolos de aplicação é efetuado. O UE também é

construído para:

a - Gerenciamento de Conexão (CASEs);

b - Protocolos de Aplicação específicos (SASEs).

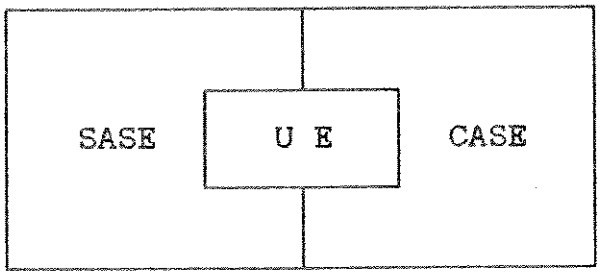


Figura 1.7 - UE na Camada de Aplicação

INTERFACE DE APLICAÇÃO:

Portanto, pode-se retirar o conceito de IF do AP, e o conceito de UE da camada de aplicação, e introduzir entre o AP e a camada de aplicação um novo conceito, o de Interface de Aplicação (AI) (Figura 1.8).

A Interface de Aplicação teria a funcionalidade da IF e os mecanismos do UE. A Interface de Aplicação ofereceria serviços de rede ao AP, e através de mecanismos próprios solicitaria para a camada de aplicação o envio de A-PDUs (Application Protocol Data Units), e vice-versa, ou seja, receberia A-PDUs da camada de aplicação, e as entregaria ao AP como resposta aos serviços solicitados anteriormente.

Neste caso, têm-se também serviços da AI para:

a - Gerenciamento de Contexto;

b - Protocolos de Aplicação específicos.

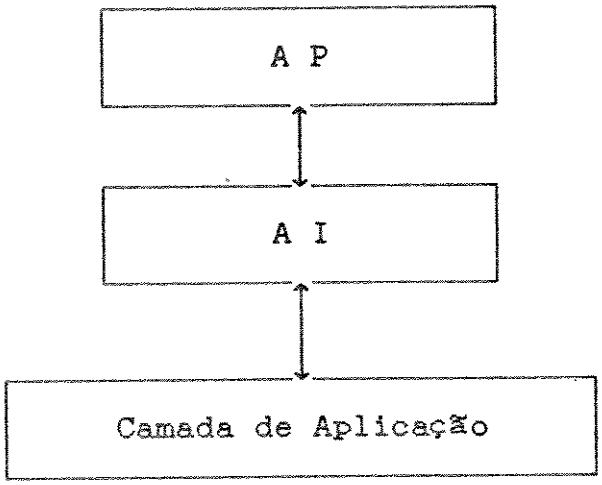


Figura 1.8 - Interface de Aplicação

As principais vantagens dessa Interface de Aplicação são:

a - As Funções de Interação estão relacionadas com os APs (qual a função que o AP deseja executar), e os UEs com as AEs (qual o mecanismo de comunicação com as AEs), mas não são APs ou AEs. Isto justifica a construção de um novo bloco funcional entre os APs e a camada de aplicação;

b - APs diferentes podem ter mesmas Funções de Interação. Portanto, ao introduzir as funcionalidades das Funções de Interação numa Interface de Aplicação, que pode ser única para os diversos APs, não se tem duplicação dessas funções.

1.2.2 - DEFINIÇÃO NO PROJETO MAP/TOP: API

A API - Interface de Programa de Aplicação - do Projeto MAP/TOP é uma interface entre o AP - Processo de Aplicação - e o sistema de comunicação. Se o sistema é baseado na arquitetura de sete camadas do RM OSI/ISO - Modelo de Referência para Interconexão de Sistemas Abertos da International Standard Organization [ISO7498] - a interface comunica com o nível mais alto da hierarquia, ou seja, com a camada de aplicação. O Modelo de Referência OSI/ISO é comentado em [MEL86a] e [MEL86b].

A figura 1.9 mostra as interações entre o AP, a API e o sistema de comunicação, representado na figura apenas pelas camadas de aplicação e apresentação. A camada de aplicação é composta por um ou mais CASEs - "Common Association Service Elements" - para o controle das associações, e por um ou mais SASEs - "Specific Application Service Elements" - que correspondem aos protocolos específicos de comunicação. Exemplos de SASEs são: MMS - Manufacturing Message Specification ([ISO9506a] e [ISO9506b]) e FTAM - File Transfer Access and Management [ISO8571], entre outros. A API é um conceito definido pelo Projeto MAP/TOP [MAP88b], e é comentada em [MAD88] e [RUP89].

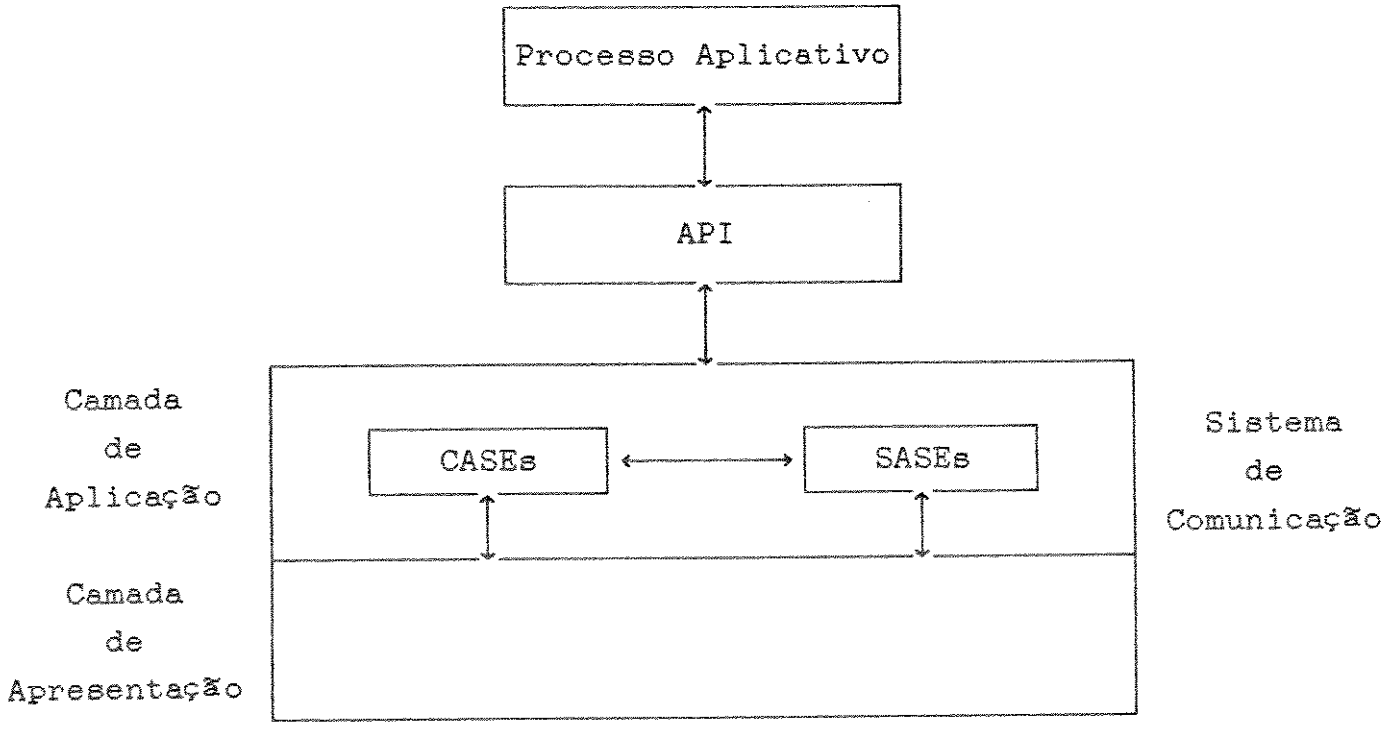


Figura 1.9 - API - Interface de Programa de Aplicação

Um dos objetivos da API é o de prover uma portabilidade maior para os APs. A portabilidade do sistema de comunicação ocorre em dois níveis:

a - Um mesmo sistema de comunicação pode ser executado em ambientes distintos (definidos por sistemas operacionais de tempo real

diferentes) com o auxílio de uma pequena interface;

b - Um mesmo AP pode utilizar o mesmo sistema de comunicação em máquinas distintas, desde que possua a interface apropriada.

A API tem a funcionalidade da interface (b), e requer características mínimas do sistema operacional, para que a API funcione também como a interface (a).

A API possui também como objetivos, além de prover portabilidade aos APs através da padronização de uma biblioteca de funções:

- decrementar o custo de programação e treinamento, pois os programadores podem usar a interface em ambientes múltiplos;

- executar algumas tarefas, que seriam normalmente de responsabilidade do usuário, tais como: verificar se alguns valores de parâmetros dos serviços oferecidos pertencem ao intervalo de valores válidos, preencher parâmetros "defaults", e dividir serviços complexos em serviços mais simples, entre outros.

A API, como interface entre o AP e o serviço de rede, deve conter:

a - Uma biblioteca de funções, que serão chamadas pelos APs para requisitar os serviços desejados da rede;

b - Um provedor de serviços de primitivas para enviar e receber primitivas para/da camada de aplicação;

Além disso, deve conter procedimentos de controle:

c1 - Um provedor de serviços de alto nível para dividir serviços complexos em diversos serviços mais simples;

c2 - Um provedor de serviços confirmados para controlar a recepção de confirmações;

c3 - Um filtro para recepção de indicações para filtrar as indicações de serviços esperados e autorizados.

A figura 1.10 mostra a interação entre o AP, as três partes da API e o serviço de aplicação do sistema de comunicação. Na figura, o AP usa os serviços da rede, chamando uma função da biblioteca com os parâmetros apropriados. A API gera em seguida a requisição para a camada de aplicação enviar a primitiva correspondente ao serviço de rede pedido.

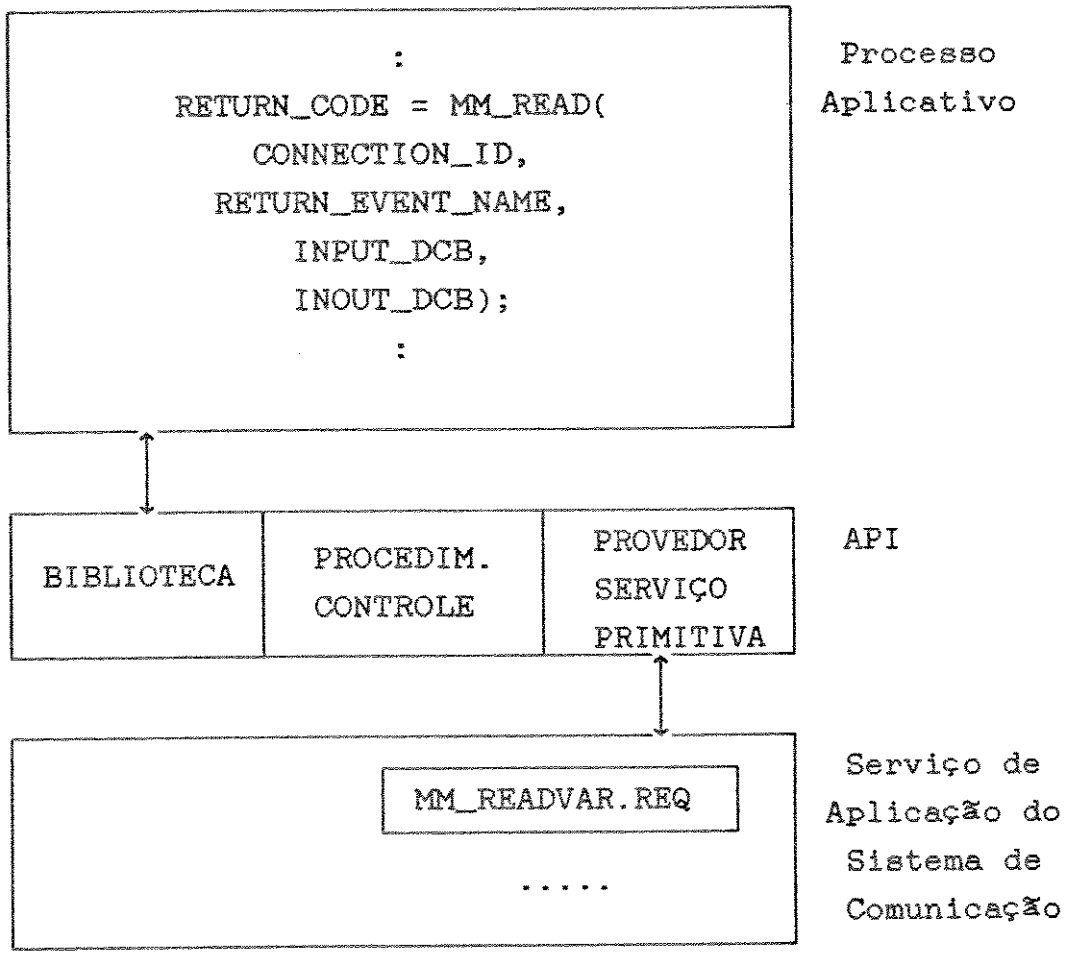


Figura 1.10 - Utilização da API

No exemplo, baseado no protocolo de aplicação MMS, o AP deseja ler o conteúdo de certas variáveis num dispositivo remoto. Para tal, o AP chama a função MM_READ da biblioteca da API (MM por ser a interface MMS, e READ por estar associada ao serviço READ do MMS). Parâmetros apropriados são passados. Como resultado desta chamada, a camada de aplicação é requisitada para enviar uma primitiva de requisição de leitura de variável remota (MM_READVAR.req do MMS). Além disto, a API retorna ao AP informação sobre o sucesso ou não da chamada, que é atribuída, no exemplo, à variável RETURN_CODE do AP.

O Projeto MAP/TOP define APIs para os protocolos MMS [MAP88e], FTAM [MAP88a] e comunicação privada [MAP88f].

CARACTERÍSTICAS DA API:

A API é independente do sistema operacional com a restrição de que este providencie um ambiente multitarefa, que suporte a comunicação e coordenação entre processos.

O modelo permite a realização de chamadas síncronas, em que o AP fica bloqueado até que a API receba a primitiva de confirmação correspondente à função. O modelo permite também a realização de chamadas assíncronas, em que o AP é liberado para continuar a sua execução, assim que a API envia a primitiva de requisição ou resposta correspondente para a camada de aplicação. Um mecanismo de gerenciamento de eventos é definido para padronizar o gerenciamento das chamadas assíncronas.

As funções da API podem ser de alto nível, baixo nível ou respondedoras. Para uma função de baixo nível somente uma primitiva de serviço do protocolo de aplicação é invocada para cada chamada. Para uma função de alto nível várias primitivas de serviço do protocolo de aplicação são invocadas para cada chamada. A chamada de uma função respondedora prepara a API para receber determinadas primitivas de indicação. Desta forma, a API não interrompe o AP para notificar a ocorrência de um evento, como no caso da chegada de uma primitiva de indicação, sem que o AP tenha previamente solicitado.

O apêndice C apresenta um exemplo de função de alto nível para o Protocolo FTAM, e mostra como esta função invoca diversas primitivas do FTAM para realizar o serviço solicitado.

O AP, para informar a API, que pretende solicitar serviços de uma determinada AE - Application Entity (Entidade de Aplicação), único objeto que pode ser endereçado num ambiente OSI ("Open System Interconnection"), pede uma ativação da AE para a API, que cria uma invocação da AE. Uma invocação da AE é uma relação entre determinado AP e determinada AE através da API. Utilizando uma invocação de AE, um AP pode possuir diversas associações (conexões) com APs remotos. A API opera num ambiente, em que múltiplas invocações de uma AE são possíveis, sendo que cada invocação a uma AE pode ter uma ou mais conexões.

1.2.3 - API x INTERFACE DE APLICAÇÃO NO AMBIENTE OSI/ISO

O conceito de API - Application Program Interface - introduzido pelo Projeto MAP/TOP (comentado na seção anterior) [MAP88b], engloba o conceito de Interface de Aplicação definido no ambiente OSI/ISO (AI) (comentado na seção 1.2.1), mas é mais geral em alguns aspectos:

a - Introduz mais funcionalidades à Interface:

- define o conceito de serviço de alto nível (seqüência de invocações de serviços dos protocolos de aplicação), e monitora a seqüência de invocações dos serviços mais simples que compõem o serviço de alto nível;
- define o conceito de serviço confirmado, monitorando não só a requisição do serviço, como também a sua confirmação;
- define o conceito de serviço respondedor, onde a API se prepara para receber requisições do par remoto.

b - Incrementa os serviços oferecidos pela Interface de Aplicação. Além dos serviços de Gerenciamento de Conexão e dos Protocolos de Aplicação específicos, define serviços:

- Auxiliares, para construção e interpretação de dados e resultados dos outros serviços;
- de Gerenciamento de Eventos, para poder utilizar os demais serviços de maneira síncrona ou assíncrona;

c - Tem objetivos mais gerais, como por exemplo, o de alcançar uma maior portabilidade do sistema de comunicação, e o de realizar algumas tarefas que seriam próprias do usuário (AP).

Neste texto, apesar do termo API ser usado pelo Projeto MAP/TOP e o termo Interface de Aplicação pertencer ao jargão de sistemas de comunicação, estes dois termos serão utilizados com o mesmo significado, pois estes o têm.

1.2.4 - FUNCIONALIDADES

INVOCAÇÃO DE ENTIDADE DE APLICAÇÃO:

O documento [ISO9545] da ISO no anexo B apresenta o modelo de uma Invocação de Entidade de Aplicação num instante particular (Figura 1.11).

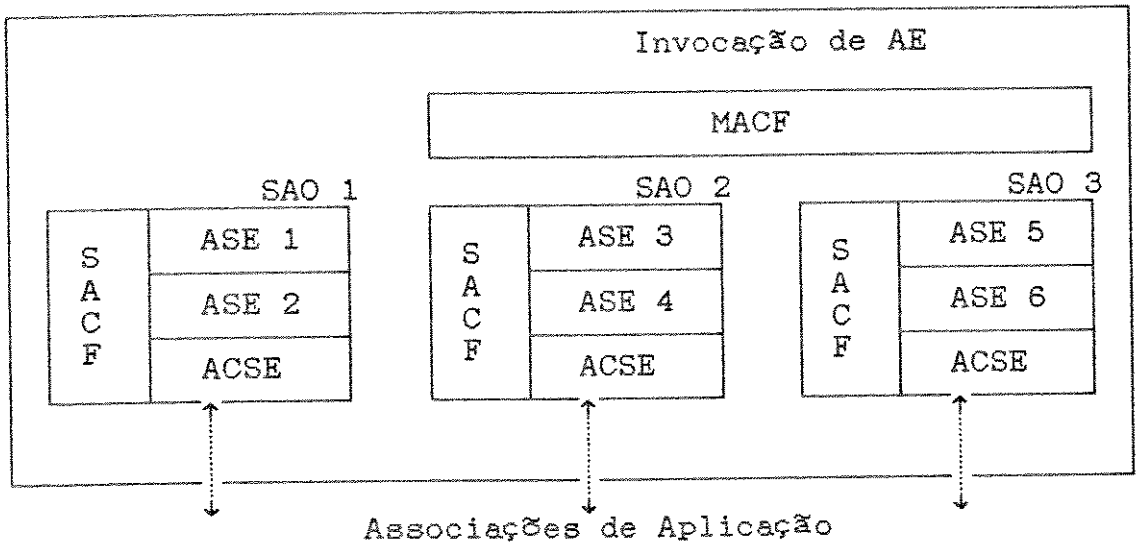


Figura 1.11 - Modelo de Invocação de AE

Neste modelo surgem os conceitos de:

a-) SAO: Single Association Object (Objeto da Associação única);

b-) SACF: Single Association Control Function (Função de Controle da Associação única);

c-) MACF: Multiple Association Control Function (Função de Controle de Associações múltiplas).

O Objeto da Associação única (SAO) define qual subconjunto das capacidades de comunicação da Entidade de Aplicação vai ser utilizado na associação. Por exemplo, determina quais os ASEs da AE que serão utilizados nesta associação. O SACF é responsável pela função de coordenação dos diversos ASEs numa associação, governando as interações segundo as regras do Contexto de Aplicação. O SACF é um componente do SAO.

O SAO e o SACF numa associação são responsáveis por:

a - Selecionar os ASEs, que serão utilizados na associação.

É um subconjunto dos ASEs que a Invocação da AE possui. Esta seleção pode ser dinamicamente alterada durante a associação;

b - Oferecer serviços dos ASEs selecionados;

c - Gerenciar a utilização dos diversos CASEs pelos SASEs. Para estabelecer ou liberar uma associação é necessário que CASEs e SASEs estejam de acordo. Esta gerência implica em troca de mensagens entre os ASEs;

d - Gerenciar o relacionamento entre os CASEs. Por exemplo, para o processamento de Transações Distribuídas [ISO10026] necessita-se dos CASEs: ACSE ([ISO8649] e [ISO8650]) e CCR ("Commitment Concurrency and Recovery") ([ISO9804] e [ISO9805]); e para o Acesso à Base de Dados Remota [ISO9579] necessita-se dos CASEs: ROSE ("Remote Operation Service Element") [ISO9072], ACSE e CCR;

As funções (c) e (d) significam, que o SACF assegura, que todas as regras de sequenciamento de envio/recebimento de primitivas numa associação sejam seguidas;

e - Construir tabelas, para controlar os recursos de uma associação. Associações orientadas à conexão devem ter a informação de todos os serviços requisitados, e ainda não respondidos;

f - Auxiliar a programação do Árbitro e Filtro de Indicações (IFA), para filtrar as primitivas de indicação recebidas em cada associação;

g - Gerenciar o Endereço de Apresentação. Para uma associação vai ser definido o seu CEP ("Connection End Point"), a ser utilizado durante a conexão dentro do P-SAP ("Presentation Service Access Point"), que relaciona a Entidade de Aplicação em questão e a Entidade de Apresentação a ser utilizada. Se a associação for sem conexão, será utilizado apenas o P-SAP.

As funcionalidades (a), (b), (c), (d), (e) e (f) podem ser executadas em parte pela API, facilitando o processamento da camada de aplicação. Desta forma, a API oferece serviços ao usuário de uma maneira mais amigável e conveniente, tornando mais transparente o relacionamento entre SASEs e CASEs na camada de aplicação.

O MACF é responsável pela coordenação das atividades de comunicação em associações separadas. O MACF governa as interações

entre SAOs numa invocação de AE. As funções do MACF podem em grande parte ser executadas pela API, pois estas não estão propriamente relacionadas com os ASEs, mas sim com o controle das associações. Estas funções podem ser vistas como mais próximas do usuário (AP) do que dos protocolos de aplicação.

O MACF numa Invocação de Entidade de Aplicação tem as seguintes funções:

a - Oferecer serviços dos ASEs selecionados nas diversas associações num ambiente de múltiplas associações interrelacionadas numa Invocação de Entidade de Aplicação;

b - Auxiliar o controle do número máximo de associações por Invocação de Entidade de Aplicação. Este limite é necessário para que se possa compartilhar os recursos disponíveis;

c - Gerenciar a seqüência de envio/recebimento de primitivas dos diversos ASEs num ambiente de múltiplas associações interrelacionadas. [JOH90] apresenta um exemplo para o TP-ASE ("Transaction Processing - ASE"): o MACF interage com o TP para garantir que uma requisição de "Commit" (efetivação) seja enviada somente se todos os nós participantes, em todas as associações da árvore de transações, tenham votado "Ready" (pronto), para uma requisição apropriada enviada anteriormente;

d - Auxiliar a construção de tabelas, para gerenciar os recursos das diversas associações. Associações orientadas à conexão devem ter as seguintes informações:

- se cada conexão está livre, pendente, ocupada ou abortada;

- se cada conexão está preparada para receber primitivas de indicação (requisição de serviço por parte do par remoto);

e- Auxiliar a programação do Árbitro e Filtro de Indicações (IFA) para filtrar as primitivas de indicação recebidas em cada associação nos critérios, que permanecerão iguais durante todo o tempo da associação, e nos critérios, que dependem de outras associações.

INVOCAÇÃO DE PROCESSO DE APLICAÇÃO:

O documento [ISO1494] da ISO apresenta o Modelo de Invocação de Processo de Aplicação (Figura 1.12).

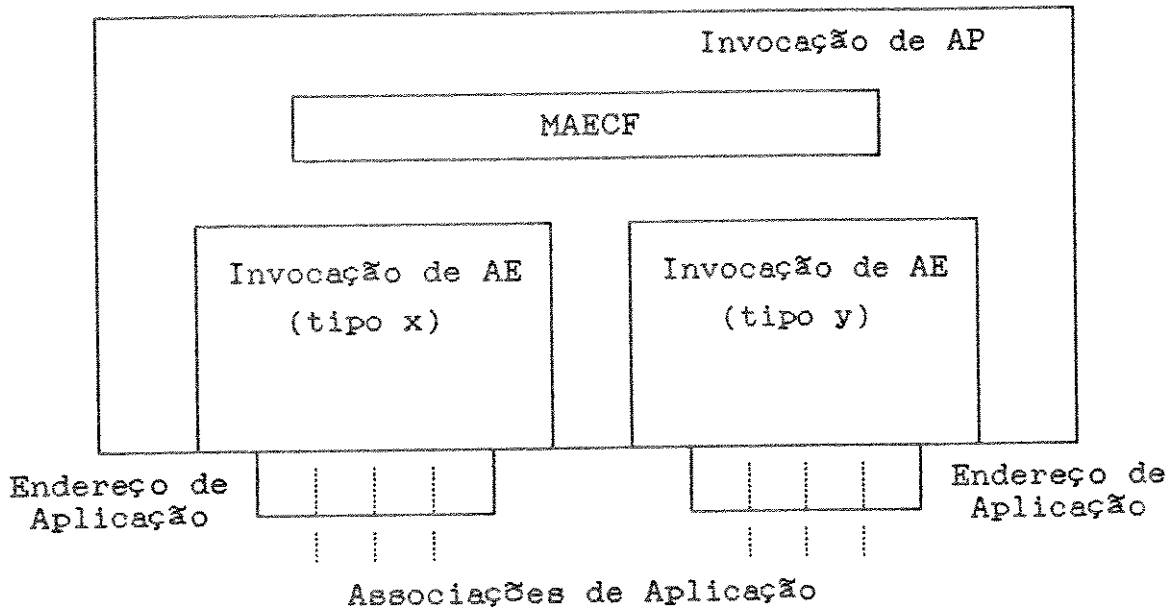


Figura 1.12 - Modelo da Camada de Aplicação

Neste modelo surge o conceito de:

d-) MAECF: Multiple Application Entity Controlling Function (Função de Controle de Entidades de Aplicação múltiplas).

O MAECF é responsável pelo controle das invocações múltiplas a uma mesma AE, e das invocações às diversas AEs, por uma mesma Invocação de Processo de Aplicação. O MAECF está associado com a capacidade de comunicação, e com a capacidade de processamento de informação. O MAECF está entre a camada de aplicação (capacidade de comunicação) e o AP (capacidade de processamento de informação). Portanto, as suas funções podem totalmente ser executadas pela API. As suas funções principais são:

a - Oferecer serviços a uma Invocação de Processo de Aplicação dos ASEs selecionados nas diversas associações num ambiente de múltiplas invocações às Entidades de Aplicação com múltiplas associações para cada Invocação de Entidade de Aplicação;

b - Controlar o número máximo de invocações por Entidade de Aplicação e o número máximo de tipos de Invocação à Entidade de Aplicação, para uma mesma Invocação de Processo de Aplicação. Estes limites são necessários, para que as diversas Invocações de Processos de Aplicação possam compartilhar os recursos disponíveis na Interface de Aplicação;

c - Auxiliar o controle do número máximo de Invocações por Entidade de Aplicação. Este limite define os recursos necessários para a Interface de Aplicação;

d - Auxiliar a construção de tabelas, para gerenciar os recursos das diversas Entidades de Aplicação. Por exemplo, deve-se ter a informação de quais invocações estão ativadas;

e - Auxiliar a construção de tabelas, para gerenciar os recursos das diversas Invocações de Entidades de Aplicação. Por exemplo, deve-se ter a informação de quais invocações estão à espera de pedidos de associação.

O MAECF tem dois tipos de funcionalidade: a de gerenciar as Invocações de Entidades de Aplicação (funções (a), (b) e (e)), e a de gerenciar as Entidades de Aplicação (funções (c) e (d)).

MÚLTIPLOS APs:

Quando se trabalha num ambiente de múltiplas invocações de um mesmo Processo de Aplicação e de múltiplos Processos de Aplicação, pode-se introduzir os conceitos de:

e-) MAPCF: Multiple Application Process Control Function (Função de Controle de Processos de Aplicação múltiplos);

f-) APCF: Application Process Control Function (Função de Controle dos Processos de Aplicação).

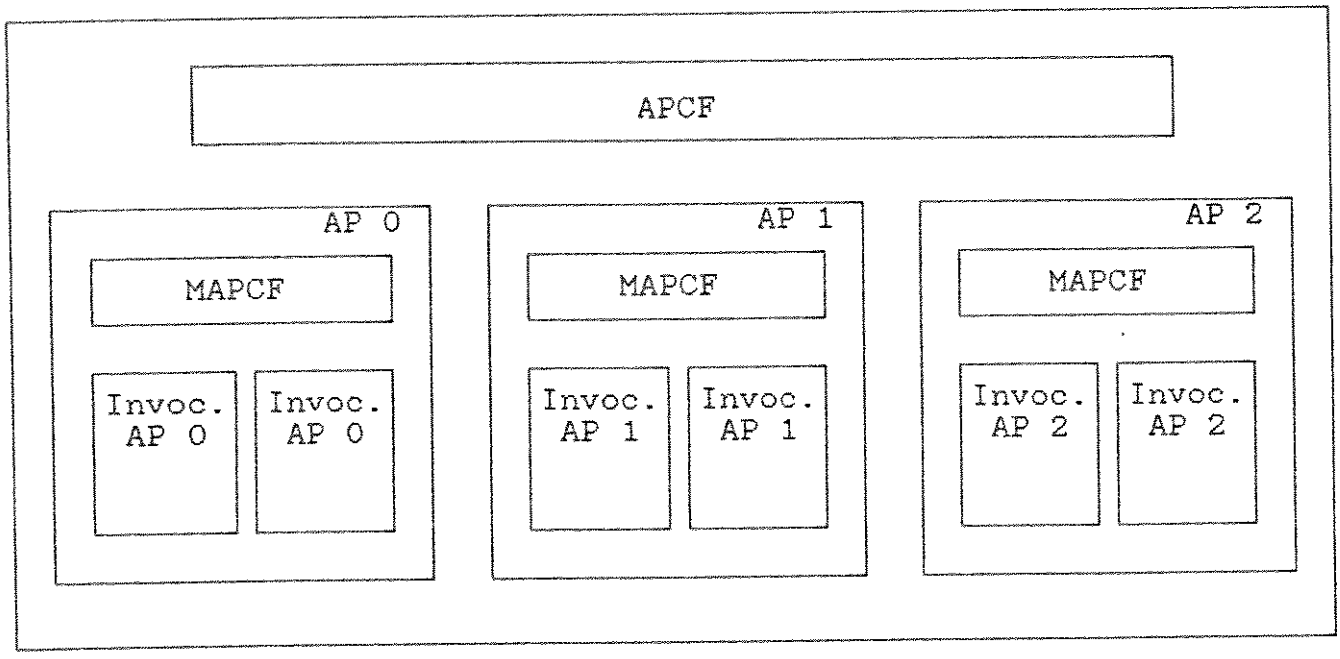


Figura 1.13 - Modelo para Múltiplos APs

O MAPCF e o APCF estão associados com a capacidade de processamento de informação num ambiente que suporta comunicação (Figura 1.13). O MAPCF e o APCF englobam a capacidade do Sistema Operacional executar um ambiente multitarefa, e ter múltiplas instâncias de um mesmo processo. O MAPCF e o APCF têm funções de sistema operacional e de sistema de comunicação. As funções relacionadas com comunicação podem totalmente ser executadas pela API.

As funções do MAPCF são:

a- Oferecer serviços a um Processo de Aplicação dos ASEs selecionados nas diversas associações, num ambiente de múltiplas associações por Invocação de Entidade de Aplicação com múltiplas invocações às Entidades de Aplicação, e múltiplas Invocações de Processos de Aplicação;

b - Controlar, se necessário, o número máximo de invocações de um mesmo Processo de Aplicação. A API não necessita desta restrição. Em função da simplicidade do sistema local, a retrição pode existir;

c - Gerenciar o envio / recebimento de mensagens (ou chamada

/ retorno de funções) entre a API e a invocação exata do Processo de Aplicação.

As funções principais do APCF são:

a - Oferecer serviços ao Sistema de Aplicação dos ASEs selecionados nas diversas associações, num ambiente de múltiplas associações por Invocação de Entidade de Aplicação com múltiplas Invocações às Entidades de Aplicação em múltiplas Invocações de Processos de Aplicação, e múltiplos Processos de Aplicação;

b - Controlar, se necessário, o número máximo de Processos de Aplicação no Sistema de Aplicação. A API também não necessita desta restrição, que pode existir em função da simplicidade do sistema local;

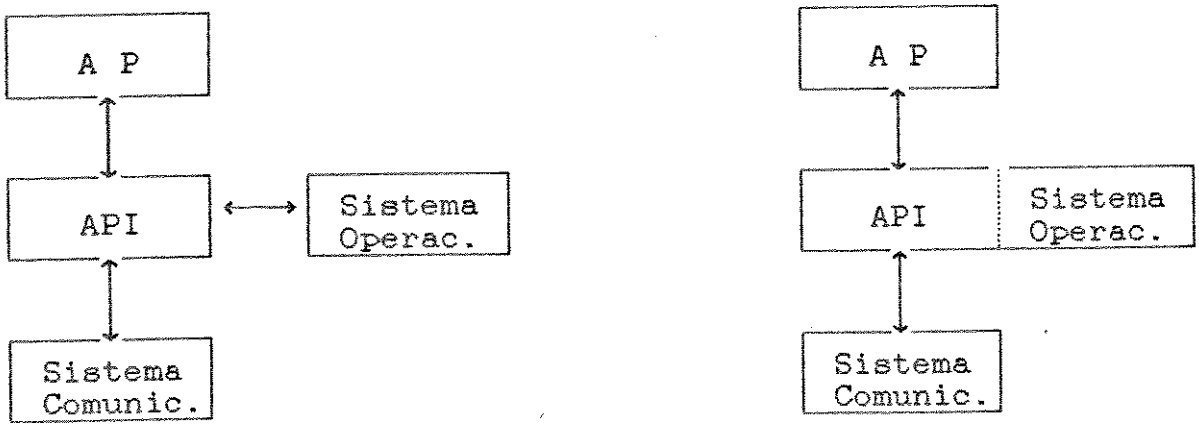
c - Gerenciar o envio / recebimento de mensagens (ou chamada / retorno de funções) entre a API e o Processo de Aplicação.

1.2.5 - API COMO SISTEMA OPERACIONAL

A API pode ser vista como uma entidade independente do sistema operacional, mas que o utiliza (Figura 1.14-a), ou como uma extensão do núcleo do sistema operacional (Figura 1.14-b). No segundo esquema, as funções da API podem ser vistas como funções do núcleo do sistema operacional. Este esquema torna-se interessante, por exemplo, nas funcionalidades do APCF e MAPCF, que são funcionalidades de sistema operacional e API, ao mesmo tempo.

As funcionalidades do SAO, SACF, MACF e MAECF não são tão próximas às de um sistema operacional, e portanto, não podem ser vistas como sua extensão, a menos que se veja o núcleo do sistema operacional, não apenas como um gerenciador de tarefas, memória e entrada/saída, mas também como um gerenciador de comunicação.

Contudo, as noções atuais de sistema operacional têm poucas similaridades com as de Interface de Aplicação. Portanto, prefere-se normalmente a API como uma entidade autônoma e independente do sistema operacional, embora o utilize, para realizar algumas funções (esquema da figura 1.14-a).



(a) - API utilizando o Sistema Operacional

(b) - API como extensão do Sistema Operacional

Figura 1.14 - Relação API / Sistema Operacional

1.2.6 - API COMO INTERFACE DE SERVIÇOS

Por outro lado, existe atualmente o conceito de "Service Shell" ou "Service Interface" (Interface de Serviços) ([TSC90], [WOL90] e [NEH90]), que define a visão externa dos serviços, num computador ou num sistema aberto de comunicação, incluindo as operações exportadas e importadas, e seus tipos correspondentes, além dos parâmetros de cada operação e seus tipos.

Definindo e implementando Interfaces de Serviços, pode-se construir um Ambiente de Serviços Abertos, num Sistema Distribuído. Neste contexto, a API é uma Interface de Serviços específica para comunicação, que pode ser acessada:

a - diretamente pelos Processos de Aplicação;

b - por outras Interfaces de Serviços que necessitam da capacidade de comunicação.

1.2.7 - SERVIÇOS RESPONDEDORES

Quando uma estação requisita um serviço de rede, fica claro,

utilizando as especificações da API, o que a mesma deve realizar. Contudo, quando uma estação recebe uma requisição de serviço (indicação), duas questões podem ser levantadas:

1 - A estação tem, ou não, autorização para responder ao serviço;

2 - Qual entidade (camada do sistema de comunicação ou processo aplicativo) responde ao serviço requisitado.

A resposta à segunda questão deve levar em consideração a complexidade da estação.

a - Se a estação é muito simples, como por exemplo, uma estação mini-MAP (possui apenas as camadas física, de enlace e de aplicação) [MAP87d], a própria camada de enlace pode responder a algumas requisições simples, como por exemplo, a requisição de leitura de uma variável. O protocolo de enlace LLC ("Logical Link Control") tipo 3 do Projeto MAP/TOP [ISO8802] permite esta flexibilidade;

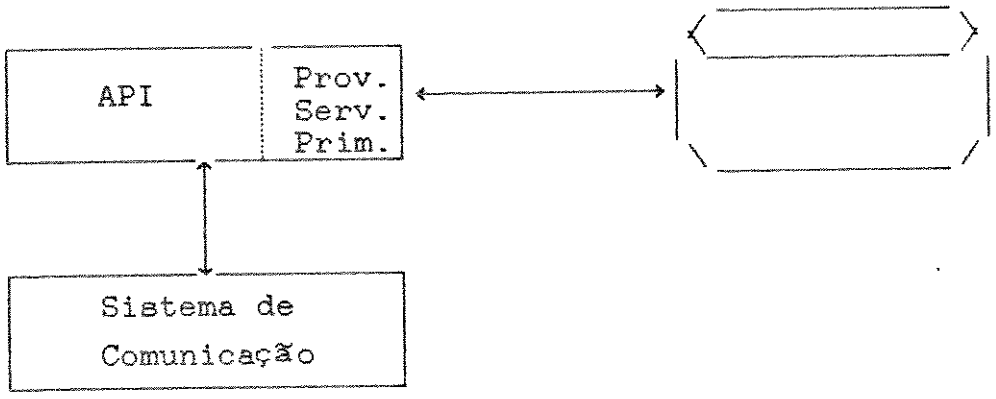
b - Se a estação é muito simples, mas a requisição não é, como por exemplo, se uma estação mini-MAP sem API é requisitada para a transferência de um programa, a camada de aplicação pode realizar o processamento necessário e responder;

c - Se a estação é simples, e a requisição não é simples, como por exemplo, se uma estação MAP com API, que implementa apenas algumas classes do MMS, é requisitada para a transferência de um programa, a API pode realizar o processamento necessário e responder (Figura 1.15-a);

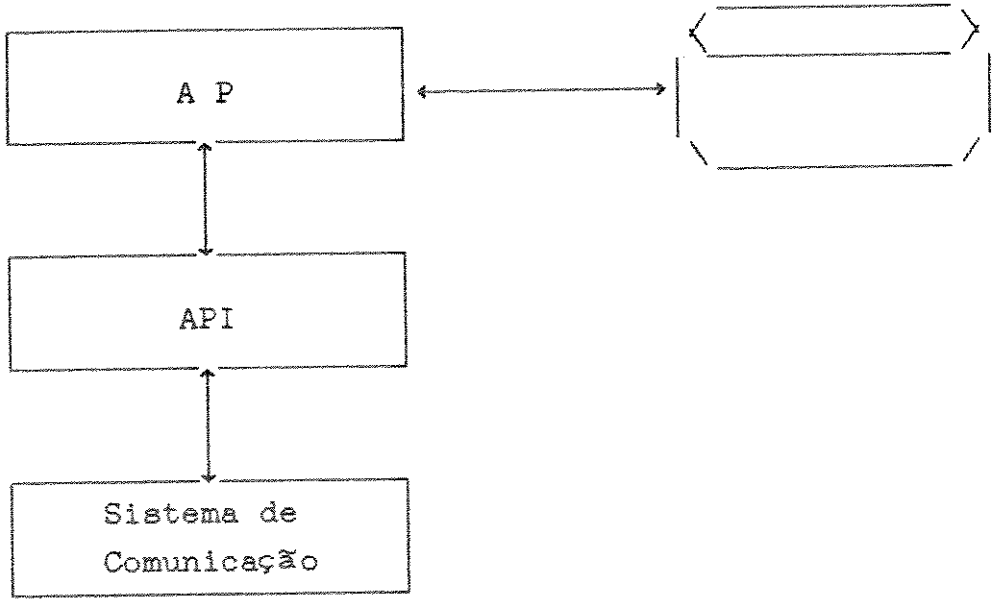
d - Se a estação é complexa, assim como a requisição, como por exemplo, se a estação servidora do FTAM é requisitada para efetuar a leitura de um arquivo, a resposta deve ser dada por um AP especializado da estação (Figura 1.15-b).

Quanto à primeira questão, os processos aplicativos (APs) da estação requisitada devem autorizar, ou não, o envio de respostas às requisições remotas.

a - Em estações muito simples, como por exemplo, em estações que possuem um só AP, este pode autorizar certas respostas, e não autorizar outras, independentemente das associações existentes, e das que venham a existir;



(a) - API



(b) - AP

Figura 1.15 - Resposta às requisições de serviços dada pelo(a)

b - Em estações muito simples, que necessitam de controle rígido de quem utiliza seus recursos, e em estações mais complexas, os APs podem programar as autorizações a nível de associação. Neste caso,

um AP ao solicitar uma associação, através da chamada de uma função da API, passa alguns parâmetros que indicam estas permissões. A API constroi uma tabela, indexada pelo número da associação, que contém as autorizações aos diversos serviços disponíveis (Figura 1.16);

Número Serviço / Núm.Associação	0	1	2	..	m
0	S	S	S		S
1	S	N	N		N
2	S	S	S		N
:					
n	S	N	S		N

S - Autorizado
N - Não Autorizado

Figura 1.16 - Exemplo de Tabela de Autorizações

c - Em estações complexas pode-se ter o procedimento anterior, mas com a diferença da autorização a um serviço específico numa associação específica poder mudar dinamicamente durante uma associação. Para tal, necessita-se de Funções Respondedoras da API que desempenhem esta funcionalidade.

1.3 - API DO PROJETO MAP/TOP: CONCEITOS E TERMINOLOGIA

1.3.1 - MODELO DE API

Nesta seção descreve-se o Modelo de API apresentado pelo Projeto MAP/TOP em [MAP88b]. A figura 1.17 mostra este Modelo, e como a maioria da informação transita através desta interface. Para não tornar a figura ilegível, não se ilustra todo o fluxo de informação.

Por exemplo, para grande parte das funções, os fluxos de informação requeridos para coordenação de eventos não estão ilustrados. As setas cheias indicam o fluxo de informação propriamente dito. As setas tracejadas mostram o fluxo de informação de controle.

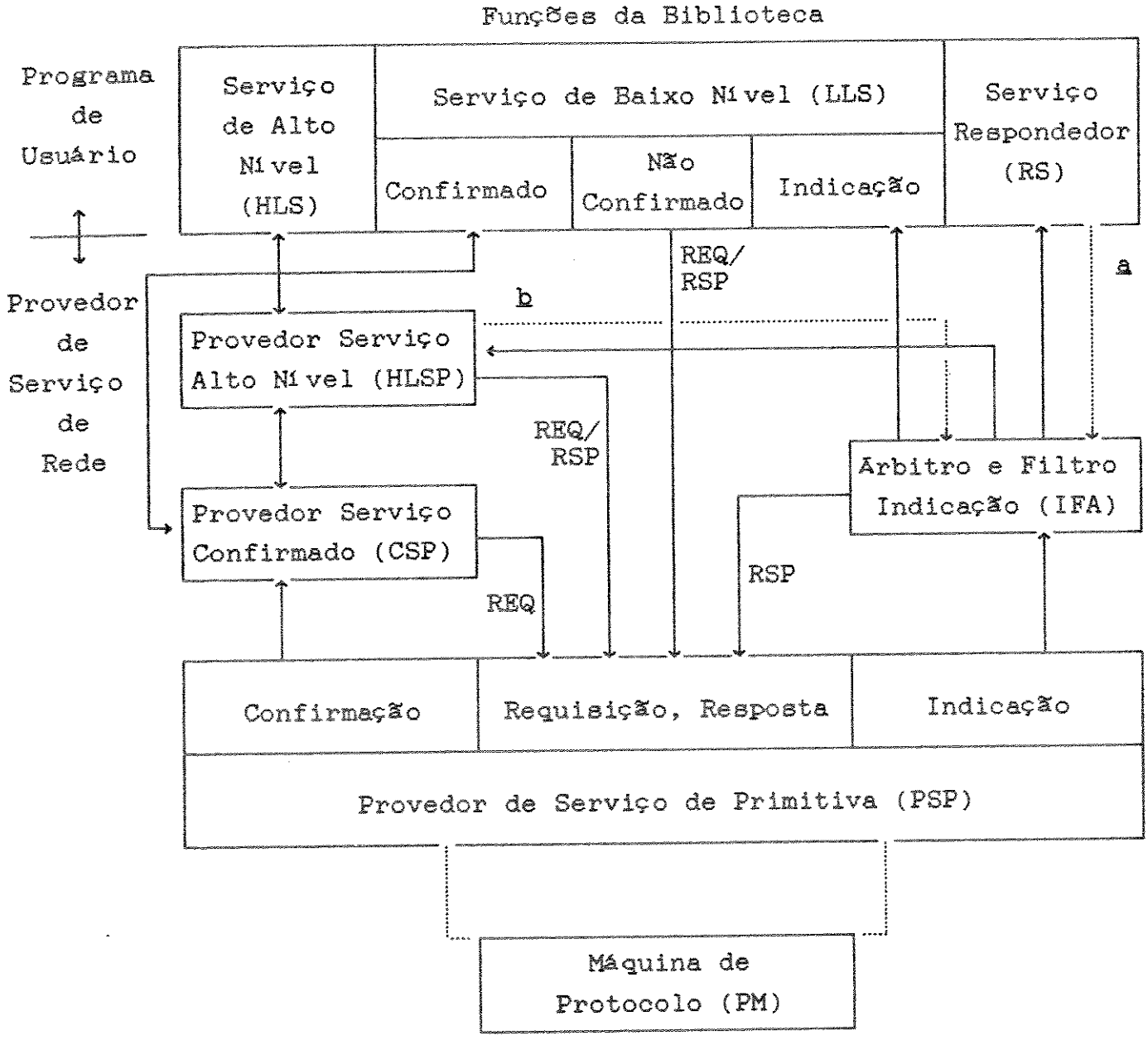


Figura 1.17 - Modelo de API do Projeto MAP/TOP

O Modelo da Interface de Aplicação é dividido em duas partes: a primeira referente ao conjunto de serviços que são do programa do usuário (corresponde à funcionalidade (a) descrita na seção 1.2.2), e a segunda englobando as funções do provedor de serviços da rede (corresponde às funcionalidades (b), (c1), (c2) e (c3) descritas na seção 1.2.2). Estas partes são chamadas Região do Programa do Usuário e Provedor de Serviços, respectivamente.

A Região do Programa do Usuário, que é na realidade uma biblioteca de funções, possui os seguintes serviços:

a - Serviços de alto nível: "Atende" as funções de alto nível;

b - Serviços de baixo nível: "Atende" as funções de baixo nível;

c - Serviços respondedores: "Atende" as funções respondedoras, ou seja, processa as primitivas de indicação recebidas de serviços não solicitados. A aceitação ou não destas primitivas de indicação depende dos critérios programados no IFA - Árbitro e Filtro de Indicações (Ver definição à frente nesta seção) pelo próprio serviço respondedor (seta a na figura 1.17). Através do serviço respondedor, o usuário da API pode delegar à mesma o uso de recursos seus, tal como o acesso a algumas variáveis de rede compartilhadas, ou delegar capacidades suas, tal como a autorização para realizar tarefas físicas num robô.

O provedor de serviços possui as seguintes partes:

a - Provedor de Serviços de Alto Nível - HLSP: Divide os serviços de alto nível em diversos serviços mais simples, e controla o processamento destes serviços. É específico da AE;

b - Provedor de Serviços Confirmados - CSP: Providencia o serviço de requisição e confirmação como um todo. É específico da conexão;

c - Árbitro e Filtro de Indicações - IFA: Examina as indicações solicitadas e as não solicitadas, que a camada de aplicação envia para a API. O IFA é programado pelas funções respondedoras (seta a na figura 1.17) para receber primitivas de indicação não

solicitadas, ou pelo HLSP (seta b) para receber primitivas de indicação solicitadas. O IFA é específico da conexão;

d - Provedor de Serviço de Primitiva - PSP: Providencia o envio de primitivas de requisição e resposta para a Máquina de Protocolo (PM), além de providenciar o recebimento de primitivas de confirmação e indicação da Máquina de Protocolo. O PSP é específico da conexão;

e - Máquina de Protocolo - PM: É descrita externamente à interface de aplicação, e corresponde ao protocolo de aplicação. É específica da conexão.

CARACTERÍSTICAS DAS FUNÇÕES DA API:

A API é modelada para linguagens de programação que permitem ao usuário definir tipos de dados e estruturas de dados complexas, além de associar estruturas de dados diferentes com o mesmo espaço na memória. Em outras palavras, a API é modelada para linguagens de programação que possuam os conceitos de funções e/ou procedimentos com algum tipo de passagem de parâmetros e o conceito de registro de tamanho variável. C, Pascal e Modula-2 são linguagens que têm os requisitos necessários.

Uma função da API é sensível ao contexto, se a mesma depende de funções chamadas anteriormente que estabeleceram um determinado contexto, ou influi em funções chamadas posteriormente por estabelecer um determinado contexto. Caso contrário, a função é livre de contexto.

Consideram-se nas funções duas classes de parâmetros: obrigatórios e opcionais. Os parâmetros obrigatórios são explicitamente especificados na chamada, enquanto que, em geral, os parâmetros opcionais são colocados num Bloco de Controle de Dados - DCB (Data Control Block). As vantagens dos parâmetros opcionais pertencerem a DCBs derivam de que:

a - Nem todos os usuários estão interessados em todos os parâmetros;

b - Para certas chamadas, os parâmetros são complexos;

c - Se o usuário quer utilizar os mesmos parâmetros

opcionais para diversas chamadas, constrói-se um único DCB, que é utilizado várias vêzes.

Os parâmetros das funções podem ser de entrada/saída ou de entrada. Os primeiros podem passar e retornar valores, enquanto que os últimos podem apenas passar valores. Numa declaração de função de API, os parâmetros de entrada obrigatórios são definidos primeiramente, seguido dos parâmetros de entrada opcionais, dos parâmetros de entrada/saída opcionais, e por último dos parâmetros de entrada/saída obrigatórios.

O nome das funções da API, utilizado nas especificações, pode conter diversas palavras, como por exemplo, a função GET_NAME_LIST associada ao protocolo MMS, que obtém a lista de nomes de objetos conhecidos a nível de rede no par remoto. O identificador do nome das funções, utilizado nas implementações, é composto por duas letras, que indicam a qual protocolo de aplicação ou tipo de gerenciamento esta função está associada, seguido do caractere "_", da primeira letra de todas as palavras menos a última do nome da função na especificação, e da última palavra do nome da função na especificação, desde que este identificador não seja igual ao de outra função.

Para o exemplo citado acima, tem-se o seguinte identificador: MM_GNLIST, onde MM representa o protocolo MMS. Outras representações são: FT para o FTAM, PC para comunicação privada e EM para gerenciamento de eventos. Se houver ambiguidade de identificadores, não apenas a primeira letra das palavras iniciais do nome da função na especificação deve ser utilizada.

1.3.2 - GERENCIAMENTO DE CONEXÃO

As funções da API necessárias para o gerenciamento das conexões são apresentadas em [MAP88c], e seguem as regras de especificação e convenção estabelecidas em [MAP88b], [ISO7498] e [ISO9545]. Portanto, o documento da ISO para a estrutura da camada de aplicação [ISO9545] é usado como uma das bases para a construção da API. Neste capítulo analisou-se a relação de funcionalidades da camada de aplicação definidas pela ISO e da API. Mostrou-se onde se enquadram os conceitos

de API nas definições da ISO, e quais os conceitos novos que a definição de API possui.

Dentro da camada de aplicação, a AE é o único objeto que pode ser endereçado num ambiente, que segue o Modelo de Referência RM OSI/ISO para Interconexão de Sistemas Abertos [ISO7498]. As funções de gerenciamento de conexão da API possuem alguns parâmetros comuns. A AE é conhecida localmente pelo parâmetro MY_AE_NAME, e remotamente pelo parâmetro RESPONDING_AE_NAME. Uma AE pode ter várias invocações, ou seja, pode estar sendo utilizada (invocada) por vários APs, e/ou sendo invocada mais de uma vez pelo mesmo AP, ao mesmo tempo. O parâmetro AE_LABEL indica a qual invocação o AP está associado.

Uma invocação pode ter várias associações (conexões). Por exemplo, o AP pode estar se comunicando com dois ou mais APs remotos em computadores remotos diferentes. O parâmetro CONNECTION_ID identifica a conexão. Cada conexão está relacionada com uma determinada invocação.

Numa conexão, o usuário pode solicitar vários serviços. Por exemplo, para o protocolo MMS, cada solicitação de serviço tem um parâmetro INVOKE_ID diferente (Figura 1.18).

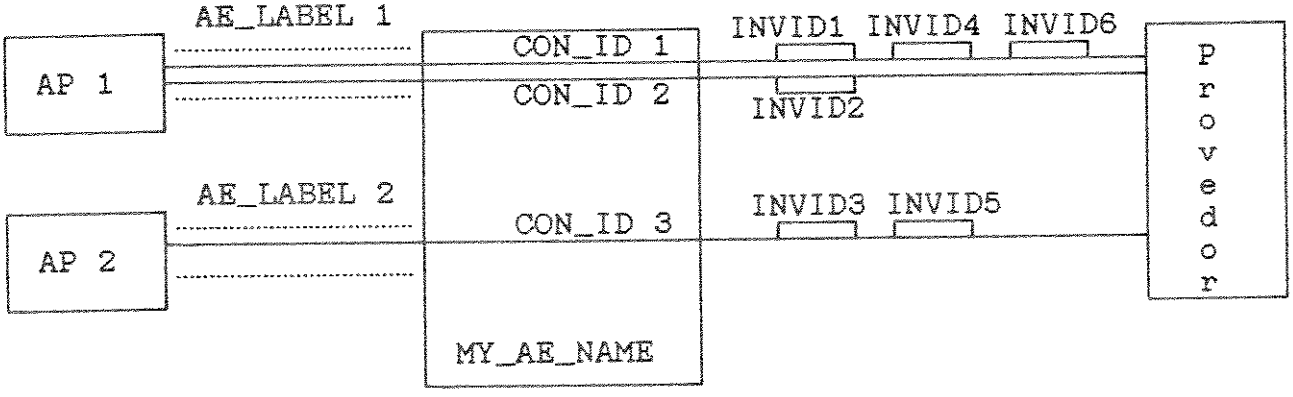


Figura 1.18 - Relação Invocação/Conexão

A camada de apresentação do Projeto MAP é orientada à conexão. Na camada de aplicação estabelecem-se associações entre AEs locais e remotas. Uma associação de aplicação utiliza uma conexão de

apresentação, e define um CEP - Ponto Final de Conexão - ("Connection End Point") no P-SAP - Ponto de Acesso do Serviço de Apresentação - ("Presentation Service Access Point") usado pela AE.

Para a API, a interação entre dois APs deverá traduzir-se numa associação a nível da camada de aplicação. Em [MAP88c] o termo conexão utilizado a nível de API é sinônimo de associação, e é usado para referenciar a associação de aplicação correspondente.

As funções da interface de gerenciamento de contexto são sensíveis ao contexto. As funções especificadas em [MAP88c] são apresentadas na Tabela 1-I. A seguir comentam-se estas funções:

a - Application Entity Activation (AEACTIVATION): Estabelece uma invocação da AE, por parte do AP chamador, com o provedor de serviço local. Se realizada com sucesso, um AE_LABEL será retornado, e poderá ser utilizado pelas funções de estabelecimento de conexão. Esta função não implica no envio ou recebimento de primitivas para/da camada de aplicação. No caso específico da interface definida em [MAP88c], cada invocação da AE pode utilizar um único ASE, além do ACSE;

b - Application Entity Deactivation (AEDEACTIVATION): Termina a invocação da AE identificada pelo parâmetro AE_LABEL. Esta função não implica no envio ou recebimento de primitivas para/da camada de aplicação. Se realizada com sucesso, este AE_LABEL associado com a invocação da AE não será mais válido;

c - Connect (CONNECT): Estabelece uma conexão com uma AE remota, usando uma invocação de AE já existente. O AE_LABEL gerado pela função (a) é requerido. Causa o envio de uma requisição A_ASSOCIATE (primitiva do ACSE). Se realizada com sucesso, um identificador de conexão CONNECTION_ID é retornado;

d - Listen (LISTEN): Declara o desejo do AP através de uma invocação de AE de aceitar uma indicação de associação. Causa a espera de uma indicação A_ASSOCIATE. Se realizada com sucesso, esta função provê um CONNECTION_ID. Esta função da API deve garantir os recursos necessários para uma associação, quando chamada;

e - Stop Listen (SLISTEN): Informa à API, que o usuário (AP) de uma particular invocação de AE não está mais interessado em receber indicações de associação. Esta função não implica no envio ou

recebimento de primitivas para/da camada de aplicação. Se existirem chamadas de funções LISTEN pendentes pertencentes a esta invocação de AE, estas funções serão retornadas com código de retorno, indicando a chamada da função SLISTEN posteriormente a elas;

f - Answer (ANSWER): Informa à API, se o AP aceita ou rejeita uma requisição de associação do par remoto. Causa o envio de uma resposta A_ASSOCIATE. Esta função tem como parâmetro de entrada o CONNECTION_ID retornado da função LISTEN;

g - Release Request (RREQUEST): Termina uma associação estabelecida através da função CONNECT ou da função ANSWER. Causa o envio de uma requisição A_RELEASE (primitiva do ACSE). Se realizada com sucesso, o CONNECTION_ID correspondente não será mais válido;

h - Release Response (RRESPONSE): Informa à API, que o AP aceita a requisição de término de associação do par remoto. Causa o envio de uma resposta positiva A_RELEASE. Para [MAP88c] não existe resposta negativa. Contudo, o AP não precisa responder imediatamente à requisição de término de associação, podendo inclusive requisitar serviços remotos. Se realizada com sucesso, o CONNECTION_ID correspondente não será mais válido;

i - Abort (ABORT): Informa à API o término abrupto de uma associação. Causa o envio de uma requisição A_ABORT (primitiva do ACSE). Se realizada com sucesso, o CONNECTION_ID correspondente não será mais válido. O processamento desta função tem prioridade sobre outras atividades da API;

j - Indication Receive (IRECEIVE): Declara o desejo do usuário de uma associação (AP) receber primitivas de indicação. Esta função recebe qualquer primitiva de indicação aceita pelo IFA, e a entrega ao AP solicitante. As indicações podem ser do serviço de gerenciamento de conexão, tais como indicações de A_ABORT, P_ABORT e A_RELEASE, ou do serviço dos ASEs específicos;

k - Application Entity Reset (AERESET): Realiza uma "reinicialização" ("Reset") numa invocação da AE. Esta função tem como parâmetro de entrada o AE_LABEL correspondente. Esta função aborta todas as conexões abertas e pendentes desta invocação, causando o envio de uma requisição de A_ABORT para cada uma das conexões. Além

disto, realiza um SLISTEN;

xx_aeactivation	Application Entity Activation
xx_aedeactivation	Application Entity Deactivation
xx_connect	Connect
xx_listen	Listen
xx_slisten	Stop Listen
xx_answer	Answer
xx_rrequest	Release Request
xx_rrsp	Release Response
xx_abort	Abort
xx_ireceive	Indication Receive
xx_aereset	Application Entity Reset

Tabela 1-I - Funções de Gerenciamento de Conexão

O AP, para utilizar os serviços de uma AE, deve inicialmente ativar a AE através da chamada da função AEACTIVATION. Esta função da API retornará um AE_LABEL (identificador de invocação),que poderá ser usado posteriormente nos pedidos de estabelecimento de associação.

Quando o AP desejar estabelecer uma associação, o AP chamará a função CONNECT. Quando a associação for estabelecida, a API retornará ao AP um CONNECTION_ID (identificador de conexão). Para esperar por uma requisição de associação do AP remoto, o AP local chama a função LISTEN. Quando a indicação de associação chegar à API local, esta retorna ao AP o CONNECTION_ID. O AP local aceita ou não a requisição de associação, chamando a função ANSWER.

O CONNECTION_ID será usado como parâmetro de entrada em todos os serviços associados aos ASEs solicitados pelo AP. O AP, para terminar a associação chama a função RREQUEST, fornecendo o CONNECTION_ID correspondente. O AP, para responder a uma requisição de término de associação por parte do AP remoto, chama a função RRESPONSE. Para desativar uma AE, o AP chama a função AEDEACTIVATION.

1.3.3 - FUNÇÕES DE SUPORTE

A especificação [MAP88d] define as funções de suporte da API (Tabela 1-II). Pode-se dividir estas funções em dois grupos, que tratam do gerenciamento de eventos e dos demais gerenciamentos.

O gerenciamento de eventos providencia serviços que definem eventos, e esperam e notificam a ocorrência de eventos. O objetivo é implementar chamadas assíncronas das funções da API. O conceito de semáforo definido por Dijkstra [DIJ65] é generalizado para permitir que um processo espere por uma disjunção de eventos múltiplos. As funções WAIT e NOTE são definidas para a espera e a notificação de eventos, e são utilizadas pelos APs e pela API, respectivamente. Contudo, estas funções, que hoje são definidas externamente ao sistema operacional, podem migrar para funções do próprio sistema operacional. O único requisito exigido do sistema operacional para implementar o gerenciamento de eventos é que este providencie um ambiente multitarefa, onde tarefas podem ser ativadas e desativadas.

O AP pode chamar assincronamente diversas funções da API, como por exemplo, diversas funções READs e WRITEs associadas com o protocolo MMS. Cada chamada é associada com um evento. O nome do evento é o parâmetro RETURN_EVENT_NAME de cada chamada. Se o parâmetro for NULL significa que a chamada é síncrona. Os conceitos de chamadas síncronas e assíncronas foram definidos na seção 1.2.2.

Depois das diversas chamadas assíncronas, o AP pode chamar a função WAIT para esperar pelo resultado de qualquer uma destas chamadas anteriores. Quando a API tiver o resultado de uma chamada de READ ou WRITE, a API notificará o AP, chamando a função NOTE. O parâmetro de entrada da função NOTE é o RETURN_EVENT_NAME da função resolvida. É necessário tantas chamadas de WAIT por parte do AP, quantas chamadas assíncronas o AP realizar.

A função WAIT tem como parâmetro de entrada o tempo de espera. Três valores são válidos para este parâmetro:

- o valor FOREVER, que é utilizado para se esperar indefinidamente por um evento;

- um valor positivo, que é utilizado para se esperar até que um evento ocorra, ou que o tempo definido por este valor expire;

- o valor 0, que é utilizado para se esperar por zero unidades de tempo. Para este valor, a função WAIT retorna um RETURN_EVENT_NAME, e consequentemente, um resultado de chamada, apenas se este evento já ocorreu, não esperando pela ocorrência de eventos.

A função WAIT tem como parâmetro de saída o RETURN_EVENT_NAME do primeiro evento que ocorreu, relacionado com o AP que chamou a função WAIT. Este RETURN_EVENT_NAME é o da chamada de função assíncrona executada pelo AP, associada com o primeiro serviço assíncrono pendente que terminou (a API recebeu as primitivas de confirmação correspondentes, e tratou-as).

Os demais gerenciamentos englobam a alocação dinâmica de DCBs, a tradução de códigos de erro retornados para "strings" (cadeia de caracteres) explicativas, e a notificação que recursos não disponíveis anteriormente já estão disponíveis. Estes serviços são oferecidos pelas funções da API: DYNAMIC_INITIALIZE_DCB (DIDCB), DYNAMIC_FREE_DCB (DFDCB), GET_PRINTABLE_ERROR (GPERROR) e NOTE_WHEN_CLEARED (NWCLEARED), respectivamente.

a. DIDCB: Esta função aloca memória e "inicializa" (inicia) um DCB para um serviço particular, preenchendo este DCB com valores "defaults". Posteriormente, o AP pode preencher os outros valores, e chamar a função da API que utiliza o DCB. Se a API desejar chamar diversas vêzes funções, que utilizam o mesmo DCB com valores idênticos, o AP necessitará chamar apenas uma vez a função de suporte. [MAP88d] permite que o AP passe como parâmetro efetivo de uma função da API, que tem como parâmetro formal um DCB, um apontador nulo. Isto significa implicitamente que a função da API deve chamar a função DIDCB, para alocar memória e "inicializar" o DCB;

b. DFDCB: Esta função libera um DCB previamente alocado pela função DIDCB;

c. GPERROR: Esta função tem como parâmetro de entrada um código de erro retornado por alguma função da API, e como parâmetro de saída um "string" que explica o erro ocorrido;

d. NWCLEARED: Esta função notifica, que recursos não disponíveis

para o estabelecimento de determinada conexão já estão disponíveis. O AP pode utilizá-la após ter chamado uma função de requisição de estabelecimento de conexão, e ter obtido como resposta o código de retorno associado com NO_CONNECTION_RESOURCE. Esta função tem como parâmetros de entrada o CONNECTION_ID associado à conexão requisitada e o RETURN_EVENT_NAME pelo qual se esperará pela ocorrência do evento, que representa a disponibilidade de recursos para o estabelecimento da conexão.

em_wait	Wait
xx_didcb	Dynamic Initialize DCB
xx_dfdcb	Dynamic Free DCB
xx_gperror	Get Printable Error
xx_nwcleared	Note When Cleared

Tabela 1-II - Funções de Suporte

1.4 - MODELOS DE INTERFACES DE APLICAÇÃO

1.4.1 - PROJETO MAP/TOP

Na seção 1.3.1 definiu-se as três partes (unidades funcionais) da API: Biblioteca (LIB), Procedimentos de Controle e Provedor de Serviços de Primitivas (PSP). Os Procedimentos de Controle descritos foram três: Provedor de Serviços de Alto Nível (HLSP), Provedor de Serviços Confirmados (CSP) e Arbitro e Filtro de Indicações (IFA).

O Modelo simplificado de API apresentado pelo Projeto MAP/TOP [MAP88b] é o da figura 1.19, e é geral para qualquer protocolo de aplicação, embora o Projeto MAP/TOP defina a API apenas para os protocolos MMS [MAP88e], FTAM [MAP88a] e Comunicação Privada [MAP88f], como foi analisado na seção 1.2.2.

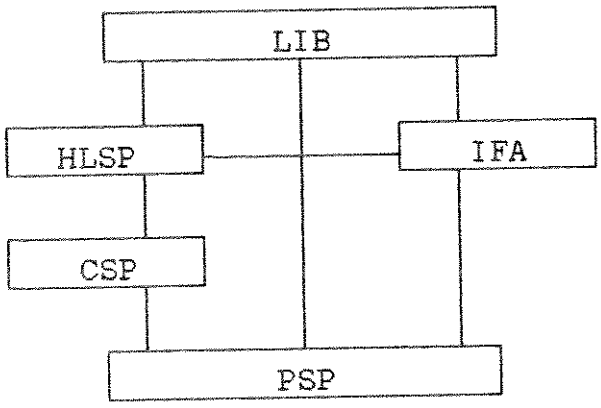


Figura 1.19 - Modelo de API do Projeto MAP/TOP

Contudo, novas funções (unidades funcionais) podem ser adicionadas à API para:

a- aumentar as funcionalidades da API para os protocolos de aplicação para os quais a API foi definida;

b- atender os requisitos de API necessários para outros protocolos de aplicação.

Esta seção 1.4 pretende propor e analisar novas unidades funcionais que poderiam ser adicionadas à API, e a partir desta discussão, propor um Modelo Geral de API. Esta discussão é apresentada resumidamente em [MAD90a] e [MAD90c]. [MAD90b] discute o Modelo Geral.

1.4.2 - PROTOCOLO MMS

A API do Projeto MAP/TOP para o Protocolo MMS é comentada com detalhes na seção 3.2.1. Nesta seção, apenas os aspectos relacionados com a modelagem desta Interface são analisados.

O Protocolo de Aplicação MMS ([ISO9506a] e [ISO9506b)] caracteriza-se por um modelo "Cliente/Servidor" por inúmeras requisições complexas de serviços feitas pelo cliente a estações servidoras simples (CNCs, CLPs, entre outras). Este fato pode levar a um esquema de resposta dos serviços solicitados (requisições que chegam como indicações) por parte da própria API. Portanto, deve-se adicionar ao Modelo da API uma nova unidade funcional para o

Processamento de Informações (IP - Indication Processing) (Figura 1.20). Esta unidade tem as seguintes funções:

a- Processar os serviços solicitados pelas primitivas de indicação aceitas;

b- Acessar a memória principal e o diretório local de arquivos quando necessário (utilizando a unidade funcional VDRDB, como será descrito em seguida);

c- Solicitar o envio de primitivas que respondem às indicações aceitas:

- ao PSP, se a resposta é de um serviço confirmado, ou a resposta será enviada através de uma requisição de um serviço não confirmado;

- ao CSP, se a resposta será enviada através de uma requisição de um serviço confirmado.

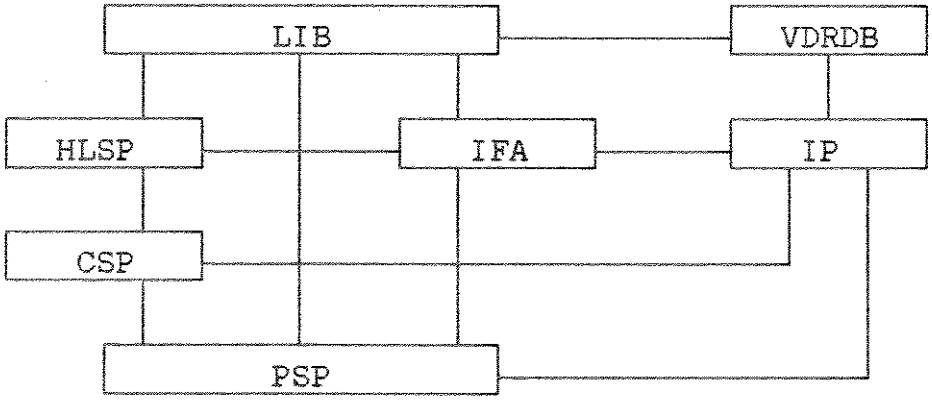


Figura 1.20 - Modelo de API para o MMS

Como conseqüência da introdução desta nova unidade funcional (IP), o AP pode diminuir o número de solicitações de INDICATION RECEIVE na proporção dos serviços de indicação que serão respondidos diretamente pela API (através do IP).

Contudo, o AP ainda precisará chamar a função INDICATION RECEIVE, para receber indicações que necessariamente serão tratadas no AP, como por exemplo, indicações de A_ABORT, P_ABORT e RELEASE_REQUEST (CONCLUDE_REQUEST no caso do protocolo MMS).

O IFA deve ser programado para direcionar as indicações para o IP. Esta programação pode ser feita em duas ocasiões:

a- No estabelecimento da associação através de parâmetros específicos das funções normais de gerenciamento de contexto da API;

b- Depois do estabelecimento da associação através de parâmetros específicos de funções respondedoras próprias para esta finalidade.

Os serviços MMS são oferecidos sobre Dispositivos Virtuais da Manufatura (VMDs) que devem corresponder a Dispositivos Reais. Torna-se necessária uma função de Mapeamento Dispositivo Virtual / Dispositivo Real e vice-versa. Introduz-se no Modelo da API a unidade funcional VDRDB (Virtual Device / Real Device Binding). É utilizando esta unidade funcional, que o IP pode realizar as funções de acesso à memória principal e Diretório de Arquivos.

O anexo A de [MAP88e] apresenta as regras de conversão entre a representação local dos dados e a representação MMS dos dados, como por exemplo, em relação ao tamanho dos tipos inteiros (com sinal e sem sinal), à representação dos tipos real, vetor ("array") e cadeia de caracteres ("strings"), entre outros. O VDRDB pode realizar estas conversões, e nestes casos, o VDRDB seria requisitado por algumas funções auxiliares da API: New MMS Type, New Data, Interpret Data Representation, Interpret MMS Type, V_PUT e V_GET.

Em particular, a função auxiliar V_GET permite ao usuário transformar informação local (endereço e representação locais) em dado na representação MMS, que pode ser enviado pela rede. A função auxiliar V_PUT converte o dado MMS recebido da rede para a representação local, e o coloca num endereço local. Estas funções utilizam a unidade funcional VDRDB para realizar as conversões entre a representação local e a representação MMS, e para acessar a memória local. De fato, as funcionalidades de V_GET e V_PUT estão muito próximas das funcionalidades do VDRDB para o caso particular de acesso às variáveis. Contudo, as funcionalidades da unidade VDRDB são mais gerais, englobando outros tipos de conversões, como por exemplo, a conversão entre Sistemas de Arquivos Virtuais e Reais.

Como foi visto na seção 1.3.1, o AP pode delegar à API o uso de seus recursos ou suas capacidades, como por exemplo o acesso às

variáveis compartilhadas de rede e a autorização para realizar tarefas físicas num robô. Quando uma indicação de um serviço delegado chegar à API, o IP pode realizar o processamento respectivo, e o VDRDB executar o mapeamento para acesso aos Dispositivos Reais (memória ou robô).

A API para o MMS na estação cliente é diferente da API na estação servidora, em relação a quais serviços de requisição e resposta o AP pode chamar. [MAP88e] especifica a API para a estação cliente, que pode solicitar todos os serviços MMS, e ser somente solicitada para alguns serviços MMS, tais como serviços de Gerenciamento de Contexto, Acesso às Variáveis e Gerenciamento de Semáforos. [MAP88e] também define que os serviços, que podem se respondidos automaticamente pela API, são os relacionados com Acesso às Variáveis e Gerenciamento de Semáforos.

As estações servidoras do MMS normalmente não têm API por serem simples. Contudo, se a estação servidora possuir uma API simples, o IP pode realizar o processamento das indicações que a estação servidora recebe, e o VDRDB realizar o mapeamento para o acesso aos Dispositivos Reais da Manufatura. Utilizando este esquema, a API simplifica o(s) AP(s) da estação servidora, pois o(s) AP(s) não necessita(m) responder a todas as requisições.

É conveniente que a estação cliente do MMS tenha API. Nesta estação, o IP pode realizar o processamento de indicações relacionadas com Gerenciamento de Semáforos e Acesso às Variáveis, e o VDRDB o mapeamento para o acesso aos Dispositivos Reais, simplificando a execução dos APs.

O Modelo da API para o protocolo MMS é apresentado na figura 1.20.

1.4.3 - PROTOCOLO FTAM

A API do Projeto MAP/TOP para o Protocolo FTAM é comentada em detalhes no Apêndice C. Nesta seção, apenas os aspectos relacionados com a modelagem desta Interface são analisados.

O Protocolo de Aplicação FTAM oferece serviços de gerenciamento, acesso e transferência de arquivos sobre uma estrutura de Sistema de

Arquivos Virtual [ISO8571]. A idéia é similar à do protocolo MMS. O protocolo MMS é utilizado na automação industrial, onde os Dispositivos Reais da Manufatura diferem entre si, mesmo para dispositivos de mesmo tipo (CLPs, CNCs, entre outros). Então, a nível de protocolo, a comunicação é toda feita sobre Dispositivos Virtuais. Desta maneira, os dispositivos são iguais para todas as estações, e a "linguagem" de comunicação torna-se única. Em cada estação, só é necessário fazer o mapeamento do Dispositivo Virtual, que todas as estações conhecem, para o Dispositivo Real, que só é conhecido pela estação local.

Para o protocolo FTAM, as estações podem possuir Sistemas de Arquivos diferentes, mas todas as estações se comunicarão utilizando uma "linguagem" comum, que é o Sistema de Arquivos Virtual definido no FTAM. Em cada estação, só é necessário fazer o mapeamento do Sistema de Arquivos Virtual, que todas as estações conhecem, para o Sistema de Arquivos Real, que só é conhecido pela estação local.

O FTAM oferece três modos de manipulação de arquivos: transferência, acesso e gerenciamento. A transferência de arquivo engloba o movimento de um arquivo completo entre dois Sistemas de Arquivos diferentes utilizando o ambiente OSI/ISO, enquanto que o acesso a um arquivo engloba a manipulação de parte de um arquivo (por exemplo, registros) num Sistema de Arquivos por parte de um usuário num sistema remoto. O gerenciamento de arquivos envolve a leitura ou alteração remota de atributos de arquivos.

Para o FTAM existem os conceitos de estação servidora e de estações clientes. A estação servidora é uma estação complexa, e isto geralmente leva a um esquema de resposta por parte de um AP especializado.

Entretanto, se a estação servidora é simples e possuir API, o IP pode responder às indicações, utilizando o VDRDB para realizar o mapeamento entre o Sistema de Arquivos Virtual e o Sistema de Arquivos Real. Por exemplo, a API para o protocolo FTAM permite que um AP no nó A solicite a cópia de um arquivo residente no Sistema de Arquivos do nó B para o Sistema de Arquivos do nó C. Neste caso, o nó B ou o nó C pode ser uma estação servidora simples. Contudo, nos dois casos

(estação servidora complexa e simples), a existência da API pode ajudar no mapeamento entre o Sistema de Arquivos Virtual e o Sistema de Arquivos Real, mesmo que a API não responda às indicações. A API pode fornecer aos APs uma interface mais próxima do Sistema de Arquivos Real da estação, devido à existência da unidade funcional VDRDB.

A API do protocolo FTAM para uma estação cliente [MAP88a] não recebe primitivas de indicação para os serviços específicos do protocolo e para os serviços de gerenciamento de contexto, com exceção das indicações de ABORT. Contudo, pode-se expandir o conceito de API para estação cliente, para o caso da API receber algumas primitivas de indicação. No exemplo citado anteriormente em que um AP em A solicitava a cópia de um arquivo de B em C, a estação B poderia ser uma estação servidora e a C uma estação cliente. Então, a estação C necessitaria receber primitivas de indicação para a escrita do arquivo em C, apesar de ser uma estação cliente. Neste caso, se a API de C possuir as unidades funcionais IP e VDRDB, a API pode independentemente de AP (a menos de sua autorização para tal), realizar os processamentos necessários e responder às requisições.

As estações clientes podem ser simples, a nível de comunicação via FTAM, e podem ter um esquema de acesso ao Sistema de Arquivos Real por parte da própria API. Neste caso, a unidade funcional VDRDB realiza os mapeamentos necessários.

A figura 1.21 apresenta o Modelo de API para o FTAM, adicionando à mesma apenas o bloco funcional VDRDB (a API realiza os mapeamentos Sistema de Arquivo Virtual / Real, mas não processa as indicações). Se adicionar os blocos funcionais IP e VDRDB, o Modelo de API para o FTAM é equivalente ao da API para o MMS (Figura 1.20).

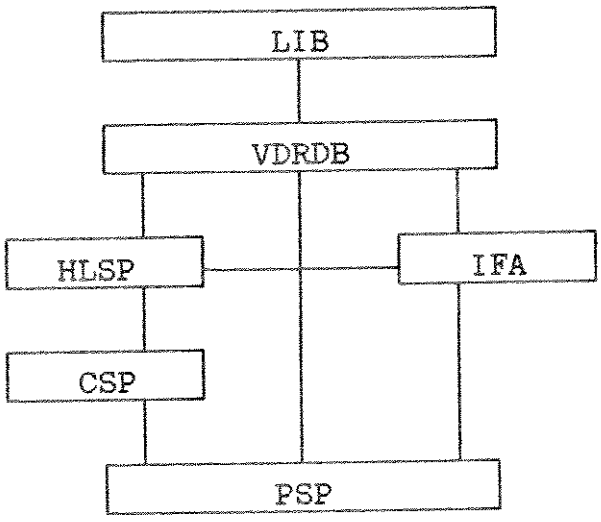


Figura 1.21 - Modelo de API para o FTAM (adição apenas de VDRDB)

1.4.4 - PROJETO TOP

O Projeto TOP [MAP87b] tem a estrutura de aplicação apresentada na figura 1.22.

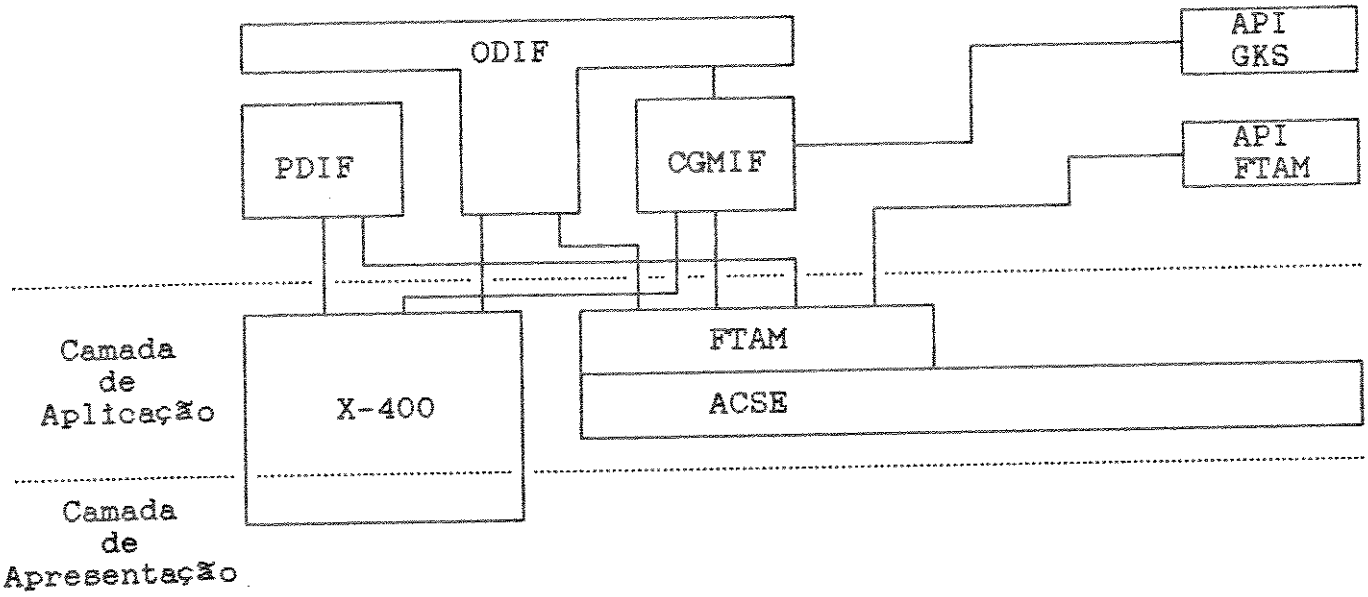


Figura 1.22 - Estrutura de Aplicação do Projeto TOP

O Projeto TOP oferece três tipos de formatos de dados de arquivos e dois tipos de interfaces de aplicação. Os formatos de dados de arquivos são:

a- CGMIF - "Computer Graphics Metafile Interchange Format", para troca de arquivos gráficos;

b- PDIF - "Product Data Interchange Format", para troca de dados de definição de produtos;

c- ODIF - "Office Document Interchange Format", para troca de documentos de escritório com texto, gráficos e "raster".

O Projeto TOP especifica o IGES - "Initial Graphics Exchange Specification" - V3.0 como o PDIF. IGES tem, por esta razão, um lugar importante na integração das operações da manufatura. A figura 1.23 mostra a necessidade de um Formatador e Tradutor entre o AP e os arquivos no formato IGES.

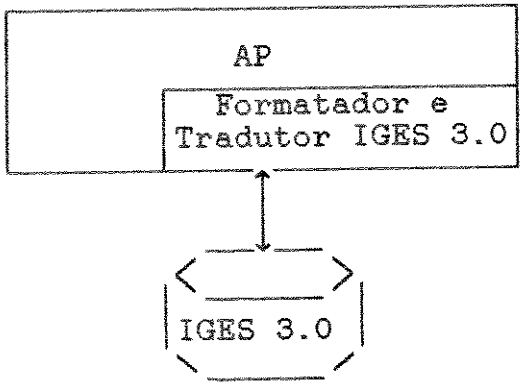


Figura 1.23 - Relação AP / IGES 3.0

O Projeto TOP especifica o uso do CGMIF, PDIF e ODIF sobre os serviços FTAM (também sobre o X-400). Os formatos CGMIF e PDIF são úteis para operações de CAD/CAM, onde partes de produtos devem ser desenhados e analisados. O formato ODIF é utilizado para tarefas de processamento de dados de escritório [MCG88].

Como visto anteriormente, para a manipulação de arquivos nos formatos TOP, precisa-se de Formatadores e Tradutores, ou seja, necessita-se de Interfaces que tenham as seguintes características:

a- ofereçam uma biblioteca de funções para manipular estes formatos (corresponde à LIB da API);

b- executem o mapeamento de arquivos com formatos convencionais para os formatos definidos no TOP e vice-versa (corresponde ao VDRDB da API).

Portanto, tem-se o modelo da figura 1.24 como o Modelo de API para os formatos de dados do TOP.

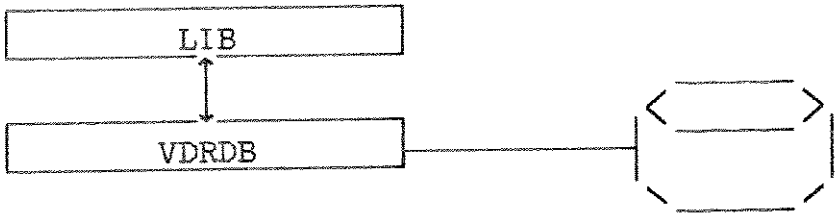


Figura 1.24 - Modelo de API para Formatos de Dados do TOP

As Interfaces de Aplicação do Projeto TOP são:

a- API para o FTAM;

b- API para o GKS - "Graphical Kernel System", que é utilizado para aplicações gráficas.

Na seção anterior mostrou-se o Modelo de API para o protocolo FTAM. O Modelo de API para o GKS, como tem que manipular os arquivos gráficos, e gerenciar o controle e o acesso destes arquivos utilizando um protocolo de aplicação (como por exemplo o FTAM), deve conter as características do Modelo para os Formatos de Dados do TOP (Figura 1.24), e do Modelo do Protocolo de Aplicação (no exemplo, o FTAM - Figura 1.21). A figura 1.25 apresenta o Modelo de API para o GKS.

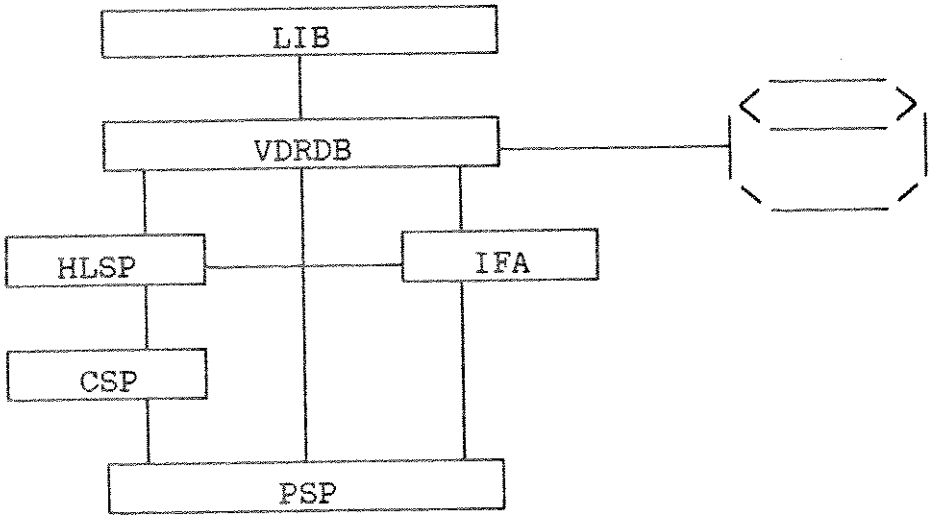


Figura 1.25 - Modelo de API para o GKS

1.4.5 - MODELO GERAL

Desta forma é possível definir um Modelo de API, que suporte apenas um protocolo de aplicação (mas genérico), e englobe todas as particularidades dos Modelos descritos anteriormente: Modelos para o Projeto MAP/TOP, para os protocolos MMS e FTAM, para os Formatos de Dados do TOP e para o GKS (Figura 1.26). Ao Modelo do Projeto MAP/TOP foram introduzidas duas unidades funcionais novas: IP e VDRDB.

As funcionalidades do IP foram descritas anteriormente, e as do VDRDB são:

a- Realizar Mapeamento Dispositivo Virtual / Dispositivo Real e vice-versa. O Mapeamento Arquivo Virtual / Arquivo Real é um caso particular;

b- Criar e manipular Formatos de Dados padrões para arquivos gráficos, de definição de produtos e de documentos de escritório.

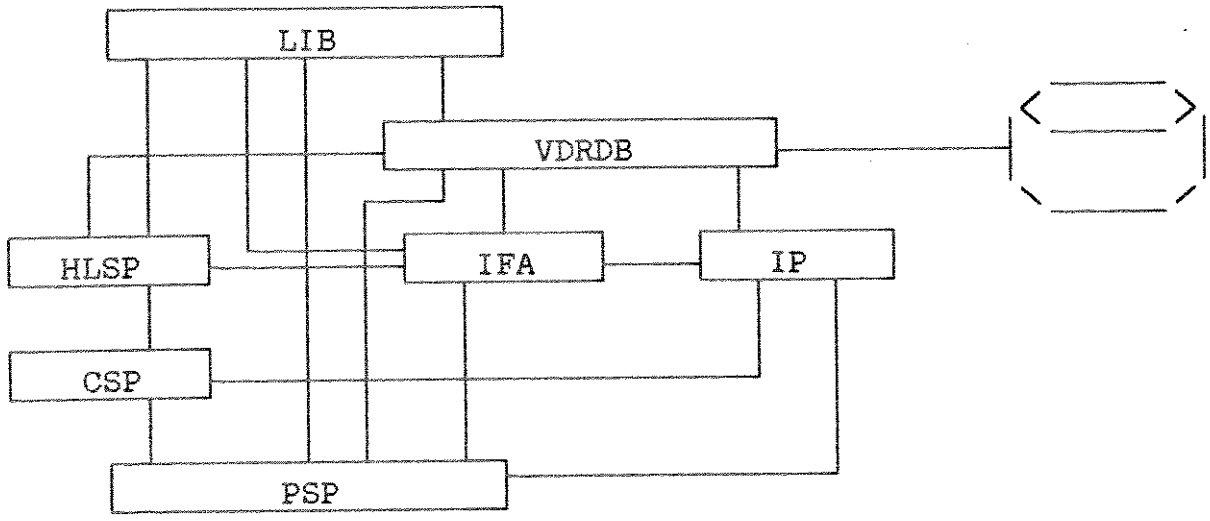


Figura 1.26 - Modelo de API para Suportar um Protocolo de Aplicação Genérico

ENTIDADES DE APLICAÇÃO COM DIVERSOS ASEs:

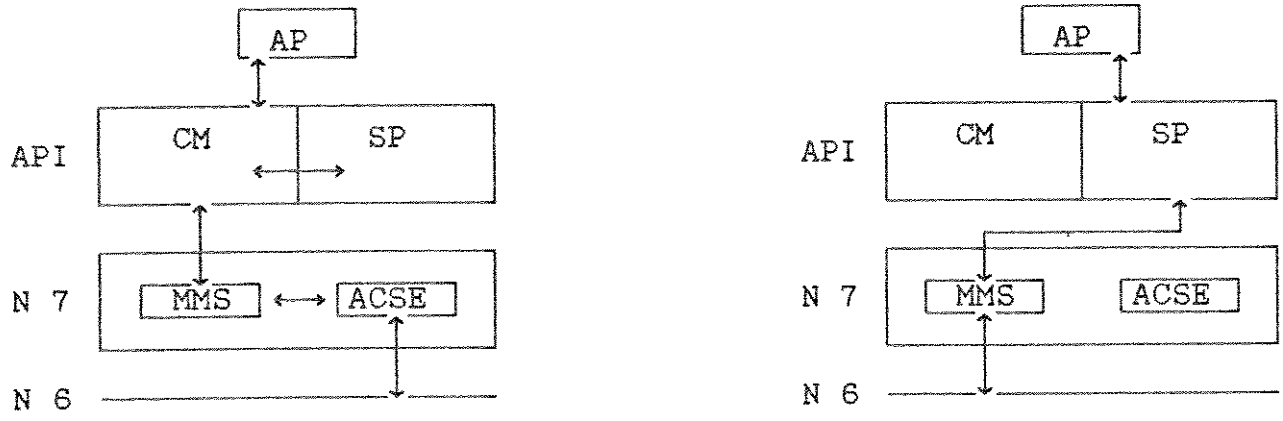
Contudo, as Entidades de Aplicação possuem normalmente mais de um ASE, ou seja, mais de um protocolo de aplicação.

Suponha-se que uma Entidade de Aplicação tenha dois ASEs, por exemplo o MMS e o ACSE ([ISO8649] e [ISO8650]). Para obter uma implementação conceitualmente correta, deve-se considerar que a API possua logicamente duas partes: uma relacionada com o Gerenciamento de Contexto (devido ao ACSE) e outra específica do SASE (devido ao MMS), chamadas CM (Context Management) e SP (Specific), respectivamente.

Na fase de estabelecimento da associação, o processo aplicativo fornece informação relativa ao ACSE e ao SASE para a CM-API.

A nível de API, o CM e o SP interagem entre si, gerando uma primitiva para a camada de aplicação. Esta primitiva contém informação para o ACSE gerar um A_ASSOCIATE, que possua no seu campo de informação um INITIATE gerado pelo SASE (no caso, pelo MMS). Desta forma, a primitiva contém, além de informação relativa ao ACSE para que o mesmo estabeleça a associação, a informação necessária para o

SASE iniciar a comunicação (Figura 1.27-a).



(a) - Durante estabelecimento e término da associação

(b) - Durante associação estabelecida

Figura 1.27 - Fluxo de Informação

Depois da associação estabelecida, o processo aplicativo chama funções da SP-API para realizar serviços específicos do SASE (Figura 1.27-b). Nos dois casos citados, o tráfego de informação no sentido contrário é similar.

Como visto anteriormente, o CM e o SP interagem entre si com a finalidade de gerar uma invocação de primitiva na camada de aplicação. Cada uma das duas partes lógicas da API (CM e SP) possui os blocos funcionais LIB, HLSP, CSP, IFA, IP e VDRDB próprios.

Duas funções de Controle de Objeto da Associação de Transmissão e Recepção (TAOCF - Transmission Association Object Control Function e RAOCF - Reception Association Object Control Function, respectivamente) monitoram a interação entre os dois componentes lógicos da API.

Desta forma, pode-se propor um Modelo Geral de API que suporte Entidades de Aplicação com mais de um ASE (Figura 1.28).

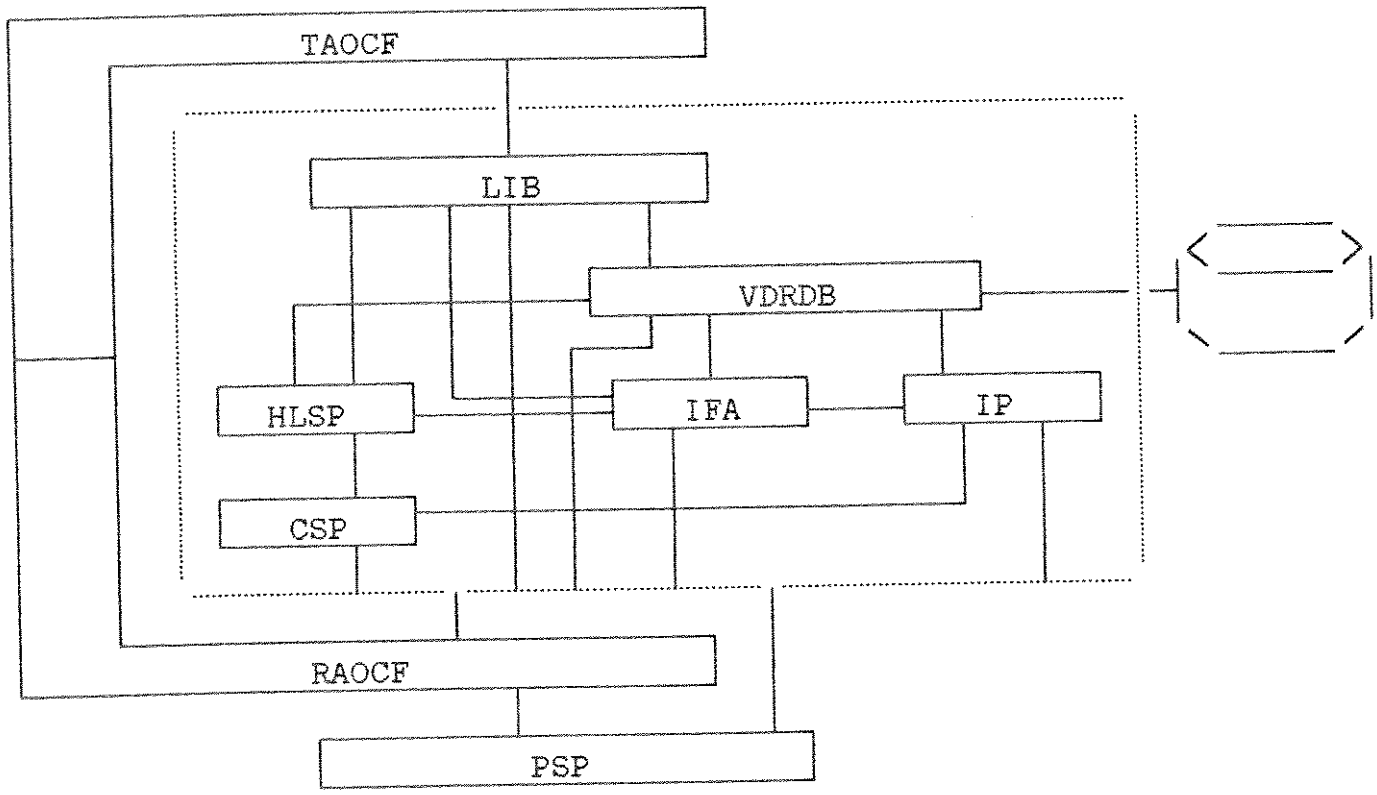


Figura 1.28 - Modelo Geral de API

O TAOCF é requisitado para realizar o processamento relativo ao controle da interação entre os componentes lógicos da API, quando um AP chama uma função da API que necessite este controle. O TAOCF controla as chamadas interrelacionadas da LIB, HLSP, CSP, IFA, IP e VDRDB dos diversos componentes lógicos da API, por parte destes componentes.

EXEMPLO:

A seguir será dado um exemplo para o caso de uma AE composta dos protocolos de aplicação MMS e ACSE.

Um AP, para requisitar o estabelecimento de uma associação, deve chamar a função CONNECT da API (através do TAOCF). A figura 1.29 apresenta o controle das interações entre os componentes lógicos da API (SP-API e CM-API), realizado pelo bloco funcional TAOCF, para este

caso de requisição de estabelecimento de associação por parte do AP local. Se a operação é realizada com sucesso, o fluxo de informação na API é o seguinte:

a- O TAOCF chama a LIB do CM, que retorna;

Neste passo é verificado os parâmetros da função CONNECT relativos à informação ACSE.

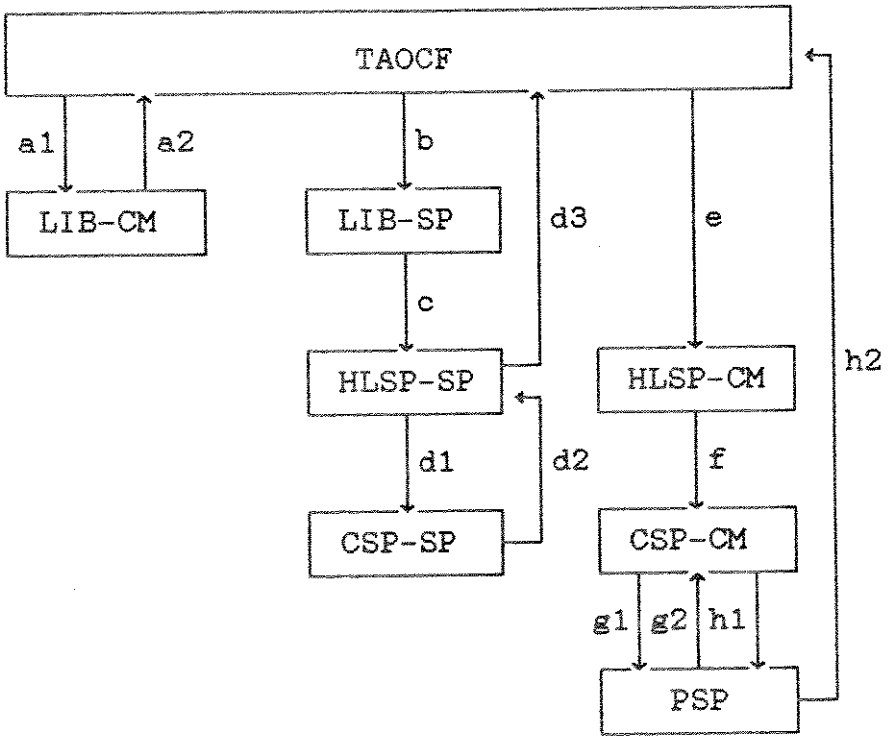


Figura 1.29 - Controle das Interações entre os Componentes Lógicos da API realizado pelo TAOCF

b- O TAOCF chama a LIB do SP;

Neste passo é verificado os parâmetros da função CONNECT relativos à informação MMS.

c- A LIB do SP chama o HLSP do SP;

d- O HLSP do SP chama o CSP do SP, e retorna ao TAOCF;

Em (c) e (d) é construída a primitiva de requisição INITIATE associada ao MMS.

e- O TAOCF chama o HLSP do CM;

f- O HLSP do CM chama o CSP do CM;

Em (e) e (f) é construída a primitiva de requisição A_ASSOCIATE associada ao ACSE.

g- O CSP do CM chama o PSP, e fica esperando pela resposta de se é possível enviar a requisição pretendida;

h- O CSP do CM chama o PSP, para enviar a requisição A_ASSOCIATE, e retorna ao TAOCF;

Neste ponto, a API fica esperando pela confirmação do serviço.

O RAOCF é requisitado para realizar o processamento relativo ao controle da interação entre os componentes lógicos da API, quando o PSP passa ao RAOCF uma primitiva de indicação ou confirmação que necessite este controle. O RAOCF controla as chamadas interrelacionadas da LIB, HLSP, CSP, IFA, IP e VDRDB dos diversos componentes lógicos da API, por parte destes componentes.

Um exemplo de necessidade deste controle corresponde ao caso, em que uma primitiva A_ASSOCIATE.ind ou A_ASSOCIATE.cnf seja entregue ao RAOCF pelo PSP da API local. No exemplo, que se está analisando, tem-se a entrega da primitiva A_ASSOCIATE.cnf.

A figura 1.30 apresenta o controle das interações entre os componentes lógicos da API (SP-API e CM-API), realizado pelo bloco funcional RAOCF, para o caso em questão.

Ao receber a primitiva de confirmação A_ASSOCIATE.cnf, o fluxo de informação na API é o seguinte:

i- O PSP passa a primitiva para o RAOCF;

j- O RAOCF chama o HLSP do SP;

k- O HLSP do SP chama o CSP do SP, e retorna ao RAOCF;

Em (j) e (k) é interpretada a primitiva de confirmação INITIATE associada ao MMS.

l- O RAOCF chama o CSP do CM;

m- O CSP do CM chama o HLSP do CM;

Em (l) e (m) é interpretada a primitiva de confirmação A_ASSOCIATE associada ao ACSE.

n- O HLSP do CM retorna ao RAOCF.

Finalmente, o RAOCF devolve ao AP solicitante a resposta do pedido de estabelecimento de associação.

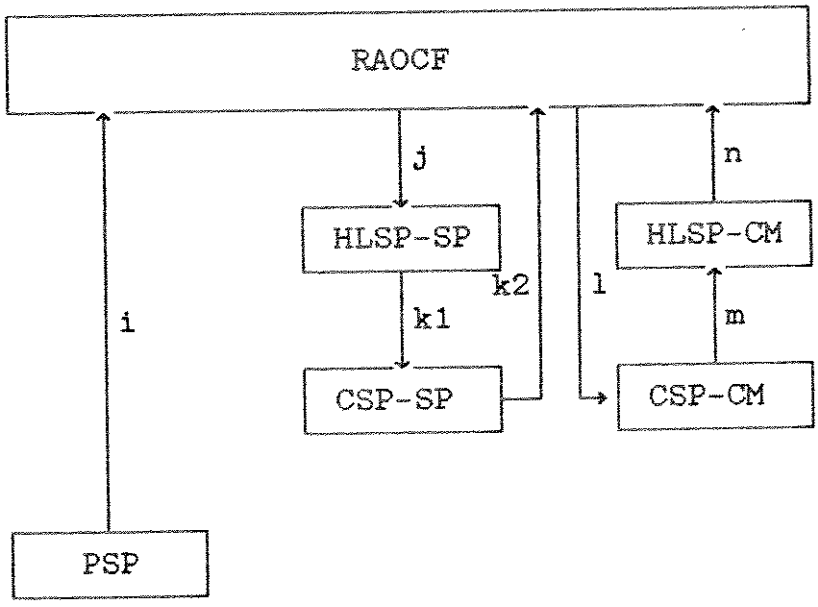


Figura 1.30 - Controle das Interações entre os Componentes Lógicos da API realizado pelo RAOCF

Tanto o TAOCF como o RAOCF interagem com os APs, embora só o TAOCF seja invocado diretamente pelo usuário. O RAOCF é invocado apenas pelo PSP. Contudo, TAOCF e RAOCF devolvem para os APs os resultados de operações solicitadas à API, em função de onde estes resultados ficam disponíveis.

O TAOCF devolve resultado aos APs para chamadas de funções da API:

a- que apresentam erro, e por isso não podem ser realizadas;

b- que não implicam no envio de primitivas para a camada de aplicação;

c- de maneira assíncrona.

O RAOCF devolve resultado aos APs para chamadas de funções de maneira:

a- síncrona;

b- assíncrona, em que a chamada implica no envio e/ou recebimento de primitivas para/da camada de aplicação.

A seção 3.2.2.4 apresenta os algoritmos do TAOCF e RAOCF para o

SISDI_MAP.

As funcionalidades do TAOCF e RAOCF são subconjuntos das funcionalidades do SAO e SACF apresentados anteriormente na seção 1.2.4. No exemplo analisado anteriormente, o TAOCF e o RAOCF têm as funcionalidades a, b e c.

SASEs E CASEs:

As Entidades de Aplicação podem ter diversos ASEs, o que implica na existência de diversas CM-APIs e SP-APIs.

Alguns Exemplos de SASEs (ASEs específicos da aplicação) são: DS - "Directory Service" [ISO9594], MMS - "Manufacturing Message Specification" ([ISO9506a] e [ISO9506b]), FTAM -"File Transfer Access and Management" [ISO8571], ASEs na área do MHS - "Message Handling System" ([ISO10021] e [CCITT400]), TP - "Transaction Processing" [ISO10026] e RDA -"Remote Database Access" [ISO9579].

Exemplos de CASEs (ASEs independentes da aplicação) são: ACSE - "Association Control Service Element" ([ISO8649] e [ISO8650]), CCR - "Commitment Concurrency and Recovery" ([ISO9804] e [ISO9805]), ROSE - "Remote Operation Service Element" [ISO9072] e RTSE - "Reliable Transfer Service Element" [ISO9066].

A Tabela 1-III mostra para os diversos SASEs, quais CASEs são utilizados. A última linha da tabela apresenta o caso do RDA e TP, sendo utilizados conjuntamente. Quando a tabela mostra que um CASE é utilizado por um SASE com um (X), significa que o SASE pode ou não utilizar o CASE em função do contexto de aplicação desejado.

No caso de uma Entidade de Aplicação ter diversos ASEs, TAOCF e RAOCF controlam, a nível de API, a interação das diversas CM-APIs e SP-APIs que necessitam interagir. Dois exemplos serão analisados com maiores detalhes nas próximas seções: o TP e o RDA.

SASE \ CASE	ACSE	CCR	RTSE	ROSE
DS	X			X
MMS	X			
FTAM	X	(X)		
MHS	X		(X)	X
RDA	X			X
TP	X	X		
RDA e TP	X	X		X

Tabela 1-III

Como já foi visto na seção 1.2.4, o TP requer controle de múltiplas associações interrelacionadas. Foi dado o exemplo de que deve-se garantir que uma requisição de "commit" só seja enviada se todos os nós participantes em todas as associações da árvore de transações tenham votado "READY" para a requisição prévia apropriada.

Algumas destas tarefas de controle podem ser delegadas para a API, se esta existir no ambiente de comunicação. Desta maneira, o TAOCF e o RAOCF têm um subconjunto das funcionalidades do MACF, apresentadas na seção 1.2.4.

PAR TAOCF-RAOCF:

É conveniente ressaltar que para a API são necessárias as funções do par TAOCF - RAOCF além das funções do HLSP, apesar do fato de que o HLSP possa invocar mais de uma primitiva da camada de aplicação para um único serviço solicitado.

A necessidade deriva do fato de que, num instante específico, o HLSP manipula invocações relacionadas com um único protocolo de aplicação, enquanto que o par TAOCF - RAOCF manipula invocações relacionadas com diversos protocolos de aplicação.

O par TAOCF - RAOCF controla a interação dos diversos componentes lógicos da API para construir / interpretar primitivas a serem enviadas / recebidas para / da camada de aplicação relacionadas com diversos protocolos de aplicação. Este relacionamento pode ser

realizado de duas maneiras:

a- Uma única primitiva de um protocolo de aplicação, em que campos estão relacionados com outros protocolos de aplicação (Figura 1.31);

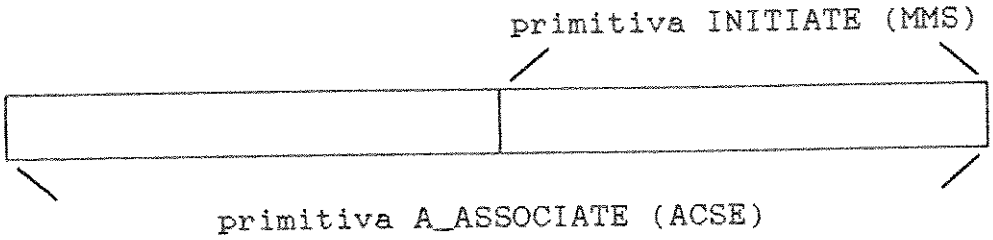


Figura 1.31 - Uma Primitiva / Vários Protocolos de Aplicação

b- Diversas primitivas de diferentes protocolos de aplicação.

1.4.6 - MODELO GERAL DE API PARA O PROCESSAMENTO DE TRANSAÇÕES

Pode-se utilizar o Modelo Geral proposto na figura 1.28 para construir a API para o protocolo de processamento de transações distribuídas TP [ISO10026].

Uma transação é uma unidade de trabalho atômica. Isto implica que as operações da unidade de trabalho sejam totalmente realizadas com sucesso ou, nenhuma delas.

O TP se aplica às atividades de processamento de informação que envolvem interação de trabalho entre sistemas abertos separados, e utiliza o conceito de transação. Suporta esta interação oferecendo um serviço que implementa a comunicação e a coordenação necessárias para a realização de um protocolo de "commit" em duas fases.

Além disso, suporta aplicações distribuídas hierárquicas. Por exemplo, uma aplicação no nó A pode acessar duas Bases de Dados nos nós B e C. Se B não dispõe do dado necessário, pode acessar outras Bases de Dados em D e E. Toda esta estrutura de comunicação em forma de árvore é considerada uma única aplicação distribuída.

Para tal, o TP faz uso do Protocolo CCR, que possui funções

básicas de gerenciamento de transações. Estas funções são também baseadas em árvores de transações. [BEV89] apresenta um exemplo de interação entre o TP e o CCR. Portanto, nestes sistemas as AEs possuem diversos ASEs:

a- Um ou mais U-ASEs - protocolos de aplicação de serviços específicos que o Usuário utiliza;

b- ACSE - responsável pelo estabelecimento e controle das associações com os sistemas remotos;

c- TP-ASE - responsável pelo gerenciamento do diálogo (coordenação e comunicação das transações distribuídas);

d- ROSE - quando necessário, responsável pela geração de A_PDUs pertencentes aos ASEs do Usuário (U-ASEs);

e- CCR - responsável pela geração de A_PDUs para efetivação ("commit") de duas fases.

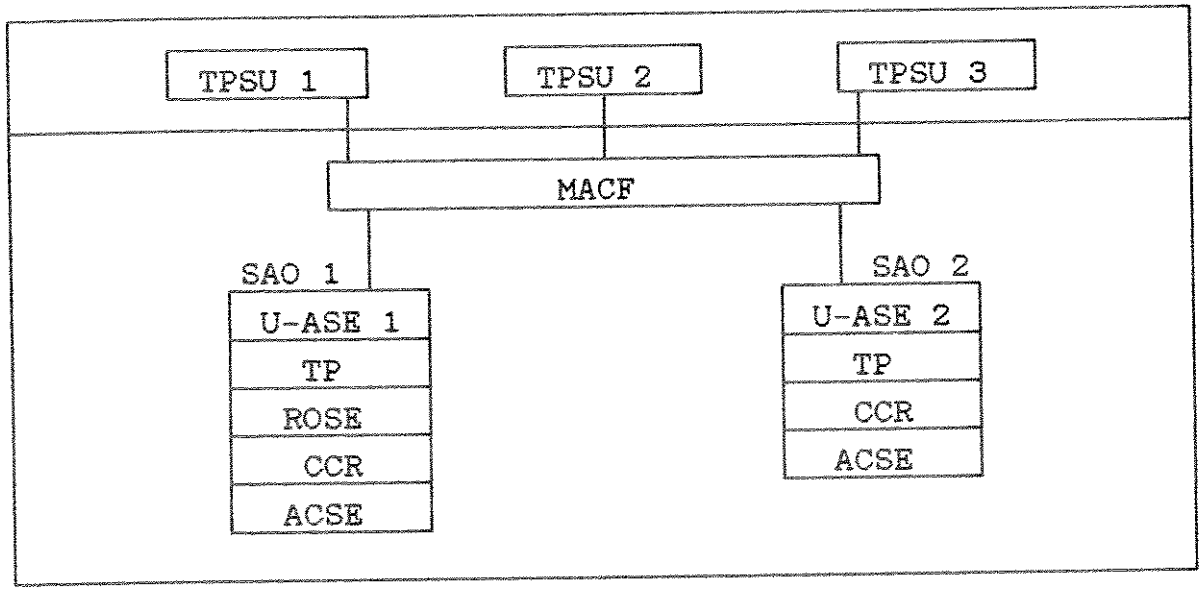


Figura 1.32 - Modelo de Serviço do TP

Neste caso, o TAOCF e o RAOCF controlam as interações entre os componentes lógicos da API, que necessitam esta interação: API-UASEs, API-ACSE, API-TPASE, API-ROSE e API-CCR (funcionalidades SACF e MACF).

A estas funcionalidades da API deve ser adicionada a de controlar Múltiplos Usuários de Serviços TP (TPSUs), como mostra a figura 1.32,

que apresenta o Modelo de Serviço do Protocolo TP.

1.4.7 - MODELO GERAL DE API PARA O RDA

O Modelo Geral de API proposto na figura 1.28 também pode ser utilizado para o RDA -"Remote Database Access" [ISO9579]. A padronização RDA facilita o acesso a uma Base de Dados remota (servidor), por um processo de aplicação residente numa estação de trabalho inteligente ou em outro sistema de Base de Dados (cliente). As duas estações comunicantes em RDA têm funcionalidades diferentes: no cliente existe um AP, que pergunta e manipula dados no servidor.

A figura 1.33 mostra o Modelo de Funcionalidades do RDA num ambiente de sistemas abertos, que é apresentado em [PAP86] e [JOH90]. [JOH90] também mostra o Modelo de Serviços do RDA (Figura 1.34).

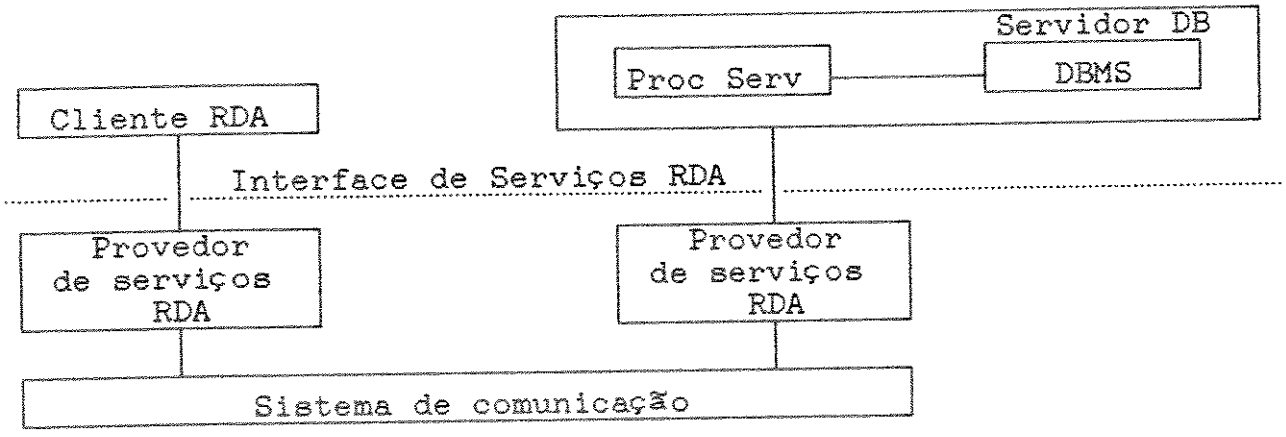


Figura 1.33 - Modelo de Funcionalidades do RDA

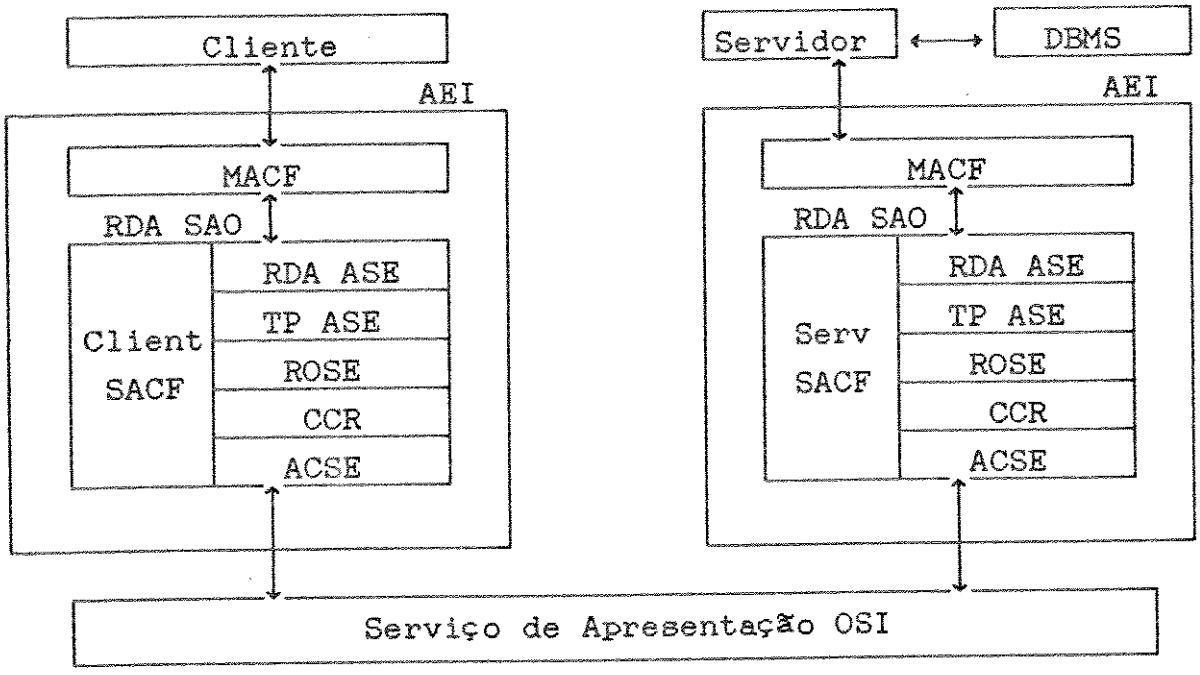


Figura 1.34 - Modelo de Serviço do RDA

O RDA tem também serviços e unidades de protocolos adicionais, para proverem mecanismos de suporte à execução de transações distribuídas. Esta versão estendida do RDA é baseada no protocolo TP. [JOH90] apresenta um exemplo de Transação RDA Distribuída, no qual pode-se ver que a interação entre os protocolos RDA, TP e CCR não é trivial.

O Modelo de Funcionalidades do RDA (baseado no modelo cliente/servidor) é similar ao Modelo do FTAM (também baseado no modelo cliente/servidor), e portanto, a API para o RDA pode ser construída de modo equivalente à API para o FTAM, no tocante às funcionalidades. Isto pode implicar na existência do bloco funcional VDRDB na estação cliente, e dos blocos funcionais IP e VDRDB na estação servidora, se a própria API responder as primitivas de indicação.

O Modelo de Serviço do RDA (composto de diversos ASEs) é similar ao Modelo do TP (também composto de diversos ASEs), e portanto, a API para o RDA pode ser construída de modo equivalente à API para o TP em relação aos serviços. Isto implica na existência dos blocos funcionais

TAOCF e RAOCF nas estações cliente e servidora.

Portanto, o Modelo da figura 1.28 pode ser utilizado para construir APIs para os Protocolos MMS, FTAM, TP e RDA, e para padrões gráficos (como por exemplo, para o GKS), de definição de produtos e de documentos de escritórios, sendo geral para as aplicações típicas de API utilizadas atualmente.

1.4.8 - MODELO GERAL DE API PARA PROTOCOLOS ASSIMÉTRICOS

Os Protocolos MMS, FTAM e RDA, como outros protocolos de aplicação, apresentam assimetrias entre as estações no que se refere à arquitetura do sistema de aplicação, e consequentemente do sistema de comunicação, segundo um modelo de estações clientes e estações servidoras.

Para o Protocolo MMS, a estação cliente pode ser uma estação supervisora de célula da manufatura com um sistema aplicativo mais complexo do que o das estações servidoras, que podem ser CLPs, CNCs, entre outras.

Para o Protocolo RDA, a estação cliente pode ser uma estação de trabalho e a estação servidora um Sistema de Base de Dados, que possua um sistema aplicativo mais complexo do que a primeira.

Isto implica na existência de sistemas de comunicação mais simples e outros mais complexos, dependendo do tipo de estação e do protocolo de aplicação.

No que se refere à Interface de Aplicação, APIs mais simples podem ser construídas utilizando o mesmo Modelo Geral proposto na figura 1.28, com a eliminação de alguns blocos funcionais e/ou algumas comunicações entre blocos, de acordo com os aspectos específicos de cada estação.

Na realidade, o Modelo proposto é mais geral do que os conceitos de estações clientes e servidoras, e portanto, pode ser aplicado a ambos. A existência de funções respondedoras na LIB, além da existência dos blocos funcionais IFA, IP e VDRDB, fornecem à API a característica de estação servidora. A existência de funções de alto e baixo nível na LIB, além da existência dos blocos funcionais HLSP e

CSP, fornecem à API a característica de estação cliente.

1.4.9 - MODELO GERAL DE API: INTERAÇÃO COM A CAMADA DE APLICAÇÃO

O Modelo Geral de API proposto (figura 1.28) é simples em relação ao suporte às Entidades de Aplicação, que possuem diversos ASEs (protocolos de aplicação). Por exemplo, num determinado computador, para acessar estas Entidades de Aplicação, necessita-se apenas de uma API. A API terá diversos componentes lógicos, porém não se necessita de diversas APIs.

A figura 1.35 mostra dois exemplos de API que suportam Entidades de Aplicação com diversos protocolos de aplicação:

a- API para os Protocolos RDA e TP;

b- API para o Protocolo MHS.

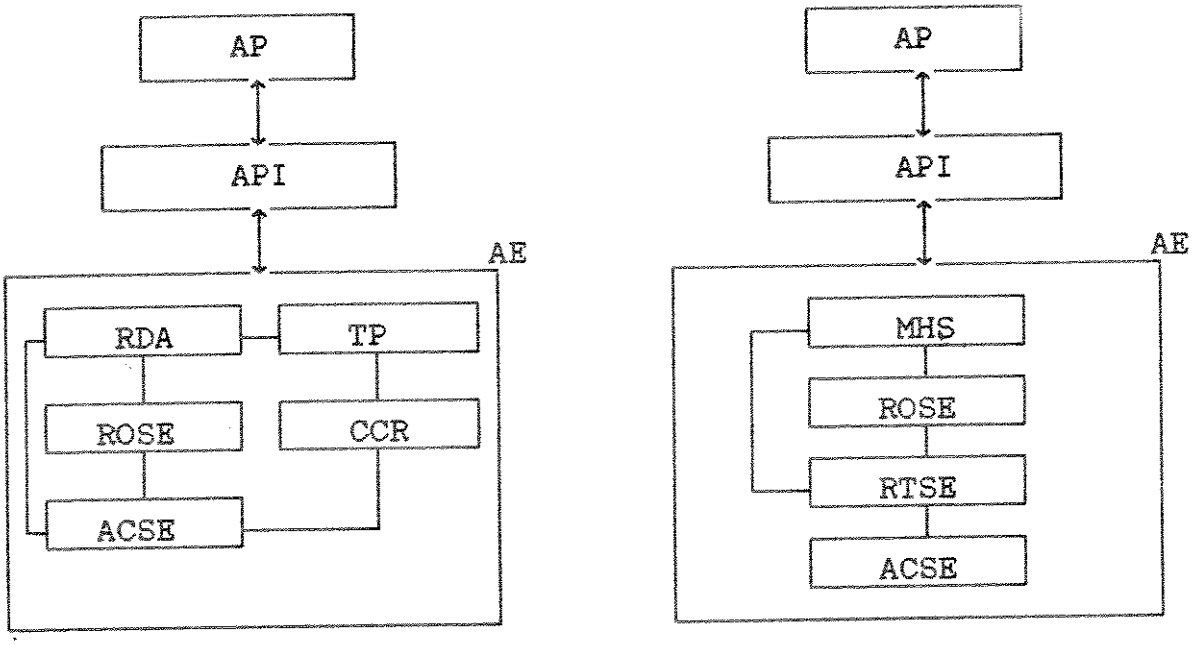


Figura 1.35

a - API para os Protocolos RDA e TP b - API para o Protocolo MHS

As interações relacionadas com os diversos ASEs, que a API controla, são interações "próximas" aos usuários. Exemplos de

interações deste tipo são aquelas que permitem ao usuário invocar uma certa seqüência de primitivas da camada de aplicação, ou invocar uma primitiva da camada de aplicação, cujo campo de informação contém uma outra primitiva da camada de aplicação pertencente a outro protocolo de aplicação. Outros tipos de interação, não "próximas" do usuário, devem ser controladas pela camada de aplicação.

Por exemplo, com o surgimento do Protocolo ROSE [ISO9072], uma Entidade de Aplicação, que possua diversos SASEs, pode tratar os serviços dos diversos SASEs como operações remotas utilizando o ROSE. O controle da interação entre os diversos ASEs e o ROSE deve ser realizado pela própria camada de aplicação (ver capítulo 4).

CAPÍTULO 2

IMPLEMENTAÇÃO DO MODELO GERAL PROPOSTO DE INTERFACES DE APLICAÇÃO

2.1 - METODOLOGIA DE DESENVOLVIMENTO DO "SOFTWARE"

A API pode ser construída a partir de métodos clássicos de desenvolvimento de "software", como por exemplo, pelo modelo "cascata" ("waterfall") de ciclo de vida [ROY70]. Neste modelo, o desenvolvimento do "software" é realizado numa seqüência de fases, em que os resultados de uma fase são os insumos para a fase seguinte. O

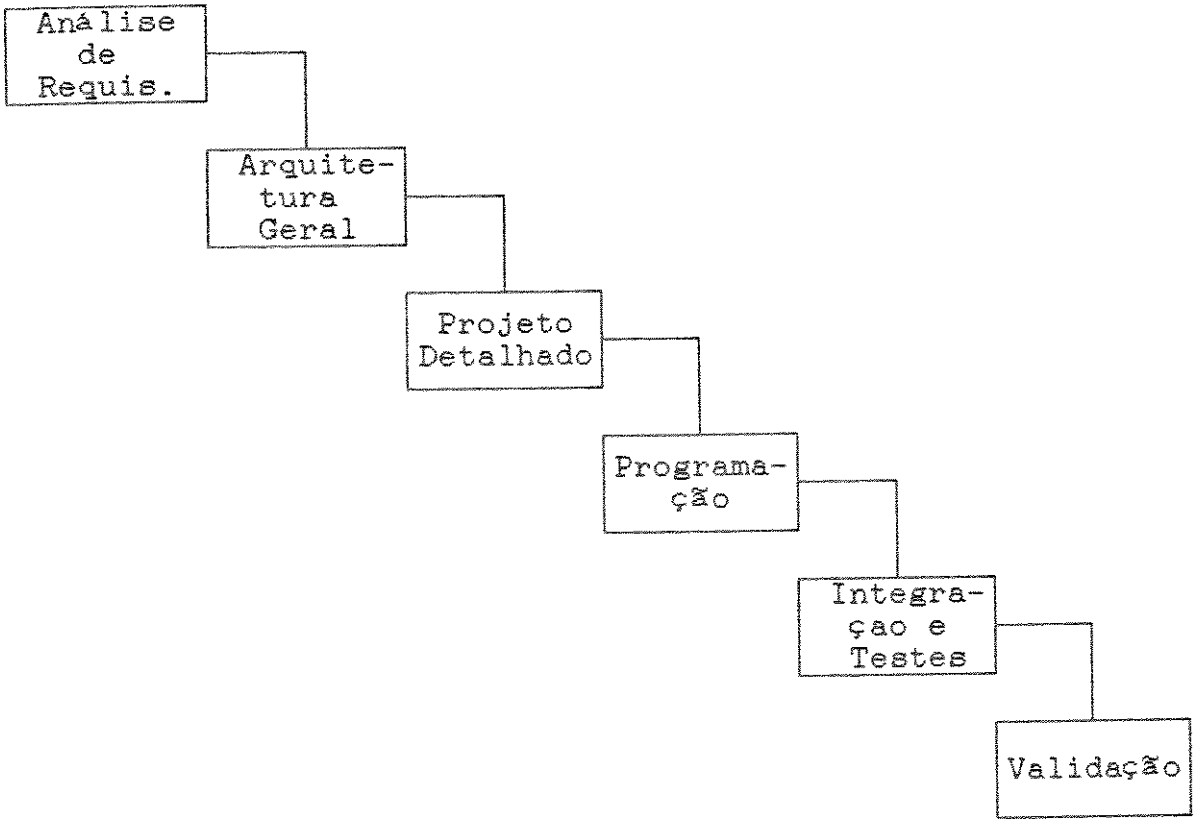


Figura 2.1 - Modelo "Cascata" de Ciclo de Vida

modelo "cascata" de ciclo de vida é apresentado na figura 2.1.

Contudo, existem outros métodos clássicos de desenvolvimento de "software" mais aprimorados, que também são adequados para a construção da API. Como exemplo, tem-se o Modelo em Espiral [BOE86], que leva em consideração o fato de que o desenvolvimento de um "software" é cíclico, e que a cada novo ciclo uma nova versão do "software" é obtida, tendo-se um novo protótipo. Cada ciclo é dividido em fases. No último ciclo obtém-se o protótipo operacional. O modelo também é comentado em [TAK90].

As Técnicas de Descrição Formal (FDTs) LOTUS [BOL87] e Estelle [BUD87] da ISO, e SDL do CCITT ("Comité Consultatif International de Télégraphique et Téléphonique") não são apropriadas para o desenvolvimento de uma especificação formal para a API. As técnicas Estelle e SDL são baseadas em máquinas de estados, enquanto que a API não as possui, pois a API não é um protocolo.

A Técnica de Descrição Formal LOTUS é baseada numa álgebra definida. O desenvolvimento da especificação é obtido através da construção e derivação de expressões a partir desta álgebra. A API pode se enquadrar nessa álgebra, mas devido ao fato de ser uma interface de serviços, a melhor maneira de descrever seu comportamento não é através de expressões.

Alguns comentários sobre metodologias de desenvolvimento de "software" aplicadas à implementação de protocolos (Engenharia de Protocolo) são apresentados em [WEB90].

2.2 - IMPLEMENTAÇÃO MODULAR DA API EM BLOCOS FUNCIONAIS

Uma rede local possui normalmente estações mais complexas e estações mais simples, como foi visto no capítulo 1, em função de serem estações clientes ou servidoras, e em função do tipo de aplicação das mesmas.

Por exemplo, nas redes locais em ambiente fabril, alguns nós da rede possuem pouca capacidade de memória e/ou de execução de programas, tais como: robôs, CLPs, entre outros (estações servidoras). Estes nós necessitam geralmente da implementação de um subconjunto do

protocolo de aplicação. Por outro lado, a estação controladora de uma célula da manufatura (estação cliente) pode necessitar da implementação de todo o protocolo de aplicação.

O fato de existirem estações mais complexas e mais simples numa mesma rede implica que a API deve ser construída de tal maneira, que possa ser implementada com todas as funções da biblioteca em alguns nós, e implementada com subconjuntos de funções (algumas unidades funcionais) em nós mais simples. Funções que executam o mesmo tipo de processamento formam uma unidade funcional. Exemplos de unidades funcionais são: Acesso às variáveis e Gerenciamento de semáforos. Dependendo do tipo de aplicação, um nó pode requerer apenas a implementação de parte de uma unidade funcional, como por exemplo, algumas funções da unidade funcional de Acesso às variáveis. Define-se um módulo como contendo as funções de uma unidade funcional ou como a parte de uma unidade funcional que deverá ser implementada num nó.

Portanto, a implementação da API deve ser modular para permitir a inclusão ou não de um módulo num determinado nó.

A seguir apresenta-se o esquema global do Modelo Geral de API proposto na figura 1.28, utilizando uma sintaxe similar à da linguagem Modula-2, para obter o conceito de implementação modular desejado. No esquema, a API como uma entidade provedora usa serviços de N ASEs distintos. Procurou-se dar nomes aos módulos, procedimentos e funções de tal modo que representem claramente as suas funcionalidades. O esquema possui oito Módulos Globais:

- Fila, Alocação e Estrutura_Dados: possuem funções de manipulação de filas, alocação de memória e estruturas de dados requeridas pelos outros módulos;

- Biblioteca, Procedimentos_de_Controle e Provedor_Serviços_Primitivas: possuem as funcionalidades das três partes conceituais da API. O primeiro módulo representa o bloco funcional LIB do Modelo Geral proposto na figura 1.28, enquanto que o segundo representa os blocos funcionais HLSP, CSP, IFA, IP e VDRDB, e o terceiro representa o bloco funcional PSP;

- TAOCF e RAOCF: possuem as funções que controlam a interação dos diversos ASEs a nível de API. Estes módulos representam

os blocos funcionais TAOCF e RAOCF do Modelo Geral.

```
Module Interface_de Aplicação;
     Module Alocação;
          procedure aloca;
          procedure desaloca;
              :
     Module Fila;
          function Insere;
          function Retira;
              :
     Module Estrutura_Dados;
          procedure obtém_recurso;
          procedure retorna_recurso;
          procedure verifica_recurso;
              :
     Module TAOCF;          { 1a Parte }
          procedure função1_TAOCF;
          procedure função2_TAOCF;
              :
          procedure funçãoN1_TAOCF;
     Module Biblioteca;  { 2a Parte }
          function Verifica_Parametros_Universais;
          Module ASE1_Alto_Nível;
               function ASE1_AN_função1_Verif_Param;
               procedure ASE1_AN_função1;
                  :
               function ASE1_AN_funçãoN2_Verif_Param;
               procedure ASE1_AN_funçãoN2;
          Module ASE1_Baixo_Nível;
               function ASE1_BN_função1_Verif_Param;
               procedure ASE1_BN_função1;
                  :
               function ASE1_BN_funçãoN3_Verif_Param;
               procedure ASE3_BN_funçãoN3;
```

```
Module ASE1_Serviço_Respondedor;
      function ASE1_SR_função1_Verif_Param;
      procedure ASE1_SR_função1;
         :
      function ASE1_SR_funçãoN4_Verif_Param;
      procedure ASE1_SR_funçãoN4;
         :
Module ASEN_Alto_Nível;
      function ASEN_AN_função1_Verif_Param;
      procedure ASEN_AN_função1;
         :
      function ASEN_AN_funçãoN5_Verif_Param;
      procedure ASEN_AN_funçãoN5;
Module ASEN_Baixo_Nível;
      function ASEN_BN_função1_Verif_Param;
      procedure ASEN_BN_função1;
         :
      function ASEN_BN_funçãoN6_Verif_Param;
      procedure ASEN_BN_funçãoN6;
Module ASEN_Serviço_Respondedor;
      function ASEN_SR_função1_Verif_Param;
      procedure ASEN_SR_função1;
         :
      function ASEN_SR_funçãoN7_Verif_Param;
      procedure ASEN_SR_funçãoN7;
Module Funções_Auxiliares;
      Module Funções_Auxiliares_Gerais;
            function G_FA_função1_Verif_Param;
            procedure G_FA_função1;
               :
            function G_FA_funçãoN8_Verif_Param;
            procedure G_FA_funçãoN8;
      Module ASE1_Funções_Auxiliares
            function ASE1_FA_função1_Verif_Param;
            procedure ASE1_FA_função1;
```

```
                    :
                function ASE1_FA_funçãoN9_Verif_Param;
                procedure ASE1_FA_funçãoN9;
                    :
            Module ASEN_Funções_Auxiliares;
                function ASEN_FA_função1_Verif_Param;
                procedure ASEN_FA_função1;
                    :
                function ASEN_FA_funçãoN10_Verif_Param;
                procedure ASEN_FA_funçãoN10;
        Module Funções_Suporte;
            function FS_função1_Verif_Param;
            procedure FS_função1;
                :
            function FS_funçãoN11_Verif_Param;
            procedure FS_funçãoN11;
Module Procedimentos_de_Controle;        { 3a Parte }
    procedure HLSP;
    procedure CSP;
    procedure IFA;
        procedure Requisição_RS_LIB;
        procedure Solicitação_LLS_LIB;
        procedure Solicitação_HLSP;
        procedure Indicação_RAOCF;
    procedure IP;
        procedure Requisita_Resposta_CSP;
        procedure Requisita_Resposta_PSP;
        procedure Requisita_VDRDB;
        procedure Processa_Indicação;
    procedure VDRDB;
        Module Mapeamentos;
            procedure Virt_Real_Mapeam1;
                :
            procedure Virt_Real_MapeamN12;
            procedure Real_Virt_Mapeam1;
```

```
                    :
                procedure Real_Virt_MapeamN13;
            Module Formatos;
                procedure Formato1_função1;
                    :
                procedure Formato1_funçãoN14;
                procedure FormatoN15_função1;
                    :
                procedure FormatoN15_funçãoN16;
    Module RAOCF;          { 4a Parte }
        procedure função1_RAOCF;
        procedure função2_RAOCF;
            :
        procedure funçãoN17_RAOCF;
    Module Provedor_Serviços_Primitivas;
        Module ASE1_Recebe_CNF_IND;
            procedure ASE1_Primitiva_CNF;
            procedure ASE1_Primitiva_IND;
                :
        Module ASEN_Recebe_CNF_IND;
            procedure ASEN_Primitiva_CNF;
            procedure ASEN_Primitiva_IND;
        Module ASE1_Envia;
            procedure ASE1_Envia_REQ_RSP;
                :
        Module ASEN_Envia;
            procedure ASEN_Envia_REQ_RSP;
        Module Solicitação;
            procedure ASE1_Solicitação_CNF_IND;
                :
            procedure ASEN_Solicitação_CNF_IND;
```

onde:

N1: número de funções do bloco funcional TAOCF;

N2: número de funções de alto nível relacionadas com o ASE 1;

N3: número de funções de baixo nível relacionadas com o ASE 1;

N4: número de funções respondedoras relacionadas com o ASE 1;

N5: número de funções de alto nível relacionadas com o ASE N;

N6: número de funções de baixo nível relacionadas com o ASE N;

N7: número de funções respondedoras relacionadas com o ASE N;

N8: número de funções auxiliares gerais (não relacionadas com um ASE específico);

N9: número de funções auxiliares relacionadas com o ASE 1;

N10: número de funções auxiliares relacionadas com o ASE N;

N11: número de funções de suporte;

N12: número de funções de mapeamento dispositivo virtual / dispositivo real;

N13: número de funções de mapeamento dispositivo real / dispositivo virtual;

N14: número de funções de manipulação do formato de dados 1;

N15: número de formatos de dados suportados pela API;

N16: número de funções de manipulação do formato de dados N15;

N17: número de funções do bloco funcional RAOCF.

Para a construção do Esquema Geral, levou-se em consideração que:

a- Para cada função da Biblioteca deve ser verificado se cada parâmetro da função tem um valor válido, seja este parâmetro universal ou específico;

b- As funções da Biblioteca são classificadas como:

- Pertencentes a um ASE específico;
- Funções auxiliares gerais ou específicas de um ASE;
- Funções de suporte;

c- As funções pertencentes a um ASE específico podem ser classificadas como:

- Funções de alto nível;
- Funções de baixo nível;
- Funções respondedoras;

d- O procedimento de controle IFA tem quatro funcionalidades:

- Atender a uma requisição do Serviço Respondedor da LIB (Biblioteca);
- Atender a uma solicitação do Serviço de Baixo Nível

da LIB;

- Atender a uma solicitação do HLSP;
- Processar uma primitiva de indicação recebida do RAOCF;

e- O Processador de Indicações (IP) possui quatro procedimentos para:

- Receber uma primitiva de indicação do IFA, processar este serviço chamando o VDRDB quando é necessário, e requisitar o envio da primitiva de resposta;
- Solicitar o envio de resposta a uma indicação para o CSP através de uma primitiva de requisição associada a um serviço confirmado;
- Solicitar o envio de resposta a uma indicação para o PSP através de uma primitiva de resposta ou requisição. No segundo caso, a primitiva de requisição está associada a um serviço não confirmado;
- Solicitar os serviços de mapeamento do VDRDB;

f- O VDRDB possui procedimentos de mapeamento dispositivo virtual / dispositivo real e vice-versa, e procedimentos para manipular os formatos de dados definidos na API;

g- O Provedor de Serviços de Primitivas possui procedimentos para:

- Receber primitivas de confirmação e indicação das Máquinas de Protocolo dos diversos ASEs;
- Enviar primitivas de requisição e resposta para as Máquinas de Protocolo dos diversos ASEs;
- Atender à solicitação de espera pela recepção de primitivas de confirmação e indicação.

A API pode ser implementada como uma única instância, ou como tendo diversas instâncias. No primeiro caso tem-se uma implementação mais simples da API, enquanto que no segundo tem-se uma maior independência das requisições de serviços de comunicação por parte dos APs, e dos recursos utilizados pela API.

Neste segundo caso, pode-se ter uma instância da API para cada

associação, ou uma instância para cada invocação de Entidade de Aplicação. Sabe-se que a troca de processo em execução implica em guardar o contexto do processo que estava em execução, e em restaurar o contexto do processo a ser executado. A troca de contexto é custosa. A solução de ter uma instância para cada invocação, à primeira vista, parece ser uma solução viável para a opção do sistema ter mais de uma instância da API. É uma solução intermediária.

Contudo, alguns blocos funcionais da API são específicos da associação, o que pode sugerir a existência de uma instância da API para cada associação.

Entretanto, a solução definitiva depende das características dos sistemas aplicativo, de comunicação e operacional locais.

2.3 - IMPLEMENTAÇÃO DOS BLOCOS FUNCIONAIS

Os blocos funcionais que formam o Modelo Geral proposto de API (figura 1.28) podem ser implementados como procedimentos da tarefa API ou como uma ou mais tarefas distintas. Pode-se para algumas tarefas criar uma instância para cada Entidade de Aplicação, para outras uma instância para cada invocação da Entidade de Aplicação, ou ainda uma instância para cada associação.

Para escrever o código de cada bloco funcional para uma API específica, é interessante analisar passo a passo a execução de cada função da API em cada bloco funcional.

Na seção 1.4.5 foi apresentado o exemplo do fluxo de informação entre os blocos funcionais do Modelo Geral proposto para a requisição de estabelecimento de associação por parte de um AP, ao chamar a função CONNECT da API.

A seguir é apresentada, a título de exemplo, a execução passo a passo da função CONNECT (função de alto nível) nos blocos funcionais LIB, HLSP, CSP e PSP para uma API, que é usuária dos serviços dos Protocolos de Aplicação FTAM e ACSE. Portanto, a API possui logicamente os componentes API-CM e API-FTAM. A seqüência de passos que compõem a execução dos blocos funcionais RAOCF e TAOCF, que coordenam as interações entre a API-CM e a API-FTAM, foi apresentada

no exemplo da seção 1.4.5.

Tabela 2.I - Função CONNECT da API associada aos Protocolos FTAM e ACSE

LIB - CM	LIB - FTAM
1. Verifica o parâmetro universal RETURN_EVENT_NAME, e se há recurso disponível na API para a chamada da função; 2. Verifica os parâmetros específicos da função, relacionados com o controle da associação: CALLED_PRESENTATION_ADDRESS, CALLED_DIR_NAME, AE_LABEL, CONTEXT_NAME, CALLED_AE_TITLE_OPTION e CALLED_AE_TITLE;	3. Verifica os parâmetros específicos da função, relacionados com o Protocolo FTAM: SERVICE_LEVEL, SERVICE_CLASS, FUNCTIONAL_UNITS, ATTRIBUTE_GROUPS, CONTENTS_TYPES_LIST, IDENTITY_OF_INITIATOR, ACCOUNT e FILESTORE_PASSWORD;

HLSP - FTAM	CSP - FTAM
4. Envia solicitação para o CSP-FTAM gerar a primitiva F_INITIALIZE.req;	

Tabela 2.I - Função CONNECT da API associada aos Protocolos FTAM e ACSE (cont.)

HLSP - FTAM	CSP - FTAM
	5. Formata o F_INITIALIZE.req com os parâmetros da função CONNECT. Se os parâmetros SERVICE_LEVEL, SERVICE_CLASS, FUNCTIONAL_UNITS e ATTRIBUTE_GROUPS não tiveram valores especificados, preenche com valores "defaults";
6. Recebe a primitiva F_INITIALIZE.req formatada; 7. Inclui a primitiva F_INITIALIZE.req no campo de dados do A_ASSOCIATE. Parâmetros do A_ASSOCIATE relacionados com o ACSE e o FTAM são preenchidos: PRESENTATION_CONTEXT_DEFINITION_LIST, PRESENTATION_REQUIREMENTS e SESSION_REQUIREMENTS;	

HLSP - CM	CSP - CM	PSP
8. Recebe primitiva de A_ASSOCIATE.req da API-FTAM; 9. Use a informação recebida para enviar um A_ASSOCIATE.req síncrono para o CSP, e obtenha a informação de retorno;		

Tabela 2.I - Função CONNECT da API associada aos Protocolos FTAM e ACSE (cont.)

HLSP - CM	CSP - CM	PSP
	10. Verifica se o número de associações por invocação de Entidade de Aplicação não foi excedido. Neste caso, incrementa o contador do número de associações desta invocação de AE; 11. Espera pela alocação de recursos para a conexão; 12. Obtém o valor para o parâmetro CALLED_AE_TITLE da primitiva A_ASSOCIATE.req; 13. Passa o A_ASSOCIATE.req para o PSP; 14. Espera pela resposta do PSP;	15. Verifica se há indicação de aborto recebida; 16. Se não há, envia primitiva A_ASSOCIATE.req; 17. Sinaliza ao CSP-CM sobre o envio;

Tabela 2.I - Função CONNECT da API associada aos Protocolos FTAM e ACSE (cont.)

HLSP - CM	CSP - CM	PSP
	18. Solicita primitiva de confirmação ao PSP; 19. Espera pela resposta do PSP;	20. Recebe primitiva A_ASSOCIATE.cnf; 21. Verifica se há indicação de aborto recebida; 22. Se não há, a primitiva recebida é entregue ao RAOCF;

HLSP - FTAM	CSP - FTAM	CSP - CM
23. Recebe a primitiva F_INITIALIZE.cnf do RAOCF; 24. Passa a primitiva F_INITIALIZE.cnf para o CSP-FTAM;	25. Recebe a primitiva F_INITIALIZE.cnf, e preenche os parâmetros de saída da função CONNECT com valores obtidos nos parâmetros correspondentes da primitiva;	

Tabela 2.I - Função CONNECT da API associada aos Protocolos FTAM e ACSE (cont.)

HLSP - FTAM	CSP - FTAM	CSP - CM
		26. Coloca toda a informação de saída na área de saída do usuário. Se a área foi pré-alocada, e não é grande o suficiente, indica OUTPUT_BUFFER_OVERFLOW; 27. Se a conexão não foi estabelecida, decrementa o contador do número de associações desta invocação de AE; 28. Sinaliza o Evento de Retorno, chamando o HLSP-CM;

HLSP - CM
29. Se a informação de retorno indica sucesso, atribua CONNECTION_ID à conexão, e sinalize o Evento de Retorno; 30. Se não indica sucesso: se NUMBER_OF_RETRY é nulo, ou há erro de OUTPUT_BUFFER_OVERFLOW, AABORT_INDICATION_RECEIVE, CONFIRMATION_FAILED ou um erro de confirmação negativa específico do FTAM, então indique o erro, e sinalize o Evento de Retorno; 31. Se o Evento de Retorno não foi sinalizado, faça RETRY_COUNTER igual ao parâmetro NUMBER_OF_RETRY, e repita os passos seguintes até o Evento de Retorno ser sinalizado:

Tabela 2.I - Função CONNECT da API associada aos Protocolos FTAM e ACSE (cont.)

HLSP - CM
a- Espere o tempo especificado pelo parâmetro DELAY_BETWEEN_RETRY; b- Envie um A_ASSOCIATE.req síncrono para o CSP-CM, e obtenha a informação de retorno; c- Se a informação de retorno indica sucesso, atribua CONNECTION_ID à conexão, e sinalize o Evento de Retorno; d- Decremente RETRY_COUNTER; e- Se - a informação de retorno não indica sucesso, e RETRY_COUNTER é zero ou - há erro de OUTPUT_BUFFER_OVERFLOW, AABORT_INDICATION_RECEIVE, APABORT_INDICATION_RECEIVE ou ASE_SPECIFIC_NEGOTIATION_FAILURE então indique o erro, e sinalize o Evento de Retorno.

A Tabela 2.I mostra as ações dos blocos funcionais LIB-CM, LIB-FTAM, HLSP-CM, HLSP-FTAM, CSP-CM, CSP-FTAM e PSP para a função CONNECT da API para o FTAM. A API envia para a camada de aplicação uma primitiva A_ASSOCIATE.req (que é utilizada pelo Protocolo ACSE para o estabelecimento da conexão), que em seu campo de informação contém uma primitiva F_INITIALIZE.req (que é utilizada pelo Protocolo FTAM para estabelecer o contexto da conexão relativo ao FTAM). Cada coluna da Tabela representa as ações de um bloco funcional, e cada linha representa uma ação (ordenada no tempo), que a API deverá executar.

A conclusão da análise passo a passo da execução das diversas funções da API em cada bloco funcional é que, para cada bloco funcional, existe código a ser executado que:

- independe de qual função da Biblioteca da API foi chamada;

- é comum às funções pertencentes a um grupo de funções da Biblioteca;

- é particular a uma função da Biblioteca.

A implementação dos blocos funcionais deve levar este fato em consideração. Isto influi na estrutura dos procedimentos dos blocos funcionais, sejam os blocos tarefas ou procedimentos da tarefa API.

2.4 - IMPLEMENTAÇÃO DE APIs UTILIZANDO LINGUAGENS ORIENTADAS A OBJETOS

A especificação [MAP88b] comenta que a aplicabilidade da API para linguagens não procedurais, como por exemplo, para algumas linguagens orientadas a objetos, é ainda indeterminada. A seguir apresenta-se alguns comentários sobre a implementação de uma API utilizando uma linguagem orientada a objetos.

a- O modelo de referência OSI/ISO, baseado em camadas, possui um nível alto de abstração. O mesmo ocorre com a API que é composta por diversas unidades funcionais. A existência de uma abstração de alto nível implica geralmente num bom emprego de tipos abstratos de dados, ao invés dos tipos convencionais. Contudo, tanto no modelo de referência quanto no modelo de API, as diversas entidades (unidades), que compõem os modelos, são autônomas, caracterizando um ambiente concorrente. Os tipos abstratos de dados não refletem esta característica de ambiente concorrente na sua totalidade. Os tipos abstratos de dados também não são adequados para um sistema em execução [TAK90]. Estes dois motivos levam a se pensar no emprego do Paradigma de Objetos no desenvolvimento de sistemas baseados no modelo de referência e/ou da API. Desta maneira, utilizando uma linguagem orientada a objetos, seriam definidas classes para modelar os conceitos da aplicação, em vez de definir tipos abstratos de dados;

b- Nas linguagens orientadas a objetos, as classes são estruturadas em hierarquias de generalização / especificação, e existe o conceito de herança. Estes conceitos não são aplicáveis em toda a sua potencialidade para o desenvolvimento da API, pois a API possui apenas parte deles. Por exemplo, uma função da Biblioteca da API pertence à classe de funções de alto nível, ou à classe de funções de

baixo nível ou à classe de funções respondedoras.

c- As linguagens orientadas a objetos também possuem os conceitos de objetos (instâncias de classes), métodos e mensagens. Objetos enviam mensagens para outros objetos, e obtêm respostas. A comunicação entre objetos (através de mensagens) é realizada de maneira assíncrona e concorrente. O método ativado por uma mensagem depende do objeto recipiente e da mensagem. A API possui diversas unidades funcionais que: são autônomas, podem funcionar concorrentemente, de alguma forma trocam mensagens entre si e cada unidade possui diversas funções. Portanto, os conceitos das linguagens orientadas a objetos apresentados neste ítem têm grande potencialidade de uso.

Os conceitos das linguagens orientadas a objetos podem ajudar, por outro lado, a resolver o problema de desenvolvimento de interfaces entre as camadas do Modelo de Referência, e entre a camada de aplicação e a API, através da definição de classes e objetos convenientes, além do envio de mensagens entre estes objetos. Como as camadas do Modelo OSI/ISO são especificadas separadamente, a construção das interfaces entre as camadas sempre constitui um problema a ser pensado.

2.5 - IMPLEMENTAÇÃO NUM AMBIENTE MULTITAREFA: SISTEMA OPERACIONAL UNIX

A especificação da API independe do sistema operacional, a menos de uma interface simples de conexão. Todo o restante da API permanece idêntico, se trocarmos o sistema operacional. Esta pequena interface com o sistema operacional consiste das funções WAIT e NOTE da API descritas na seção 1.3.3.

O sistema local é composto pelos diversos processos de aplicação (APs), a API (um ou mais processos) e o sistema de comunicação. A troca do sistema operacional local deve também provocar algumas alterações no sistema local, em relação a como é realizada a criação e o escalonamento dos diversos processos, e como são passadas as informações entre os processos.

Portanto, a troca do sistema operacional local implica na troca das funções WAIT e NOTE da API, além de alterações nas partes da API

que tratam do gerenciamento dos processos (que pertencem à API ou que interagem com a API), e nas partes que tratam da troca de mensagens entre estes processos (ver também a seção 3.2.2.1).

Nesta seção são apresentadas algumas idéias para a implementação de uma API sobre o sistema operacional UNIX. O UNIX possui as seguintes chamadas de sistema com a finalidade de controle de processos:

a- fork: cria uma cópia do processo que estava executando. O processo que executou o "fork" é o processo pai, e o processo criado é o processo filho. A execução do "fork" é a única maneira de criar um novo processo;

b- exit: termina a execução de um processo. O retorno é feito ao núcleo do UNIX, e não ao processo chamador;

c- exec: executa um programa. O UNIX substitui o processo corrente pelo novo programa;

d- wait: espera pelo término de execução de um de seus processos filhos. Se nenhum dos processos filhos terminou, o núcleo do UNIX suspende o processo pai até que um dos processos filhos termine.

Um AP (que é um processo), ao desejar executar a API, cria um novo processo utilizando a operação "fork". O processo filho criado, que é uma cópia do AP, chama a operação "exec" para executar o novo programa, que é a API. O processo pai (AP) pode continuar executando seu processamento específico. O processo filho (que executa o programa API), ao terminar seu processamento, chama a operação "exit". O processo pai (AP), para esperar pelo término do processo filho (API), executa a operação "wait".

A API pode ser implementada como um processo, ou como vários processos. Tanto num caso como no outro, cada processo (no primeiro caso o único processo, e no segundo caso qualquer um dos processos que compõem a API) pode desejar fazer uma cópia de si mesmo, com a finalidade de que um processo execute uma operação, enquanto o outro execute outra função. Para alcançar este objetivo, o processo deve executar a operação "fork".

Por exemplo, considere-se como um processo a unidade funcional CSP da API. Neste caso, o processo CSP pode executar várias operações

"fork" para criar diversos processos filhos, em que cada processo filho trata dos serviços confirmados em cada uma das associações existentes na API. Cada processo filho corresponde a uma instância do processo CSP.

O UNIX apresenta os seguintes métodos para comunicação entre processos:

a- "Pipe": só pode ser usado entre processos que tenham um processo pai em comum;

b- "Fifo" ("Named Pipe"): similar ao "Pipe", mas possui um nome associado, permitindo que processos não relacionados o utilizem;

Para que um processo escreva num "Pipe" ou "Fifo" é necessário que outro processo tenha aberto anteriormente o "Pipe" ou "Fifo" para leitura.

c- "Message Queue" (Fila de Mensagens): similar à "Fifo", mas sem a restrição de abertura da "Fifo", citada acima;

d- Memória Compartilhada: único método em que as mensagens trocadas entre processos não passam pelo núcleo do UNIX. O uso de semáforos pode resolver o problema de sincronização dos acessos aos segmentos compartilhados de memória.

Dependendo da implementação do sistema local (APs, API, sistema de comunicação) e da API (um ou vários processos), um ou mais dos métodos acima apresentados podem ser empregados.

Como o AP e a API são de naturezas diferentes, e o AP pode enviar mensagens para a API sem que a mesma esteja esperando, pode-se utilizar como método de comunicação entre o AP e a API o de "Message Queue" (Fila de Mensagens).

Internamente à API, os processos que a compõem (partes da API e/ou instâncias) podem ter mesma "ascendência", e geralmente têm sincronização entre seus processamentos. Portanto, pode-se utilizar os métodos de comunicação "Pipe" e "Fifo" para estes processos.

2.6 - IMPLEMENTAÇÃO NUM AMBIENTE MULTITAREFA: SINCRONIZAÇÃO ENTRE PROCESSOS

2.6.1 - INTRODUÇÃO

A API requer um ambiente multitarefa que suporte coordenação e comunicação entre tarefas, para a sua execução. A coordenação e comunicação entre tarefas deve:

a- Garantir a consistência das variáveis compartilhadas;

b- Possuir um esquema eficiente de escalonamento de tarefas.

As soluções encontradas podem ser classificadas em quatro grupos. Esta classificação é baseada em idéias apresentadas em [MAD85].

2.6.2 - SINCRONIZAÇÃO ATRAVÉS DE SEMÁFOROS, EVENTOS, CONDIÇÕES E MONITORES

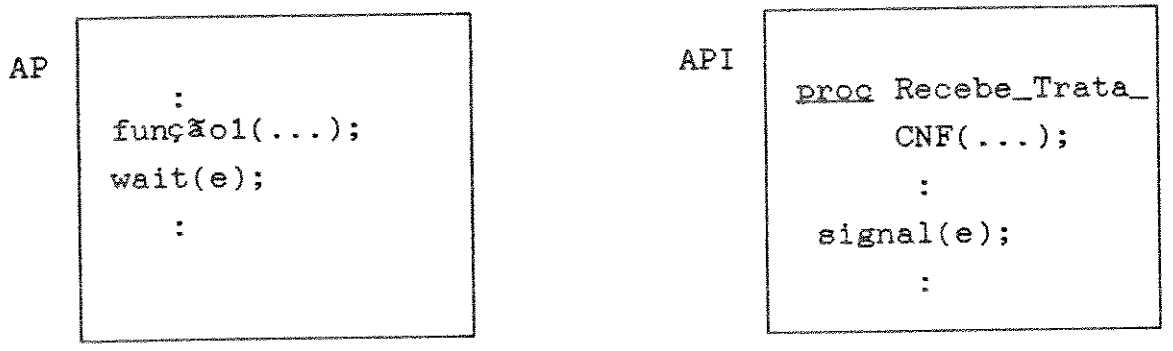
Estas soluções são apresentadas por Dijkstra em [DIJ65] e [DIJ71], Brinch Hansen em [BRI72], Wirth em [WIR77] e Hoare em [HOA72].

Por exemplo, a proposta de Brinch Hansen utiliza o conceito de eventos, e tem os comandos wait(e) e signal(e). O comando wait(e) bloqueia o processo à espera do evento e, e coloca-o numa fila de processos que esperam por e. O comando signal(e) notifica um outro processo (da fila de espera por e) da ocorrência do evento e.

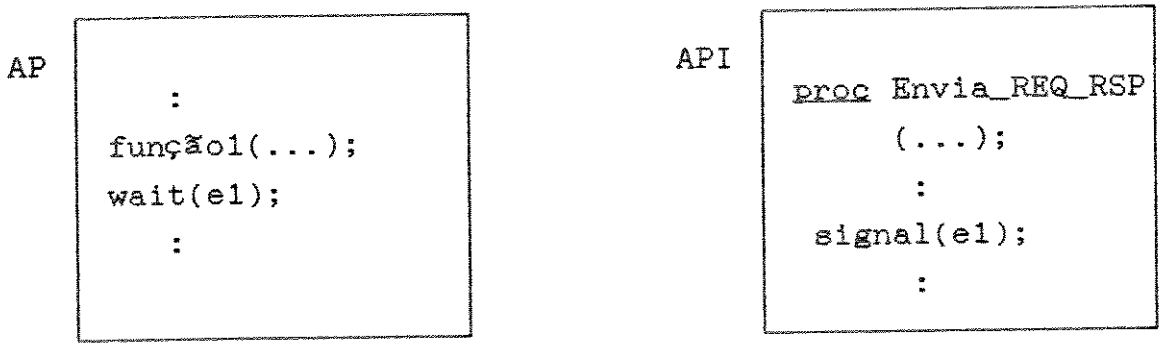
O AP, ao chamar uma função da API de modo síncrono, executa o comando wait(e), ficando bloqueado até obter a resposta do par remoto. A API recebe as primitivas de confirmação correspondentes, e após tratá-las, executa o comando signal(e) (Figura 2.2-a).

O AP, ao chamar uma função da API de modo assíncrono, também executa wait(e). A API, após enviar as primitivas de requisição ou resposta correspondentes, executa signal(e), e o AP é desbloqueado para executar suas tarefas (Figura 2.2-b).

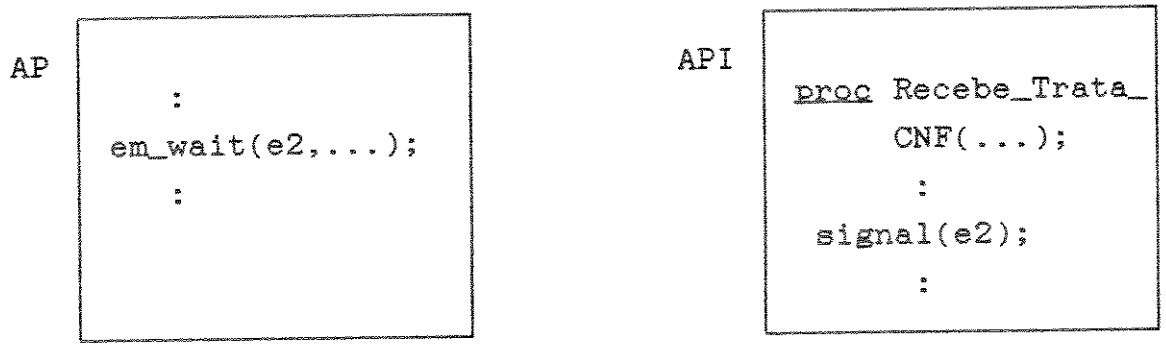
Quando o AP chamar a função em_wait(e), o AP fica bloqueado até obter uma resposta da API. A resposta é dada em função da chegada de uma das primitivas de confirmação esperadas pela API, que completa a execução de um dos serviços solicitados pelo AP de modo assíncrono ainda não respondidos (Figura 2.2-c).



(a) - Chamada da função1 de Modo Síncrono



(b) - Chamada da função1 de Modo Assíncrono



(c) - Chamada da Função em_wait

Figura 2.2 - Sincronização através de Eventos

2.6.3 - SINCRONIZAÇÃO ATRAVÉS DE TROCA DE MENSAGENS

Algumas das soluções deste tipo são apresentadas por Brinch Hansen em [BRI73] e Harland em [HAR81].

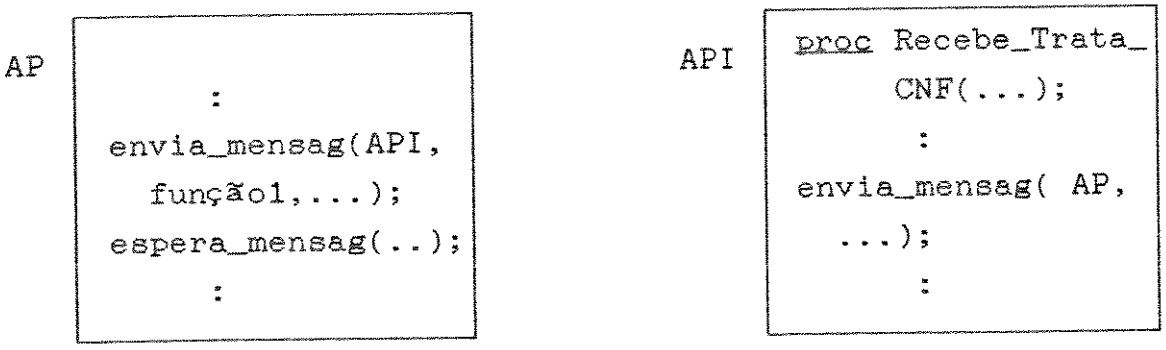
Exemplos de primitivas utilizadas nestes sistemas são: send_message(receiver, message, buffer), wait_answer(result, answer, buffer), wait_message(sender, message, buffer), wait_event, entre outras.

Implementações baseadas em "mailbox" estão incluídas neste grupo, e possuem primitivas do tipo: wait_message(mailbox, pointer) e send_message(mailbox, pointer), e também filas de processos associados ao "mailbox".

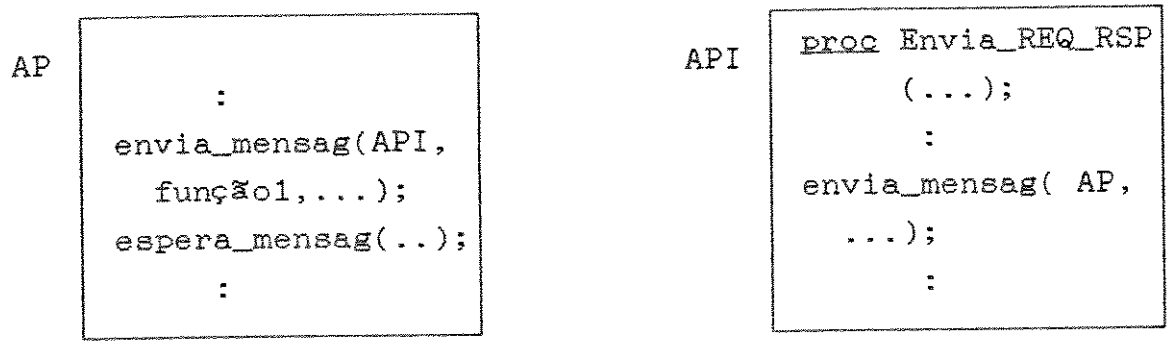
O AP, ao desejar executar uma função da API de modo síncrono, envia uma mensagem para a API, cujo conteúdo é o nome da função e seus parâmetros. O AP fica bloqueado, esperando uma mensagem de resposta da API. A API recebe as primitivas de confirmação correspondentes, e após tratá-las, envia a mensagem de resposta (Figura 2.3-a).

O AP, ao desejar executar uma função da API de modo assíncrono, também envia uma mensagem para a API, e fica esperando por uma mensagem de resposta da API. A API, ao enviar as primitivas de requisição e resposta correspondentes, envia a mensagem de resposta ao AP, desbloqueando-o (Figura 2.3-b).

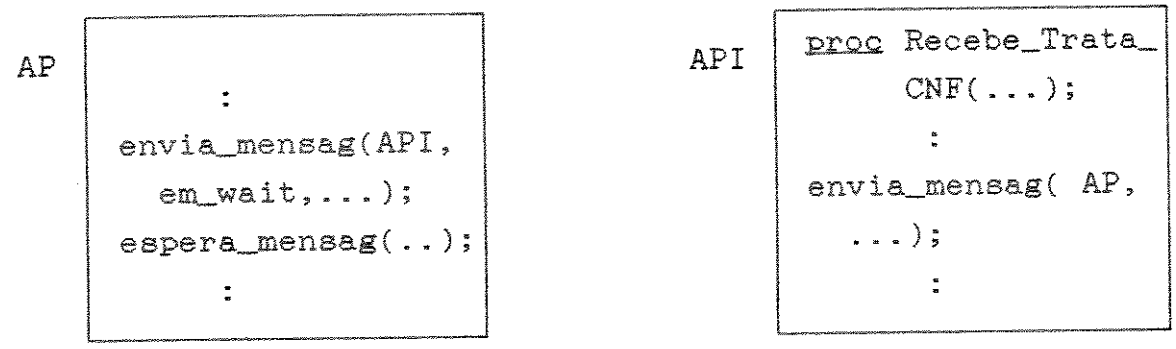
Quando o AP desejar executar a função em_wait da API, envia uma mensagem para a API, cujo conteúdo é o nome em_wait da função e seus parâmetros. O AP fica bloqueado à espera de uma mensagem da API. A mensagem de resposta da API para o AP é enviada depois que a API recebe e trata as primitivas de confirmação correspondentes a uma função qualquer da API ainda não respondida, que foi chamada de modo assíncrono (Figura 2.3-c).



(a) - Chamada da função1 de Modo Síncrono



(b) - Chamada da função1 de Modo Assíncrono



(c) - Chamada da Função em_wait

Figura 2.3 - Troca de Mensagens

2.6.4 -SINCRONIZAÇÃO ATRAVÉS DE "RENDEZ-VOUS"

Esta solução é apresentada na linguagem CSP por Hoare em [HOA78], e na linguagem ADA [ICH79] e [WEG79].

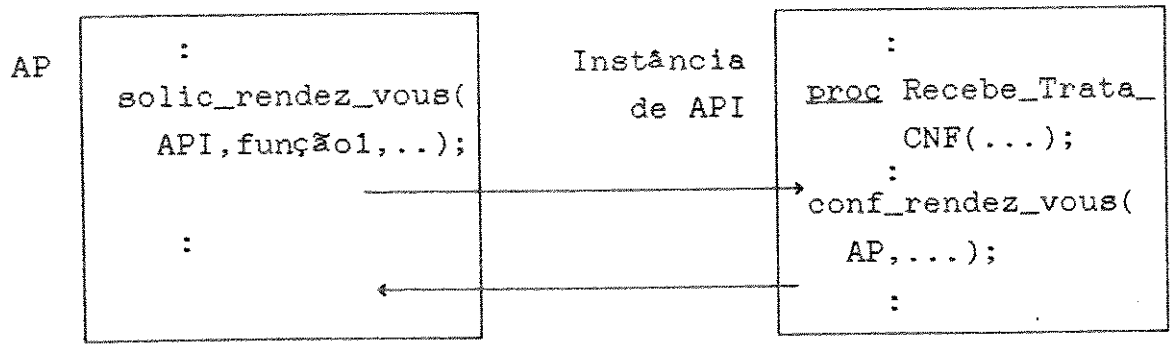
As soluções deste tipo implicam no envio de mensagens entre processos, e na espera de ocorrência do "rendez-vous" (encontro) entre estes processos.

Um processo A em determinado ponto, ao necessitar de outro processo B, envia a este segundo um pedido de "rendez-vous", e aguarda a resposta. O processo B, quando tiver disponível a mensagem necessária para a continuidade do processo A, confirma o "rendez-vous", enviando uma mensagem. Neste ponto, os dois processos se "encontram", ou seja, realizam o "rendez-vous". Depois deste encontro (sincronização), os dois processos podem continuar seus respectivos processamentos.

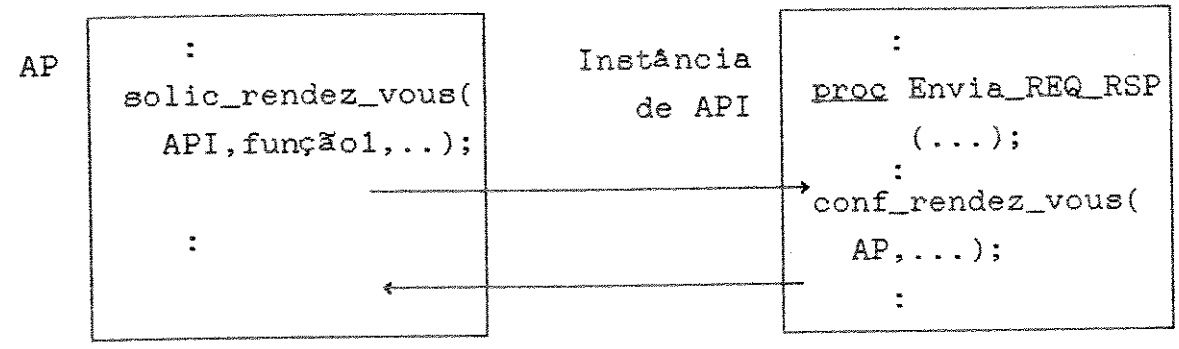
O AP, ao desejar executar uma função da API de modo síncrono, envia uma mensagem de pedido de "rendez-vous" para uma instância da API, cujo conteúdo é o nome da função e seus parâmetros. Esta instância da API, ao obter a resposta para o AP, após a recepção e o tratamento das primitivas de confirmação associadas, envia a mensagem de "rendez-vous" para o AP (Figura 2.4-a).

O AP, ao desejar executar uma função da API de modo assíncrono, também envia uma mensagem de pedido de "rendez-vous" para uma instância da API. Esta instância, ao obter a resposta para o AP, após o envio de primitivas de requisição e resposta associadas, envia a mensagem de "rendez-vous" para o AP (Figura 2.4-b).

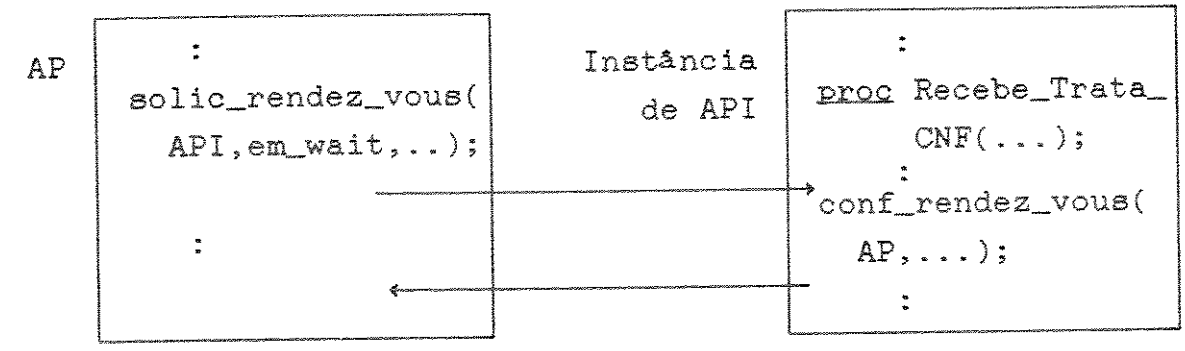
Quando o AP desejar executar a função em_wait da API, o AP envia uma mensagem de pedido de "rendez-vous" para uma instância da API, cujo conteúdo é o nome da função em_wait e seus parâmetros. A mensagem de "rendez-vous" é enviada desta instância da API para o AP, quando a instância receber e tratar as primitivas de confirmação correspondentes a uma função qualquer ainda não respondida, que foi chamada por este AP de modo assíncrono (Figura 2.4-c).



(a) - função1 chamada de modo síncrono



(b) - função1 chamada de modo assíncrono



(c) - Função em_wait

Figura 2.4 - "Rendez-Vous"

2.6.5 - SINCRONIZAÇÃO ATRAVÉS DE TRANSFERÊNCIA DE CONTROLE ENTRE CORROTINAS

Esta solução é apresentada na linguagem Modula-2 por Wirth [WIR80].

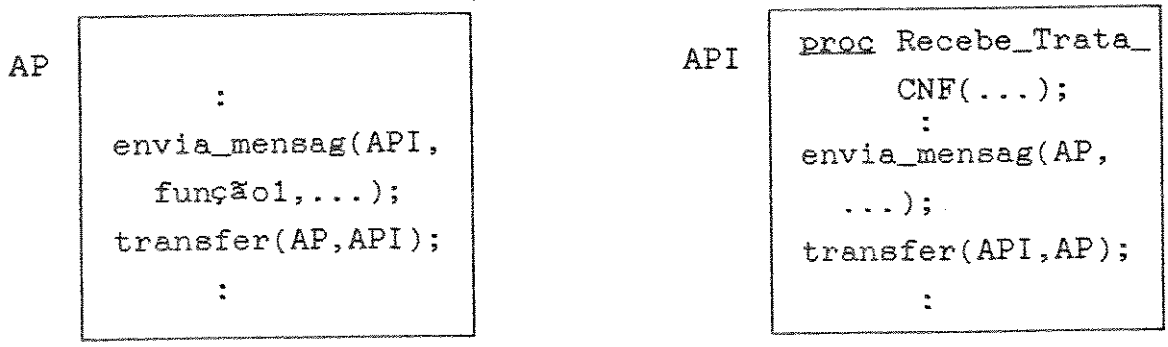
Em Modula-2 os processos são executados como corrotinas. O controle é transferido explicitamente de um processo para outro. A transferência de processos é feita através dos comandos TRANSFER(p1, p2) e IOTRANSFER(p1, p2, a), onde: p1 e p2 são variáveis do tipo processo e a é o tipo de uma interrupção.

Comparando este mecanismo de suporte para processos concorrentes com os anteriormente vistos, constata-se que Modula-2 resolve o problema de transferência entre processos de uma maneira mais primitiva. Em [WIR80], Wirth argumenta que as ferramentas fornecidas por Modula-2 são suficientes para a implementação eficiente de mecanismos de comunicação / sincronização mais elaborados, tais como monitores e troca de mensagens.

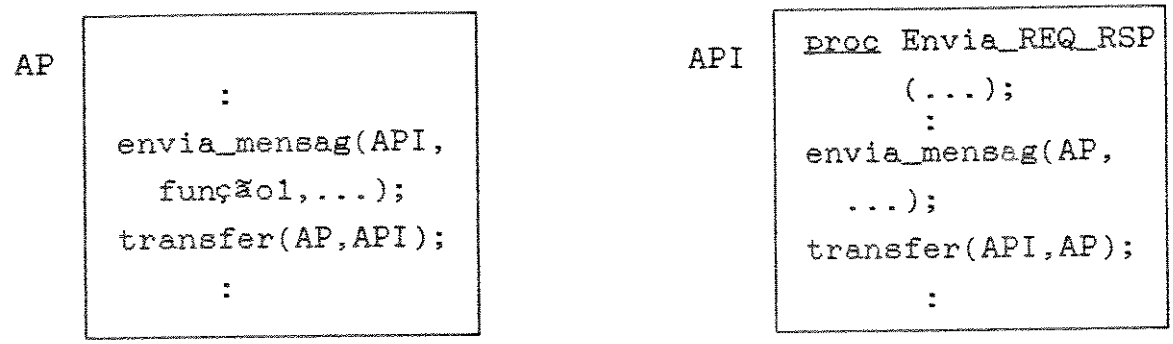
O AP, para executar uma função da API de modo síncrono, passa os parâmetros para a API numa área global do sistema, e transfere o controle para o processo API. O processo API, quando tiver a resposta do serviço solicitado, após a recepção e o tratamento das primitivas de confirmação associadas à função, passa este resultado para o AP numa área global do sistema, e transfere o controle para o AP (Figura 2.5-a).

O AP, para executar uma função da API de modo assíncrono, procede da mesma maneira do que para o caso anterior, assim como a API, com a exceção de que a API envia um resultado para o AP, quando tiver enviado as primitivas de requisição e resposta necessárias, e não somente quando tiver recebido as primitivas de confirmação associadas (Figura 2.5-b).

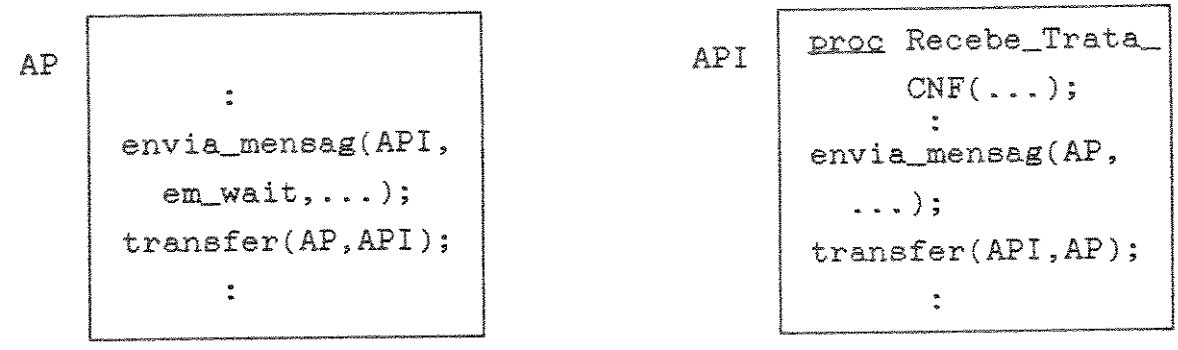
Quando o AP desejar executar a função em_wait da API, o AP passa os parâmetros para a API numa área global do sistema, e transfere o controle para o processo API. A API passa os parâmetros de saída para o AP, e transfere o controle ao mesmo, quando tiver recebido e tratado



(a) - função1 de modo síncrono



(b) - função1 de modo assíncrono



(c) - Função em_wait

Figura 2.5 - Transferência de Controle entre Corrotinas

as primitivas de confirmação correspondentes a uma função qualquer da

API ainda não respondida, que foi chamada de modo assíncrono (Figura 2.5-c).

A vantagem da utilização deste quarto tipo de sincronização é que um processo, ao terminar algum tratamento de informação, transfere imediatamente o controle ao processo que deve continuar o tratamento da informação.

2.7 - ESTRUTURAS DE DADOS

2.7.1 - ESTRUTURAS DE DADOS DAS INTERFACES DA API

Deverá existir uma estrutura de informação para a interface entre a API e o AP, e uma outra para a interface entre a API e a camada de aplicação.

Se a primeira interface for implementada através de chamadas de funções, a estrutura da informação é exatamente a estrutura dos parâmetros das funções definidas na especificação da API correspondente. Se for implementada através de troca de mensagens, a estrutura de informação, utilizando-se da sintaxe da linguagem C, pode ter a seguinte estrutura de mensagem:

```
typedef struct{
     Return_event_name          return_event_name;
     Return_code                return_code;
     Function_type              function;
     union
          {
          Param_funcao1         param_funcao1;
          Param_funcao2         param_funcao2;
                                :
          Param_funcaoN         param_funcaoN;
          } function_type_union;
     }Message_API_AP;
```

onde:

- return_event_name e return_code: são os campos da mensagem, que representam os parâmetros universais de mesmo nome das funções da API;

- function: é o campo, que define o tipo da função;

- param_funcao1, param_funcao2, ..., param_funcaoN: são os campos, que contêm os parâmetros das funções funcao1, funcao2, ..., funcaoN, respectivamente.

A interface entre a API e a camada de aplicação, se for implementada através de troca de mensagens, tem como estrutura de informação a seguinte estrutura de mensagem:

```
typedef struct{
     Application_entity        application_entity;
     Connection_id             connection_id;
     Primitive_type            primitive;
     Service_type              service;
     int                       data_area_lenght;
     union
          {
          ASE1_struct          ase1_struct;
          ASE2_struct          ase2_struct;
                               :
          ASEN_struct          aseN_struct;
          } data;
     } Message_API_Aplicacao;
```

onde o campo:

- application_entity: indica qual AE é endereçada. A figura 2.6 apresenta uma maneira de estruturar os processos (entidades), para a implementação em camadas de sistemas de comunicação baseados em troca de mensagens.

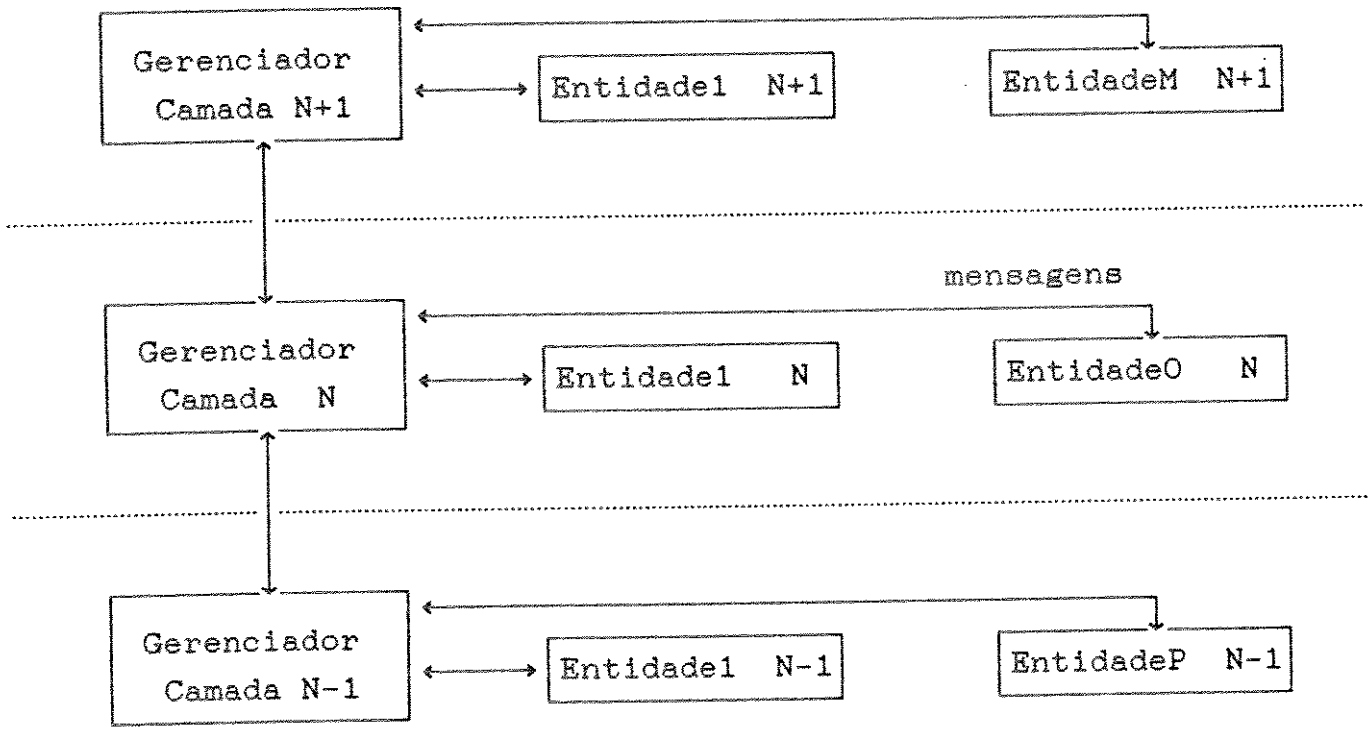


Figura 2.6 - Estrutura dos Processos das Diversas Camadas do RM OSI/ISO

Este campo é utilizado pelo gerenciador da camada de aplicação;

- connection_id: é o identificador da conexão;

- primitive: indica o tipo da primitiva. Os tipos possíveis são: requisição, indicação, resposta positiva, resposta negativa, confirmação positiva e confirmação negativa;

- service: indica o tipo de serviço. Os tipos possíveis são: confirmado do ASE1, não confirmado do ASE1, ..., confirmado do ASEN e não confirmado do ASEN;

- data_area_lenght: é o tamanho da área de dados, que é o campo seguinte da mensagem;

- data: é o campo de dados, que contém as estruturas de informação dos diversos ASEs. As estruturas de informação dos ASEs possuem dois campos:

a- nome do serviço;

b- parâmetros do serviço.

2.7.2 - ESTRUTURA INTERNA DE DADOS DA API

A estrutura de dados da API é composta de tabelas (vetores). São propostas as seguintes tabelas, onde entre parênteses é indicado o tipo do elemento da tabela:

a- Tabela de AE_LABELs (1 bit): Cada elemento indica se um AE_LABEL (identificador de invocação de Entidade de Aplicação) está ativo ou não. A tabela pode ser dividida em intervalos, onde cada um representa os AE_LABELs da mesma Entidade de Aplicação (Figura 2.7).

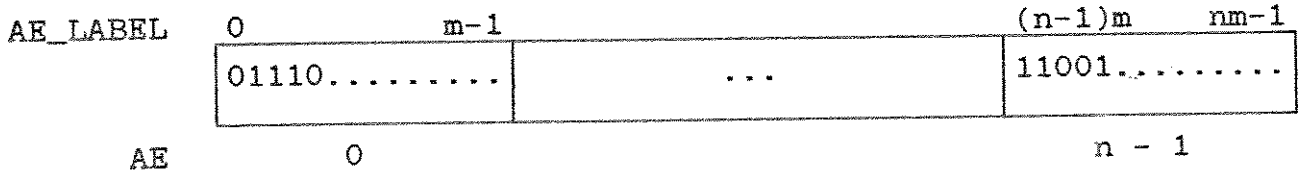


Figura 2.7 - Tabela de AE_LABELs

Tendo-se n Entidades de Aplicação e m Invocações por Entidade de Aplicação, a Tabela possui n x m elementos.

b- Tabela de Associações (CONNECTION_IDs) (2 bits): Devido ao fato de uma associação poder estar em um de quatro estados, os elementos da Tabela são representados por 2 bits. Cada elemento indica se uma associação está livre, estabelecida, pendente ou abortada.

Tendo-se l associações por Invocação de Entidade de Aplicação, a Tabela possui n x m x l elementos.

Duas soluções para a implementação desta Tabela são possíveis:

- Para o caso em que a API pode gerar um CONNECTION_ID a nível de interface no intervalo que a API desejar. Isto deve ocorrer tanto para a situação em que a API requisita o estabelecimento da associação, quanto para a situação em que a API remota requisita o estabelecimento da associação.

Neste caso, devido ao fato de cada associação pertencer a uma Invocação de AE (AE_LABEL), a Tabela pode ser dividida em

intervalos, onde cada um representa os CONNECTION_IDs de uma mesma Invocação de AE (Figura 2.8).

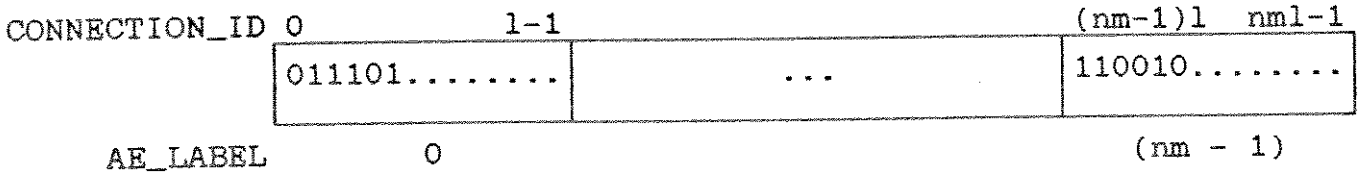


Figura 2.8 - Tabela de CONNECTION_IDs

- Para o caso em que a API pode gerar um CONNECTION_ID no intervalo que a API desejar, somente para a situação em que a API requisita o estabelecimento da associação.

Neste caso, a Tabela também é construída com **n x m x l** elementos. Contudo, a Tabela não é dividida em intervalos associados às invocações das AEs. Desta forma, precisa-se de uma Tabela auxiliar para relacionar as associações (CONNECTION_IDs) com as Invocações das AEs (AE_LABELs).

Uma solução para a Tabela auxiliar é construí-la também com **n x m x l** elementos (número total de associações). Neste caso, cada elemento da Tabela auxiliar possuirá $\lceil \log_2 n.m \rceil$ bits, ou seja, o número de bits necessários para representar as **n x m** Invocações de AE distintas.

c- Tabela de Serviços Pendentes (1 bit): Cada elemento indica se um serviço solicitado está pendente ou não.

Como cada serviço pendente é realizado numa associação, a Tabela pode ser dividida em intervalos, onde cada um representa os serviços pendentes de uma mesma associação (Figura 2.9). O número do serviço pendente é conhecido no Protocolo MMS como INVOKE_ID.

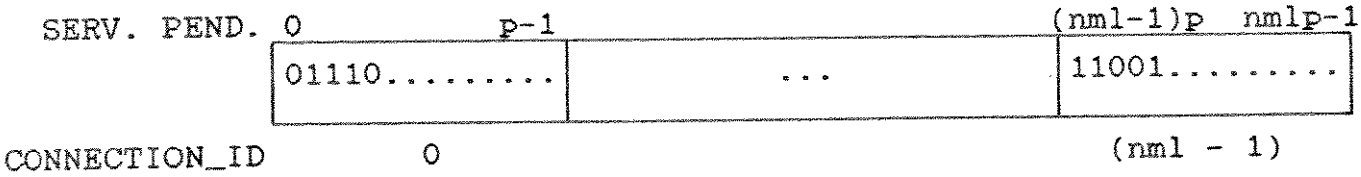
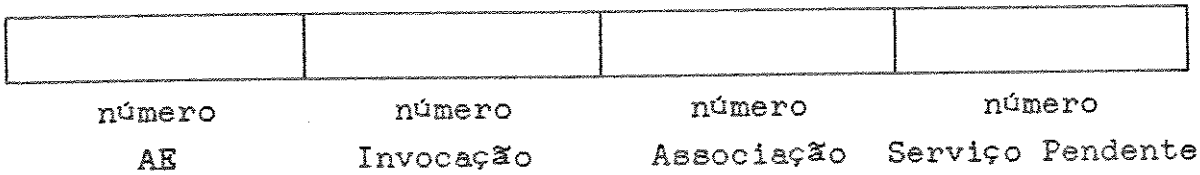


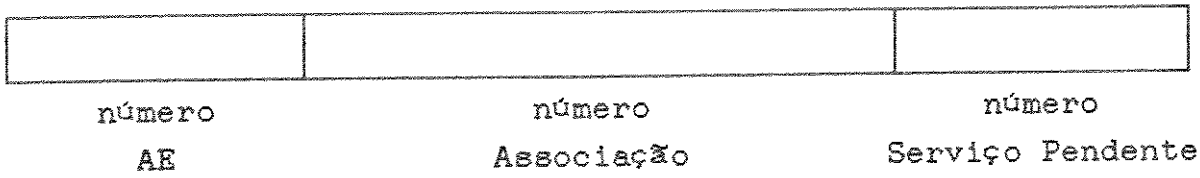
Figura 2.9 - Tabela de Serviços Pendentes

Tendo-se p serviços pendentes por associação, a Tabela possui n x m x l x p elementos.

Pode-se construir as Tabelas anteriores de maneira a dividi-las em intervalos como foi sugerido (Figuras de 2.7 a 2.9). Neste caso (I), o número de um serviço pode ser visto como a concatenação do número da AE, com o AE_LABEL da Invocação nesta AE, com o CONNECTION_ID da Associação nesta Invocação, e com o número do INVOKE_ID (seviço pendente) nesta Associação (Figura 2.10-a).



(a) - Caso I



(b) - Caso II

Figura 2.10 - Número do Serviço

Outra possibilidade, como foi visto anteriormente, é construir apenas as Tabelas de AE_LABELs e Serviços Pendentes de maneira a

dividi-las em intervalos como foi sugerido (Figuras 2.7 e 2.9). Neste caso (II), o número de um serviço pode ser visto como a concatenação do número da AE, com o CONNECTION_ID da Associação nesta AE, e com o número do INVOKE_ID (serviço pendente) nesta Associação (Figura 2.10-b).

d- Tabela de RETURN_EVENT_NAMEs (1 bit): Cada elemento indica se um RETURN_EVENT_NAME está sendo utilizado ou não. A Tabela tem n x m x l x p elementos;

e- Tabela de LISTENs (1 bit): Cada elemento indica se uma Invocação de AE está esperando ou não por primitivas de indicação de associação. Como o AP pode chamar a função LISTEN l vêzes (número máximo de associações por Invocação de AE) em cada Invocação de AE, a Tabela possui n x m x l elementos;

f- Tabela de INDICATION_RECEIVEs (1 bit): Cada elemento indica se a API foi ou não chamada através da função INDICATION_RECEIVE. Se o AP pode chamar a função INDICATION_RECEIVE r vêzes em cada associação, a Tabela possui n x m x l x r elementos. Para a definição do valor r, deve-se levar em consideração o número de primitivas de indicação diferentes, que podem ser recebidas nas diversas associações.

CAPÍTULO 3

EXEMPLO DE INTERFACE DE APLICAÇÃO PARA OS PROTOCOLOS MMS E ACSE: SISDI-MAP

3.1 - SISDI-MAP

O SISDI-MAP é o Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP, desenvolvido pelo Grupo de Redes do Departamento de Computação e Automação Industrial da Faculdade de Engenharia Elétrica da Unicamp. Como o MMS necessita do protocolo ACSE para o controle das associações, no SISDI-MAP a camada de aplicação é implementada possuindo uma AE, que contém os protocolos MMS e ACSE. O SISDI-MAP possui também a Interface de Aplicação correspondente.

Os objetivos do SISDI-MAP são o de obter experiência de implementação de protocolos, e o de ter um sistema didático que pode ser utilizado em aulas práticas sobre protocolos para comunicação fabril. O SISDI-MAP é comentado em [PAG89].

O modelo conceitual do SISDI-MAP é apresentado na figura 3.1. O SISDI-MAP é um sistema multitarefa composto pelos seguintes processos:

a - Interface de Operação de Usuário: permite ao usuário do SISDI-MAP criar uma chamada de função da API, utilizando um AP para este fim (opção A), ou executar um AP já existente (opção B). Além disso, fornece ao usuário informações sobre as mensagens enviadas/recebidas para/da rede, quando tratadas nos diversos processos: API, MMS, ACSE e Apresentação. Finalmente, permite que o usuário introduza erros nas mensagens quando da simulação de tráfego das mesmas na rede através do processo de apresentação;

b - Processos Aplicativos: representam os processos reais com a função de solicitar os serviços oferecidos pelo MMS no caso da opção B da Interface de Operação de Usuário. Representam os processos criados pela Interface de Operação de Usuário para conterem serviços isolados solicitados pelo usuário do SISDI-MAP, com a finalidade didática de

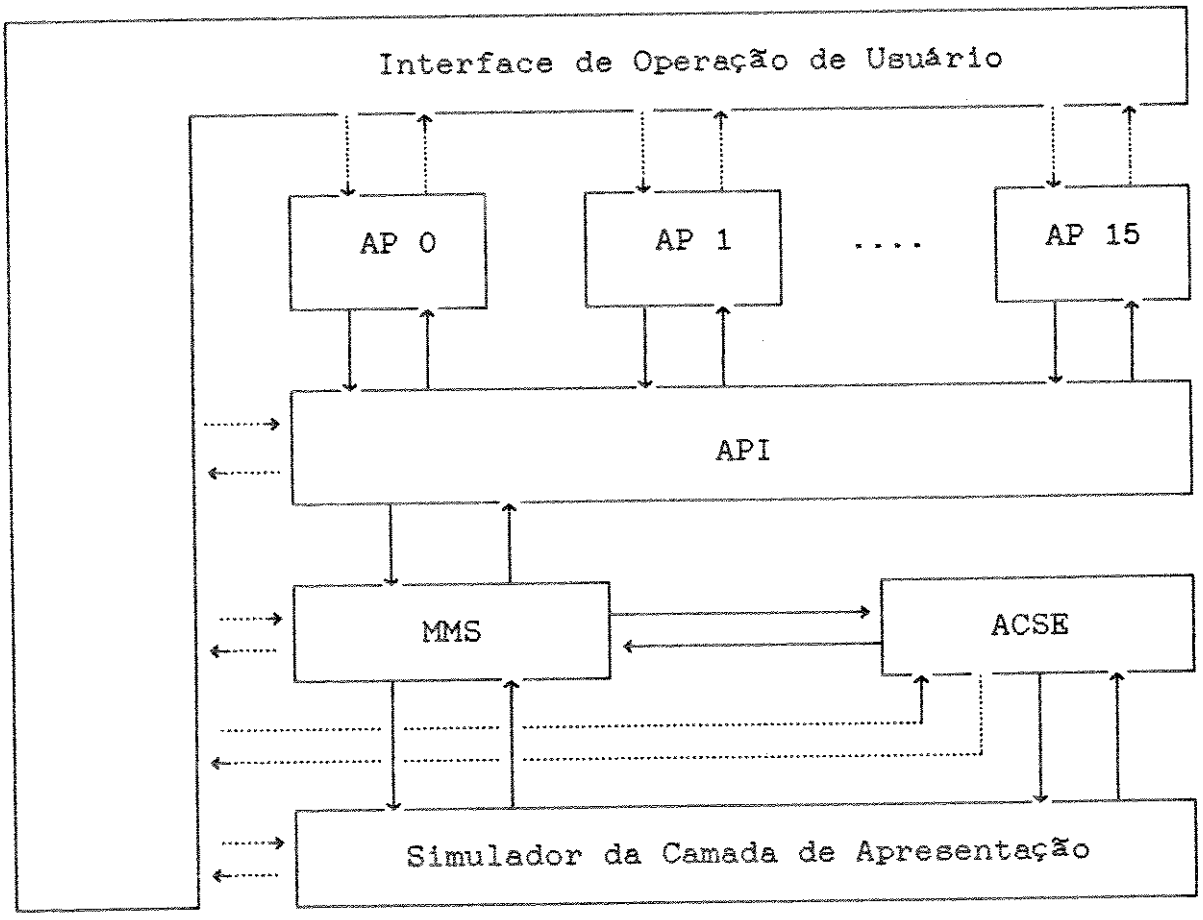


Figura 3.1 - SISDI-MAP

acompanhar o envio de uma mensagem MMS através do SISDI-MAP no caso da opção A da Interface de Operação de Usuário;

c - API: executa as funções da Interface de Aplicação para o MMS;

d - Protocolo MMS: executa as funções do provedor MMS;

e - Protocolo ACSE: executa as funções do provedor ACSE;

f - Simulador da Camada de Apresentação: permite a simulação das primitivas de serviço da camada de apresentação, e das demais camadas inferiores do modelo OSI/ISO. Desta maneira, simula-se a comunicação entre nós distintos da rede. Além disso, o usuário do SISDI-MAP, através da Interface de Operação de Usuário, pode modificar as primitivas nesta camada, simulando erros que poderiam ocorrer na transmissão pela rede. Desta forma, pode-se observar o comportamento das máquinas de protocolo MMS e ACSE e da Interface de Aplicação perante estes erros.

A implementação da Interface de Operação de Usuário faz parte da dissertação de mestrado [LIM]. Um exemplo de APs para controle de uma célula flexível da manufatura é apresentado em [COR90a]. A implementação do Protocolo MMS e do Simulador da Camada de Apresentação é o conteúdo das dissertações [JAC90] e [FER], e a implementação do Protocolo ACSE da dissertação [PAG91]. A implementação da API é a parte prática desta tese de doutorado e das dissertações de mestrado [COR90] e [SOU90].

O SISDI-MAP foi implementado para microcomputadores compatíveis com o PC, e é executado num ambiente DOS com a adição de um núcleo de tempo real. Este núcleo fornece funções apropriadas para comunicação e necessárias para a existência de um ambiente multitarefa, tais como: criação de tarefas, envio e recepção de mensagens, alocação de memória, criação de portas, definição e manipulação de semáforos, entre outras. A implementação do núcleo de tempo real é apresentado na dissertação [ZAB89].

Na figura 3.1 as setas cheias indicam o tráfego de mensagens entre os diversos protocolos, interface de aplicação e APs, que existiria se o SISDI-MAP não fosse um sistema didático, e sim um sistema real de comunicação. As setas tracejadas indicam o tráfego de informação para o usuário do SISDI-MAP:

a - Acompanhar o processamento de serviços solicitados aos protocolos e à interface de aplicação, e o envio e a recepção de mensagens entre si;

b - Acompanhar a simulação do tráfego de mensagens pela rede através do Simulador da Camada de Apresentação, e introduzir erros nestas mensagens;

c - Criar chamadas de funções à API, e obter as respostas correspondentes (opção A);

d - Ativar APs já existentes, acompanhando as suas execuções no tocante aos serviços de comunicação (opção B).

No SISDI-MAP a comunicação entre processos é baseada em troca de mensagens, utilizando portas providas pelo núcleo de tempo real. As portas fornecidas pelo núcleo funcionam como "mailboxes" para as/das quais todos os processos podem enviar / receber mensagens. Entretanto, no SISDI-MAP as portas estão associadas de forma estática aos

processos, ou seja, uma porta específica só recebe mensagens de um único processo, e só tem estas mensagens retiradas por um único outro processo. Isto é feito para simplificar o modelo de implementação do SISDI-MAP.

Quando da iniciação do SISDI-MAP, a primeira tarefa a ser chamada é a Interface de Operação de Usuário. Esta tarefa cria todas as portas e semáforos a serem utilizados por todas as tarefas, e depois cria as tarefas API, MMS, ACSE e Simulador da Camada de Apresentação, passando como parâmetros as portas e semáforos a serem utilizados pelas mesmas.

As portas dos processos MMS e ACSE, utilizadas pelo processo Simulador da Camada de Aplicação, e as portas deste Simulador, utilizadas pelo MMS e ACSE, constituem o P-SAP (Ponto de Acesso de Serviço da Camada de Apresentação). Para cada associação de aplicação, que utiliza o P-SAP, existirá um CEP (Ponto Final da Conexão) interno ao P-SAP.

O Processo Interface de Operação de Usuário é mais prioritário do que os outros processos, que têm a mesma prioridade. Isto é feito para que o usuário do SISDI-MAP possa ter informações mais rapidamente sobre as mensagens que trafegam pelo sistema. A partir destas informações, o usuário pode introduzir erros nas mensagens, cujas transmissões pela rede são simuladas, e verificar as consequências destes erros.

No SISDI-MAP existe apenas uma única instância de cada tarefa: API, MMS, ACSE e Simulador da Camada de Aplicação, inclusive no processamento de dois ou mais nós da rede, em qualquer momento de execução. Desta forma, diminui-se a memória requerida para a execução do SISDI-MAP, hoje instalado numa única estação. Este fato foi obtido, implementando-se as tabelas necessárias para cada tarefa indexada pelo número da conexão. O número da conexão é a concatenação do identificador da conexão no nó, com o identificador do nó ao qual a conexão pertence.

O SISDI-MAP apresenta valores máximos para alguns parâmetros:

Número de Nós: 4

Número de AEs por Nó: 1

Número de AEs: 4

Número de APs por Nó: 4

Número de APs: 16
Número de Invocações por AE: 4
Número de Conexões por Invocação de AE: 8
Número de Serviços Pendentes por Conexão: 16
Tamanho de Mensagem: 120 bytes

O Processo API possui 17 portas: 16 para receber mensagens dos 16 APs e uma para receber mensagens do MMS. O Processo API envia mensagens para 18 portas pertencentes aos 16 APs, MMS e Interface de Operação de Usuário (mensagens de controle). O Processo API também possui 16 semáforos relacionados com os 16 APs. A Interface de Operação de Usuário, quando cria o Processo API, passa como parâmetro as 35 portas e os 16 semáforos.

3.2 - API PARA O MMS: ASPECTOS DE IMPLEMENTAÇÃO

3.2.1 - ESPECIFICAÇÃO DO PROJETO MAP/TOP

A API para o MMS [MAP88e] apresenta serviços associados aos serviços definidos na parte 1 do MMS [ISO9506a]. Somente um subconjunto dos serviços MMS são tratados na especificação da API, e corresponde ao subconjunto de serviços MMS definidos no Projeto MAP na parte relativa à camada de aplicação.

[MAP88e] define a API para estações clientes que requisitam todos os serviços MMS, e recebem indicações de alguns serviços (e consequentemente, respondem), como por exemplo, dos serviços relacionados com as unidades funcionais de Acesso às Variáveis e Gerenciamento de Semáforos.

A API para o MMS utiliza os serviços de gerenciamento de contexto apresentados em [MAP88c], e define parâmetros específicos e/ou processamento específico, relacionados ao MMS, para as funções CONNECT, LISTEN, ANSWER, ABORT e INDICATION RECEIVE.

A Tabela 3-I apresenta as funções de gerenciamento de conexão utilizadas pela API do MMS, que não têm parâmetros específicos do MMS, e não realizam processamento específico relacionado com o MMS.

mm_aeactivation
mm_aedeactivation
mm_slisten
mm_aereset

Tabela 3-I - Funções de Gerenciamento de Contexto sem Parâmetros Específicos do MMS

A API não utiliza os serviços de gerenciamento de conexão RELEASE REQUEST e RELEASE RESPONSE para o término de conexão, e define funções próprias para esta finalidade: CONCLUDE e CONCLUDE RESPONSE.

Para que o usuário do MMS aja como servidor (no modelo cliente/servidor) de serviços de Acesso às Variáveis e Gerenciamento de Semáforos, ou faça referência aos objetos do MMS utilizando nomes, a API do MMS define serviços próprios de gerenciamento de conexão para tal: VMD ACTIVATE, VMD DEACTIVATION e VMD RESET.

A API também define as funções CANCEL e CANCEL RESPONSE para o gerenciamento da conexão, pois o Protocolo MMS oferece o serviço de cancelamento de serviços solicitados anteriormente.

A Tabela 3-II apresenta todos os serviços de gerenciamento de contexto projetados especificamente para a API do MMS.

mm_vactivation	VMD Activation
mm_vdeactivation	VMD Deactivation
mm_vreset	VMD Reset
mm_conclude	Conclude
mm_coresponse	Conclude Response
mm_cancel	Cancel
mm_caresponse	Cancel Response

Tabela 3-II - Funções de Gerenciamento de Contexto Específicas do MMS

A função INDICATION RECEIVE é também utilizada para receber primitivas de indicação relacionadas com serviços MMS. O parâmetro de saída INDICATION_NAME desta função determina que tipo de indicação foi recebida. A função INDICATION RECEIVE deve ser chamada tantas vêzes, quantas indicações são esperadas.

Um exemplo de utilização de INDICATION RECEIVE é o caso do AP desejar esperar pelo recebimento de indicação de REPORT INFORMATION. Neste caso, o nó mestre pode receber informação do estado da aplicação escrava residente num robô ou CNC, entre outros.

Contudo, a API pode ser preparada para responder automaticamente as primitivas de indicação simples. Neste caso, nenhuma interação explícita com o usuário é requerida. Estas primitivas devem estar relacionadas com as unidades funcionais de Acesso às Variáveis ou Gerenciamento de Semáforos.

A API para o MMS procura ser tão amigável ao usuário quanto possível, e é uma interface de alto nível, pois não existe uma correspondência um a um entre as funções desta interface e as primitivas de serviço do MMS. Por exemplo, a função FILE GET, que realiza uma transferência de arquivo, é de alto nível, pois invoca três primitivas de serviço MMS: FILE_OPEN.req, FILE_READ.req e FILE_CLOSE.req. A API para o MMS possui, além de funções de alto nível, funções de baixo nível (existem funções específicas para invocar cada uma das primitivas de serviço MMS) e respondedoras (por exemplo, a função INDICATION RECEIVE).

O Apêndice E apresenta as funções da API para cada uma das unidades funcionais do MMS: Suporte aos VMDs (Dispositivos Virtuais da Manufatura), Gerenciamento de Domínios, Gerenciamento de Invocação de Programa, Acesso às Variáveis, Gerenciamento de Semáforos, Comunicação com o Operador, Gerenciamento de Eventos, Gerenciamento de Jornal e Gerenciamento de Arquivos.

O Apêndice E apresenta também as funções auxiliares da API para o MMS que permitem a construção e a interpretação das estruturas (objetos) do MMS. Por exemplo, permitem a construção e a interpretação da estrutura "lista de variáveis", utilizada pelas funções de leitura e escrita de variáveis. Em particular, as funções auxiliares V_GET e V_PUT são utilizadas para atualizar e ler o valor de uma variável no

VMD, respectivamente.

A função auxiliar V_GET permite ao usuário transformar informação local em dado na representação MMS. Para tal, a função tem como parâmetros de entrada o endereço e a descrição da informação local. O resultado da função é a informação convertida para dado MMS, que pode ser enviado pela rede (variável de rede no VMD).

A função auxiliar V_PUT permite ao usuário converter e colocar um dado MMS recebido da rede (variável de rede no VMD) na sua forma local. O parâmetro de entrada da função contém o dado MMS que necessita ser transformado. O resultado da função é o dado MMS convertido para informação na representação local, e colocado num endereço local.

As funções V_GET e V_PUT podem ser chamadas pelos APs ou pela própria API.

A função auxiliar NEW VARIABLE ASSOCIATION é conceitualmente uma das mais importantes funções auxiliares relacionadas com a unidade funcional de Acesso às Variáveis, pois permite ao usuário criar um objeto chamado associação de variável. Uma associação de variável permite ao AP associar uma representação local a uma representação de rede para uma variável de rede. O parâmetro LOCAL_MAPPING da função NEW VARIABLE ASSOCIATION provê o endereço local e a especificação de tipo local da variável de rede em questão. O objeto associação de variável é utilizado por várias funções da API, tanto por funções auxiliares, quanto por funções associadas ao Protocolo MMS.

A chamada de uma função auxiliar não causa o envio ou recebimento de primitivas para/da camada de aplicação. A existência de funções auxiliares facilita bastante as tarefas do usuário, pois as estruturas do MMS a serem construídas e interpretadas podem ser complexas.

Uma análise mais detalhada das funções auxiliares associadas à unidade funcional de Acesso às Variáveis é apresentado em [SOU90].

Portanto, existem dois tipos de funções na Biblioteca da Interface de Aplicação para o MMS:

a - Funções Principais, que mapeiam serviços MMS;

b - Funções Auxiliares, que ajudam o usuário, sem no entanto mapearem serviços MMS.

Todas as funções principais são sensíveis ao contexto, e são

projetadas para serem chamadas de maneira síncrona ou assíncrona. O parâmetro RETURN_EVENT_NAME indica qual destes dois modos está sendo utilizado. Todas as funções auxiliares são livres de contexto, e podem ser chamadas somente sincronamente.

3.2.2 - API NO SISDI-MAP

3.2.2.1 - INTRODUÇÃO

A implementação da API no SISDI-MAP foi realizada baseando-se no Modelo Geral proposto de API, apresentado na figura 1.28. Desta forma, a API possui os blocos funcionais LIB, HLSP, CSP, IFA, IP, VDRDB, PSP, TAOCF e RAOCF. Esta implementação é comentada em [MAD90a], [MAD90b], [MAD90c] e [COR90b].

Como a camada de aplicação é composta pelos Protocolos MMS e ACSE, a API necessita dos blocos funcionais TAOCF e RAOCF para controlar as interações entre os dois componentes lógicos da API: API-MMS e API-ACSE.

Devido ao fato da API para o MMS oferecer serviços de alto nível e confirmados, a API implementa os blocos funcionais HLSP e CSP. A API, para se preparar para receber indicações, utiliza o bloco funcional IFA. Por exemplo, a API recebe indicações de estabelecimento de associação.

Para responder automaticamente às primitivas de indicação, a API utiliza os blocos funcionais IP e VDRDB. Por exemplo, a API pode responder automaticamente (ou seja, sem intervenção de APs, com exceção da intervenção inicial para delegar esta autoridade à API) a um pedido de leitura por parte do par remoto.

Nesta seção 3.2.2 comentam-se alguns algoritmos principais destes blocos funcionais utilizados no SISDI-MAP.

Esta implementação também é baseada no esquema global do Modelo Geral de API, apresentado na seção 2.2. Desta forma, pode-se obter uma implementação modular da API, de tal maneira que a API pode ser implementada com todas as funções da Biblioteca em alguns nós, e implementada com apenas alguns módulos de funções nos nós mais simples. O conceito de módulo foi definido também na seção 2.2.

Como metodologia de desenvolvimento de "software" para a construção da API para o SISDI-MAP foi utilizado o modelo "cascata" de ciclo de vida com algumas variações de fases, em relação às fases apresentadas na seção 2.1. Por exemplo, devido ao fato da API ser apenas uma parte do Projeto SISDI-MAP, e portanto, necessitar ser integrado ao Projeto, foi criada uma fase específica no modelo "cascata" para definir as interfaces da API com o MMS, os APs e a Interface de Operação de Usuário.

Seguindo a especificação do SISDI-MAP, a API é implementada como uma única tarefa, e como tendo uma única instância. Como foi visto na seção 3.1, esta única tarefa (instância) trata de todos os processamentos relativos à Interface de Aplicação nos diferentes nós simulados.

Isto é possível, implementando-se as tabelas que representam as estruturas de dados da API, indexadas pelos AE_LABELs ou CONNECTION_IDs. Os AE_LABELs (e os CONNECTION_IDs) são representados como a concatenação do número do nó com o número dos AE_LABELs (e dos CONNECTION_IDs) no nó.

A definição de uma única instância para a API leva em consideração a simplicidade que procurou-se dar ao SISDI-MAP. Cada bloco funcional da API é um procedimento da tarefa API.

A implementação da API permite múltiplas invocações da AE, composta pelos Protocolos MMS e ACSE, por APs diferentes. O SISDI-MAP tem a restrição de que um AP pode ter apenas uma invocação da AE em qualquer instante da sua computação. A API do SISDI-MAP permite múltiplas associações para cada invocação.

A comunicação entre as tarefas API e MMS é realizada através de troca de mensagens, conforme a especificação do SISDI-MAP. O mesmo ocorre para a comunicação entre as tarefas API e Interface de Operação de Usuário, mas o tráfego de mensagens é apenas no sentido da API para a Interface de Operação de Usuário.

Para a interface entre as diversas tarefas APs e a tarefa API, além da comunicação entre tarefas é necessário ter sincronização entre tarefas, devido à existência de serviços síncronos e assíncronos.

No SISDI-MAP, a sincronização entre um AP e a API é obtida através da utilização de semáforos. A comunicação entre os APs e a API

é realizada através de troca de mensagens. O núcleo de tempo real utilizado fornece primitivas de manipulação de semáforos e para troca de mensagens. Desta forma, para a interface AP-API, o SISDI-MAP utiliza os conceitos de sincronização através de semáforos apresentado na seção 2.6.2, e os conceitos de sincronização e comunicação através de troca de mensagens apresentados na seção 2.6.3.

A API, como todas as tarefas do SISDI-MAP, é executada sobre o sistema operacional DOS e o núcleo de tempo real comentado na seção 3.1. Como foi visto na seção 2.5, a troca do sistema operacional local implica na troca das funções WAIT e NOTE da API. Além disso, devem ser realizadas alterações nas partes da API que tratam do gerenciamento das tarefas e da troca de mensagens entre as diversas tarefas.

No SISDI-MAP, em que a API é implementada como uma única tarefa, as partes da API que tratam do gerenciamento de processos e troca de mensagens, pertencem às rotinas básicas do TAOCF, RAOCF e PSP. Estas partes consistem de chamadas ao núcleo de tempo real.

A seção 2.5 apresenta alguns comentários sobre a troca do sistema operacional utilizado, se o novo sistema operacional é o UNIX.

3.2.2.2 - FUNÇÕES IMPLEMENTADAS

A API do SISDI-MAP foi construída para um pequeno subconjunto de funções associadas aos Protocolos MMS e ACSE. As funções implementadas associadas ao Protocolo MMS pertencem às unidades funcionais de Gerenciamento de Contexto e Acesso às Variáveis. As funções da API podem ser classificadas em quatro grupos:

a - Funções de Gerenciamento de Contexto: permitem o gerenciamento dos contextos das aplicações, como por exemplo, a ativação e a desativação da Entidade de Aplicação, além do estabelecimento e liberação de associações (conexões);

b - Funções associadas ao Protocolo MMS: permitem a invocação de primitivas de serviço do MMS;

c - Funções Auxiliares: permitem a construção e a interpretação das estruturas de dados definidas pelo Protocolo MMS, como por exemplo, a estrutura que representa uma lista de variáveis;

d - Funções de Gerenciamento de Eventos: permitem chamadas

de funções da API de maneira assíncrona.

A Tabela 3-III apresenta as funções de Gerenciamento de Contexto implementadas no SISDI-MAP. Por primeiro, estas funções foram implementadas de acordo com a especificação de API relacionada com a versão 5.0 do MMS, que é a especificação [MAP87a]. Posteriormente, com o surgimento de [MAP88b], que é a especificação de API associada com a versão 6.0 do MMS, as funções de Gerenciamento de Contexto foram alteradas para que as mesmas pudessem ficar de acordo com a nova especificação [MAP88b]. As funções da Tabela 3.III foram comentadas nas seções 1.3.2 e 3.2.1.

As funções de Gerenciamento de Eventos implementadas são: EM_WAIT e EM_NOTE. A primeira é chamada por qualquer um dos APs, quando este deseja esperar pelo resultado de qualquer função chamada previamente pelo AP de maneira assíncrona. A segunda é chamada pela API, para notificar este AP de que um resultado esta disponível.

MM_AEACTIVATION
MM_AEDEACTIVATION
MM_CONNECT
MM_LISTEN
MM_SLISTEN
MM_ANSWER
MM_ABORT
MM_CONCLUDE
MM_CORESPONSE
MM_IRECEIVE

Tabela 3-III - Funções de Gerenciamento de Contexto

A implementação das funções de Gerenciamento de Contexto e de Gerenciamento de Eventos da API é em grande parte trabalho prático da dissertação de mestrado [COR90a], e em pequena parte trabalho prático desta tese.

A Tabela 3-IV apresenta as funções auxiliares implementadas no

SISDI-MAP. Estas funções foram implementadas de acordo com a primeira especificação de API para o MMS [MAP87a], associada com a versão 5.0 do MMS. As funções auxiliares implementadas estão relacionadas com a construção de estruturas do MMS, associadas com a leitura e escrita de variáveis. Algumas funções associadas ao Protocolo MMS e pertencentes à unidade funcional de Acesso às Variáveis foram implementadas, como por exemplo a função READ_VARIABLE para leitura remota de variáveis. Estas funções também foram implementadas de acordo com a especificação [MAP87a].

A implementação das funções auxiliares e das funções associadas ao Protocolo MMS é o trabalho prático da dissertação de mestrado [SOU90]. Foram feitas modificações nestas funções para torná-las compatíveis com o Protocolo MMS, que é implementado na versão 6.0 no SISDI-MAP.

MM_NRESTRICTION	NEW RESTRICTION
MM_NPACCESS	NEW PARTIAL ACCESS
MM_MLIST	MAKE LIST
MM_NCOMPONENT	NEW COMPONENT
MM_NTYPE	NEW TYPE
MM_NVSPECIFICATION	NEW VARIABLE SPECIFICATION
MM_NSACCESS	NEW SCATTERED ACCESS
MM_NVACCESS	NEW VARIABLE ACCESS
MM_NVASSOCIATION	NEW VARIABLE ASSOCIATION
MM_NDATA	NEW DATA

Tabela 3-IV - Funções Auxiliares

Como trabalho prático desta tese tem-se também a definição e a implementação das estruturas de dados e dos algoritmos básicos da API. A definição e a implementação destas estruturas de dados e algoritmos básicos possibilitam a integração de todas as funções da API: funções associadas com o Protocolo MMS e funções auxiliares (dissertação [SOU90]), além das funções de gerenciamento de contexto e eventos

(dissertação [COR90a]).

CHAMADAS SÍNCRONAS E ASSÍNCRONAS:

No SISDI-MAP, as funções da API associadas com o Protocolo MMS podem ser chamadas sincronamente ou assincronamente, enquanto que as funções auxiliares podem ser chamadas apenas sincronamente. As funções de Gerenciamento de Contexto podem ser chamadas também sincronamente ou assincronamente, mas algumas considerações devem ser feitas:

a - As funções MM_AEACTIVATION, MM_AEDEACTIVATION e MM_SLISTEN (STOP LISTEN) da API não enviam primitivas para a camada de aplicação. Portanto, a chamada síncrona ou assíncrona destas funções resulta no mesmo caso, pois não há primitivas de confirmação a se esperar;

b- As funções MM_ABORT, MM_ANSWER e MM_CORESPONSE (CONCLUDE RESPONSE) da API enviam primitivas para a camada de aplicação, mas estas primitivas não implicam na recepção de primitivas de confirmação. As funções MM_ANSWER e MM_CORESPONSE enviam primitivas de resposta para a camada de aplicação, e a função MM_ABORT envia uma primitiva de requisição, mas não é um serviço confirmado. Portanto, a chamada síncrona ou assíncrona destas funções resulta no mesmo;

c - As funções MM_LISTEN e MM_IRECEIVE (INDICATION RECEIVE) da API esperam por primitivas de indicação. Desta forma, se um AP chamar uma destas funções de maneira síncrona, o AP fica bloqueado até receber a primitiva de indicação desejada. Portanto, não é apropriado a chamada síncrona destas funções;

d - A função MM_CONCLUDE da API envia uma primitiva de requisição de término de associação, e espera pela primitiva de confirmação enviada pelo par remoto. Se o AP chamar a função sincronamente, o AP deve ter certeza que o par remoto responde apenas à primitiva de indicação de CONCLUDE, e não gera qualquer primitiva de requisição de CONCLUDE. Caso contrário, poderá ocorrer uma colisão de requisições de CONCLUDE, e devido ao fato do AP ter chamado sincronamente a função, o AP ficará bloqueado.

ESTAÇÕES CLIENTES E SERVIDORAS:

O SISDI-MAP simula vários nós da rede. Os nós da rede são estações clientes ou servidoras. Dependendo do tipo do nó, a especificação da API define quais os serviços são oferecidos aos APs, ou seja, quais os serviços de requisição e quais os serviços de resposta são oferecidos. Desta maneira, dependendo do tipo de nó, a API pode enviar para a camada de aplicação um subconjunto diferente das primitivas de requisição e resposta, e receber desta camada um subconjunto diferente das primitivas de indicação e confirmação.

O SISDI-MAP permite ao usuário do sistema definir o tipo (estação cliente ou servidora) de cada nó simulado, e verifica posteriormente se os serviços solicitados à API de um nó são permitidos em função do tipo deste nó.

3.2.2.3 - EXEMPLO DE PROCESSO DE APLICAÇÃO E USO DE FUNÇÕES AUXILIARES

A figura 3.2 apresenta um exemplo de um trecho de um Processo de Aplicação, escrito em C, que chama funções da API para utilizar os serviços do Protocolo MMS. Apenas algumas chamadas à API são mostradas, relativas à ativação e desativação da Entidade de Aplicação, estabelecimento e liberação de uma conexão, e solicitação de alguns serviços MMS: "status", "get name list", "get access attribute", além dos serviços de leitura e escrita remota de variáveis.

O exemplo inclui o uso de funções auxiliares para a construção de listas de associações de variáveis. O exemplo pretende mostrar também quais funções auxiliares da nova especificação da API para o MMS [MAP88b] devem ser utilizadas neste caso. A dissertação [SOU90] implementou justamente as funções auxiliares da especificação anterior da API para o MMS [MAP87a], para o caso em questão.

No exemplo, as seguintes funções auxiliares são utilizadas:

- MAKE LIST
- NEW ALTERNATE ACCESS
- NEW VARIABLE SPECIFICATION
- NEW VARIABLE ASSOCIATION e

- NEW DATA.

(a), (b), (x) e (z) são chamadas de funções de gerenciamento de contexto. (c), (d), (e), (f), (t) e (u) são chamadas de funções relacionadas com serviços do Protocolo MMS. (v) e (w) são chamadas de uma função de gerenciamento de eventos, e as chamadas de (g) a (s) são de funções auxiliares numa seqüência típica de uso.

A seguir comenta-se suscintamente cada uma das chamadas realizadas pelo AP à API.

(a) - O AP invoca a Entidade de Aplicação, chamada MY_AE_NAME, e a API retorna o identificador da invocação AE_LABEL;

(b) - O AP requer o estabelecimento de uma associação, através da invocação AE_LABEL, com a Entidade de Aplicação remota chamada YOUR_AE_NAME. Se a associação é estabelecida, a API retorna o identificador da associação CON_ID1. O AP pode solicitar o estabelecimento de outras associações, se desejar;

```
                    :
(a) mm_aeactivation( my_ae_name, ..., &ae_label );
                    :
(b) mm_connect( ae_label, ..., your_ae_label, ..., &con_id1 );
                    :
(c) mm_status( con_id1, ..., inout_status_dcb );
                    :
(d) mm_gnlist( con_id1, ..., inout_gnlist_dcb );
                    :
(e) mm_gaattribute( con_id1, ..., inout_gaattribute_dcb1 );
                    :
(f) mm_gaattribute( con_id1, ..., inout_gaattribute_dcb2 );
                    :
(g) mm_naaccess( ..., alt_access_dcb1, &alt_access_selection1, .
                 .. );
                    :
(h) mm_naaccess( ..., alt_access_dcb2, &alt_access_selection2, .
                 .. );
```

```
                    :
(i) mm_mlist( null_ptr, alt_access_selection1, ..., &alt_acc_lis
              t, ... );
                    :
(j) mm_mlist( alt_acc_list, alt_access_selection2, ..., &alt_acc
              _list, ... );
                    :
(k) mm_nvspecification( ..., var_dcb1, &new_var1, ... );
                    :
(l) mm_nvspecification( ..., var_dcb2, &new_var2, ... );
                    :
(m) mm_nvspecification( ..., var_dcb3, &new_var3, ... );
                    :
(n) mm_nvassociation( ..., var_asso_dcb1, &new_var_asso1, ... );
                    :
(o) mm_nvassociation( ..., var_asso_dcb2, &new_var_asso2, ... );
                    :
(p) mm_mlist( null_ptr, new_var_asso1, ..., &var_asso_list, ...
              );
                    :
(q) mm_mlist( var_asso_list, new_var_asso2, ..., &var_asso_list,
               ... );
                    :
(r) mm_ndata( kind_of_data, data_dcb, &new_data, ... );
                    :
(s) mm_nvassociation( ..., var_asso_dcb3, &new_var_asso3, ... );
                    :
(t) mm_read( con_id1, read_event, input_read_dcb, inout_read_dcb
              );
                    :
(u) mm_write( con_id1, write_event, input_write_dcb, inout_write
              _dcb );
                    :
(v) em_wait( wait_time1, &event1, ... );
```

```
                    :
(w) em_wait( wait_time2, &event2, ... );
                    :
(x) mm_conclude( con_id1, ... );
                    :
(z) mm_aedeactivation( aelabel, ... );
                    :
```

Figura 3.2 - Exemplo de Processo de Aplicação

(c) - O AP solicita o estado da estação remota. Como em todas as chamadas relativas aos serviços do Protocolo MMS, esta chamada possui como parâmetro de entrada pelo menos o identificador CON_ID1. O campo STATUS_INFORMATION do bloco INOUT_STATUS_DCB retorna a informação sobre o estado da estação remota;

(d) - O AP pede a lista de nomes das variáveis da estação remota, que são conhecidas a nível de rede. O campo LIST_OF_IDENTIFIERS do bloco INOUT_GNLIST_DCB retorna a lista requerida;

(e) e (f) - O AP solicita os atributos de variáveis da estação remota. O nome destas variáveis pode ter sido obtido a partir de (d). O campo GET_ACCESS_ATTRI_OUTPUT dos blocos INOUT_GAATTRIBUTE_DCB1 e INOUT_GAATTRIBUTE_DCB2 retorna os atributos;

De (g) a (s), o AP chama várias funções auxiliares da API com o objetivo final de construir duas listas de associações de variáveis. Uma lista é para a realização de uma operação de leitura remota de variáveis, enquanto que a outra é para a realização de uma operação de escrita remota de variáveis.

No exemplo, a lista de associações de variáveis para leitura contém dois elementos, ou seja, duas associações (ver (q)), e a lista de associações de variáveis para escrita contém um único elemento, ou seja, uma única associação (ver (s)). O nome das variáveis a serem lidas e escrita pode ter sido obtido a partir de (d). A especificação destas variáveis (como por exemplo, o tipo e o endereço) pode ter sido obtida através de (e) e (f).

(g) e (h) - O AP pede a construção de acessos alternados. Por exemplo, pode-se querer ler apenas os elementos de i a j e de k a l de uma variável do tipo "array". A função NEW ALTERNATE ACCESS constroi apenas o acesso alternado, independente da variável em que o acesso alternado será utilizado. No exemplo, o parâmetro ALT_ACCESS_DCB1 contém os índices i e j, enquanto que o parâmetro ALT_ACCESS_DCB2 contém os índices k e l. Os parâmetros ALT_ACCESS_SELECTION1 e ALT_ACCESS_SELECTION2 retornam os dois acessos alternados;

(i) e (j) - O AP requer a construção da lista de acessos alternados, utilizando os acessos alternados construídos em (g) e (h) nos parâmetros ALT_ACCESS_SELECTION1 e ALT_ACCESS_SELECTION2. O parâmetro ALT_ACCESS_LIST da função MAKE LIST retorna a lista. A lista de acessos alternados também é construída independente da variável em que a lista será utilizada (inclusive, podendo ser utilizada em mais de uma variável);

(k), (l) e (m) - O AP solicita a especificação das novas variáveis remotas a serem lidas ou escrita. Como parâmetro de entrada VAR_DCB da função NEW VARIABLE SPECIFICATION pode-se ter o nome da variável remota obtido em (d), o endereço e o tipo da variável remota obtidos em (e) ou (f) (ou apenas o endereço para alguns casos), ou a definição de um novo acesso disperso ("scattered access") a uma variável remota obtida em (d) ou (e) ou (f). Por exemplo, pode-se definir um acesso disperso a uma variável remota do tipo "array", como sendo apenas o acesso aos elementos de i a j e de k a l.

Na terceira opção de parâmetro de entrada (acesso disperso), este parâmetro pode ter duas origens:

- Ser o parâmetro de saída da função auxiliar NEW SCATTERED ACCESS. Este parâmetro de saída define um acesso disperso, por exemplo, a partir de um acesso alternado. O parâmetro de saída ALT_ACCESS_SELECTION da função NEW ALTERNATE ACCESS é um dos parâmetros de entrada da função NEW SCATTERED ACCESS;

- Ser o parâmetro de saída da função auxiliar MAKE LIST chamada para construir uma lista de acessos dispersos obtidos por chamadas da função NEW SCATTERED ACCESS.

Os parâmetros de saída NEW_VAR1, NEW_VAR2 e NEW_VAR3 em (k), (l) e (m) contêm as especificações das variáveis remotas a serem lidas ou

escrita no exemplo;

(n) e (o) - O AP requer a definição das associações de variáveis a serem lidas. A associação de variável liga a representação local à representação de rede de uma variável de rede. No campo VAR_SPEC dos parâmetros de entrada VAR_ASSO_DCB1 e VAR_ASSO_DCB2 colocam-se as especificações das variáveis remotas a serem lidas, por exemplo, os parâmetros NEW_VAR2 e NEW_VAR3 obtidos em (l) e (m).

Se NEW_VAR2 ou NEW_VAR3 contém uma variável remota com acesso disperso, na chamada de NEW VARIABLE ASSOCIATION, relativa a esta variável remota, pode-se passar como parâmetro de entrada um acesso alternado para a representação local da variável remota em questão. Por exemplo, este parâmetro de entrada pode ser o parâmetro de saída ALT_ACCESS_LIST obtido em (j).

Os parâmetros de saída NEW_VAR_ASSO1 e NEW_VAR_ASSO2 contêm as associações de variáveis relativas à NEW_VAR2 e NEW_VAR3, respectivamente;

(p) e (q) - O AP solicita a construção da lista de associações de variáveis, que serão lidas utilizando as associações de variáveis construídas em (n) e (o) nos parâmetros NEW_VAR_ASSO1 e NEW_VAR_ASSO2. O parâmetro VAR_ASSO_LIST da função MAKE LIST retorna a lista. A lista de associações de variáveis é construída independente de quantas operações de READ a utilizarão;

(r) - O AP pede a criação de um novo dado, cuja representação pode ser entendida pela API (representação MMS). Esta representação pode ser utilizada como parâmetro de entrada para a função NEW VARIABLE ASSOCIATION relacionada com uma variável remota que será escrita. Este novo dado é o que será escrito remotamente. Os parâmetros de entrada KIND_OF_DATA e DATA_DCB da função NEW DATA contêm informações sobre a representação e o endereço locais do dado. O parâmetro de saída NEW_DATA contém o dado na representação conhecida pela API (representação MMS);

(s) - O AP requer a definição da associação de variável a ser escrita. No campo VAR_SPEC do parâmetro de entrada VAR_ASSO_DCB3 tem-se a especificação da variável remota a ser escrita, por exemplo, o parâmetro NEW_VAR1 obtido em (k). No campo DATA_HANDLE de VAR_ASSO_DCB3 tem-se o dado a ser escrito remotamente, ou seja, o

parâmetro NEW_DATA obtido em (r). O parâmetro de saída NEW_VAR_ASSO3 contém a associação de variável desejada.

(t) - O AP solicita a leitura remota de uma lista de associações de variáveis através da conexão CON_ID1. O campo VAR_ASSOC_LIST do parâmetro INPUT_READ_DCB contém a lista de associações de variáveis, ou seja, o parâmetro VAR_ASSO_LIST obtido em (q).

A chamada da função READ é assíncrona, e o nome do evento associado à chamada é o parâmetro READ_EVENT. Devido ao fato da chamada ser assíncrona, o AP não fica bloqueado pela espera da resposta do pedido de leitura. Consequentemente, o AP pode continuar os seus processamentos específicos, após a API devolver o controle, significando que a requisição de leitura remota foi passada à camada de aplicação. Quando o AP desejar esperar pela resposta do serviço pedido deve chamar a função WAIT.

O campo LIST_OF_ACCESS_RESULT do parâmetro INOUT_READ_DCB contém os resultados da leitura das associações de variáveis;

(u) - O AP requer a escrita remota de uma associação de variável através da conexão CON_ID1. O campo VAR_ASSO_LIST do parâmetro INPUT_WRITE_DCB contém a associação de variável, ou seja, o parâmetro NEW_VAR_ASSO3 obtido em (s).

Neste caso, o dado a ser escrito remotamente foi construído pela função auxiliar NEW DATA, e é parte de NEW_VAR_ASSO3. Outras possibilidades são:

- O dado foi construído pela função V_GET e, portanto, também está na representação MMS;
- O dado está na sua forma local. Neste caso, a API deve executar a função V_GET para transformá-lo na representação MMS.

A chamada da função WRITE é assíncrona, e o nome do evento associado à chamada é o parâmetro WRITE_EVENT. Devido ao fato da chamada ser assíncrona, o AP não fica bloqueado pela espera da resposta do pedido de escrita. Portanto, o AP pode continuar os seus processamentos específicos, após a API devolver o controle. Quando o AP desejar esperar pela resposta de qualquer serviço solicitado assincronamente deve chamar a função WAIT.

O campo LIST_OF_WRITE_RESULT do parâmetro INOUT_WRITE_DCB contém o resultado da escrita da associação de variável;

(v) - O AP chama a função WAIT do gerenciamento de eventos da API, para esperar pelo término de um dos dois serviços solicitados anteriormente de maneira assíncrona (READ ou WRITE). O tempo de espera é o parâmetro de entrada WAIT_TIME1, e o serviço, que primeiro recebeu confirmação (resposta remota), é o associado ao parâmetro de saída EVENT1.

Se o serviço de READ foi completado primeiro, EVENT1 recebe o evento READ_EVENT. Se o serviço de WRITE foi completado primeiro, EVENT1 recebe o evento WRITE_EVENT. No exemplo, supõe-se que o tempo de espera WAIT_TIME1 é suficiente para se completar pelo menos um dos dois serviços;

(w) - O AP chama a função WAIT do gerenciamento de eventos da API, para esperar pelo término do outro serviço solicitado anteriormente de maneira assíncrona (READ ou WRITE). O tempo de espera é o parâmetro WAIT_TIME2 e, no exemplo, supõe-se que seja suficiente para se completar este segundo serviço.

Se o segundo serviço a se completar for o READ, o parâmetro EVENT2 recebe o evento READ_EVENT. Caso contrário, recebe o evento WRITE_EVENT;

(x) - O AP solicita a liberação (término) da associação CON_ID1;

(z) - O AP requer o término da invocação AE_LABEL.

USO DE FUNÇÕES AUXILIARES:

A utilização de funções auxiliares para manipulação de variáveis não é trivial, como pode ser visto no exemplo. Contudo alguns comentários devem ser feitos:

a - A manipulação de variáveis não é trivial no Protocolo MMS. Portanto, a API ajuda o usuário na construção e interpretação das variáveis. Se o sistema de comunicação não tem a API, o usuário é que deve construir e interpretar as variáveis;

b - No exemplo, procurou-se sempre casos genéricos. Simplificações podem ser realizadas. Por exemplo, usou-se como parâmetro de entrada da função READ uma lista de associações de variáveis, que para ser construída, o AP precisou chamar diversas funções auxiliares. Uma outra possibilidade de parâmetro de entrada da

função READ é uma lista de nomes de variáveis, que é muito mais fácil de ser construída. Um outro exemplo é que foi analisada a leitura remota de parte de uma variável do tipo "array". Para tal finalidade, o AP chamou algumas funções (NEW ALTERNATE ACCESS, MAKE LIST) que não necessitaria chamar, se este caso não existisse;

c - No exemplo, o AP chamou diversas funções auxiliares para posteriormente realizar uma única operação de READ e uma única operação de WRITE. Se outras operações de READ e WRITE fossem realizadas com os mesmos parâmetros de entrada, nenhuma das funções auxiliares anteriores teria que ser executada novamente. Geralmente, para repetir a execução de qualquer chamada de função, uma ou mais vêzes, com os mesmos parâmetros de entrada, nenhuma das funções auxiliares anteriores tem que ser executada novamente. Se os parâmetros de entrada devem ser outros, isto implica que parte das funções auxiliares anteriores deve ser executada outra vez.

3.2.2.4 - ALGORITMOS PRINCIPAIS

TREAT_AP:

O procedimento principal do TAOCF é o TREAT_AP, que trata das chamadas de funções da API pelos APs (Figura 3.3).

Os parâmetros do procedimento TREAT_AP são:

- O apontador para a estrutura da mensagem AP_API;
- A porta de saída para que a API envie as mensagens para o AP (porque a implementação descrita é baseada em troca de mensagens);
- O semáforo associado com o AP.

Dependendo do tipo de função, TREAT_AP chama um procedimento específico (api_aeact, api_aedeact, api_connect, api_read, etc). Se para o serviço requerido, a API não precisa esperar por uma primitiva de indicação ou confirmação enviada pela Máquina de Protocolo MMS, o valor TRUE é atribuído à variável FINISHED_SERVICE.

A função da API, chamada assincronamente, pode requerer algum recurso para as primitivas enviadas pelo par remoto como uma requisição ou resposta. Neste caso, TREAT_AP chama a função RETURN_RESOURCE_REQUEST_FUNCTION, ao invés de chamar o procedimento

específico. Esta função aloca os recursos necessários, e chama o procedimento específico, atualizando a variável FINISHED_SERVICE.

```
void                treat_AP(
  Block_api_ap          *pointer_interface_api_ap;
  struct t_queue        *output_port;
  struct t_queue        *semaphore)
{
Boolean                     finished_service;
      :
switch (pointer_interface_api_ap->function)
    {
    case mm_aeactivation :
       pointer_interface_api_ap->return_code = api_aeact(...);
       finished_service = TRUE;
       break;
    case mm_aedeactivation :
       pointer_interface_api_ap->return_code = api_aedeact(...);
       finished_service = TRUE;
       break;
            :
    case mm_connect :
    case mm_read :
            :
       return_resource_request_function(
            pointer_interface_api_ap);
       break;
    }
 if (pointer_interface_api_ap->return_code || finished_service)
    {
        :
    send_message(output_port, pointer_interface_api_ap);
    V(semaphore);
    }
```

```
    else
        if (mode == ASYNCHRONOUS)
            {
            send_message(output_port, pointer_interface_api_ap);
            V(semaphore);
            }
}
```

Figura 3.3 - Procedimento TREAT_AP

Depois do processamento do procedimento específico, se o serviço está terminado ou se houve algum erro, a API retorna ao AP. Para tal, a API envia a mensagem de resposta correspondente para a porta de saída, e chama a operação V (o parâmetro é o semáforo associado ao AP).

Se a função foi chamada assincronamente, a API retorna ao AP, informando que o serviço requerido, até agora, está correto, e que a API espera por uma primitiva de indicação ou confirmação da Máquina de Protocolo MMS.

TREAT_MMS:

O procedimento principal do RAOCF é o TREAT_MMS, que trata das primitivas recebidas da camada de aplicação, ou seja, da Máquina de Protocolo MMS (Figura 3.4).

O parâmetro do procedimento TREAT_MMS é o apontador do tipo endereço de uma estrutura da interface API_MMS (que é similar à estrutura da primitiva MMS), que aponta para a mensagem MMS recebida da Máquina de Protocolo MMS.

Dependendo do tipo de primitiva (indicação ou confirmação), uma função específica é chamada. O processamento da informação nestas funções resulta no processo de construção de uma mensagem do tipo da estrutura da interface API_AP. Esta mensagem é apontada por POINTER_INTERFACE_API_AP, e será enviada para o AP como resposta ao

serviço solicitado (função da API).

Depois, a API testa se o serviço foi solicitado sincronamente ou assincronamente pelo AP. Se a chamada do serviço foi síncrona, a API retorna ao AP:

- enviando a mensagem apontada por POINTER_INTERFACE_API_AP, através do procedimento SEND_END_AP, que utiliza o procedimento SEND_MESSAGE;
- chamando a operação V_LABEL (o parâmetro é o AE_LABEL), que notifica o AP.

```
void              treat_MMS(
  Block_api_mms              *pointer_interface_api_mms)
{
          :
switch (pointer_interface_api_mms->primitive)
    {
    case confirm_pos :
    case confirm_neg :
         pointer_interface_api_ap = treat_confirm(
            pointer_interface_api_mms);
         break;
    case indication :
         pointer_interface_api_ap = treat_indication(
            pointer_interface_api_mms);
         break;
    }
 if (pointer_interface_api_ap != NULL)
    {
         :
    if (mode == ASYNCHRONOUS)
       {
       if (test(wait_map, bit))
            {
            p_remove(...);
```

```
                send_end_AP(ae_label, pointer_interface_api_ap);
                V_label(ae_label);
                }
                else
                {
                pointer_queue = p_remove(...);
                w_insert(pointer_queue, pointer_interface_api_ap);
                }
        }
        else
        {
        send_end_AP(ae_label, pointer_interface_api_ap);
        V_label(ae_label);
        }
    }
}
```

Figura 3.4 - Procedimento TREAT_MMS

Se a função foi solicitada assincronamente pelo AP, é necessário verificar se o AP chamou ou não previamente a função WAIT. Se o AP chamou a função WAIT, a API remove os recursos alocados para esta função, e retorna ao AP:

- enviando a mensagem apontada por POINTER_INTERFACE_API_AP, através do procedimento SEND_END_AP;
- chamando a operação V_LABEL (o parâmetro é o AE_LABEL), que chama a função NOTE (o parâmetro é o RETURN_EVENT_NAME).

Se o AP não chamou a função WAIT, a API utiliza os recursos alocados para esta função para inserir a mensagem a ser enviada pela API para o AP na fila de serviços prontos. Quando o AP chamar a função WAIT, as mensagens serão removidas desta fila por ordem de chegada (ordem FIFO - First In First Out).

LIB, HLSP, CSP e PSP:

Os algoritmos básicos das unidades funcionais LIB, HLSP, CSP e PSP são apresentados em [MAP88b].

IFA, IP e VDRDB:

Os procedimentos básicos das unidades funcionais IFA, IP e VDRDB foram apresentados na seção 2.2.

3.2.2.5 - ESTRUTURAS DE DADOS

Nesta seção comenta-se a estrutura de dados básica da API no SISDI-MAP. A estrutura de dados é composta de:

a - Tipos de mensagens trocadas nas interfaces API-AP e API-MMS;

b - Tabelas que contêm informações sobre as invocações da AE, as associações de cada invocação, e os serviços requisitados nas diversas associações;

c - Filas internas que contêm mensagens trocadas nas interfaces API-AP e API-MMS;

d - Tabelas adicionais que contêm informações sobre os semáforos e as portas de comunicação associados aos APs.

MENSAGENS DAS INTERFACES API-AP E API-MMS:

A estrutura da mensagem utilizada na interface API-AP é a estrutura Message_API_AP apresentada na seção 2.7.1, que contém os campos RETURN_EVENT_NAME, RETURN_CODE, FUNCTION_TYPE e FUNCTION_TYPE_UNION. A estrutura da mensagem utilizada na interface API-MMS é a estrutura Message_API_Aplicacao apresentada também na seção 2.7.1, mas sem o campo APPLICATION_ENTITY, pois a camada de aplicação no SISDI-MAP possui apenas uma Entidade de Aplicação. Portanto, a estrutura da mensagem da interface API-MMS tem os campos CONNECTION_ID, PRIMITIVE, SERVICE, DATA_AREA_LENGHT e DATA.

TABELAS:

A API possui as tabelas apresentadas na seção 2.7.2 para armazenar informações sobre as invocações, associações e serviços:

- Tabela de AE_LABELs
- Tabela de CONNECTION_IDs
- Tabela de Serviços Pendentes
- Tabela de RETURN_EVENT_NAMEs
- Tabela de LISTENs
- Tabela de INDICATION_RECEIVEs.

Com relação à construção da Tabela de CONNECTION_IDs, optou-se pelo caso da API poder gerar o CONNECTION_ID no intervalo que a mesma desejar (opção a apresentada na seção 2.7.2), independentemente de se a requisição de estabelecimento de associação foi local ou remota.

De fato, cada associação no SISDI-MAP tem dois CONNECTION_IDs, um relacionado com a API (a API pode gerar o CONNECTION_ID no subintervalo desejado) e outro relacionado com o MMS (a API pode gerar o CONNECTION_ID no subintervalo desejado somente no caso da requisição de estabelecimento de associação ser local). Este fato implica na existência de duas tabelas na API, que relacionam os CONNECTION_IDs associados à API com os CONNECTION_IDs associados com o MMS e vice-versa.

Portanto, o número de um serviço no SISDI-MAP pode ser visto como a concatenação do número do nó (número da AE), com o número da invocação no nó, com o número da associação na invocação, e com o número do serviço pendente (INVOKE_ID) na associação.

Na seção 2.7.2, o número de vêzes que um AP pode chamar a função INDICATION RECEIVE em cada associação, sem que nenhuma destas chamadas tenha sido respondida, foi definido como r. No SISDI-MAP, o valor de r depende do:

a - número de primitivas de indicação diferentes que podem ser recebidas numa associação, numa estação cliente e numa estação servidora;

b - número de serviços pendentes suportados pela API numa associação, numa estação cliente e numa estação servidora.

Além das Tabelas citadas anteriormente, a API possui uma Tabela

de WAITs, onde cada elemento da tabela é do tipo bit. Cada elemento desta tabela indica se um AP está ou não em estado de espera, ou seja, se este AP chamou ou não a função WAIT da API para esperar por respostas de chamadas assíncronas anteriores.

Tendo-se n Entidades de Aplicação (n nós simulados no SISDI-MAP), m Invocações por Entidade de Aplicação, e uma Invocação por AP (restrição do SISDI-MAP), a Tabela de WAITs possui n x m elementos.

Todas as Tabelas da API são utilizadas pela maioria dos blocos funcionais do Modelo Geral de API.

FILAS INTERNAS:

A implementação da API possui as seguintes filas:

a - INTERFACE_API_AP: que contém mensagens (com informações parciais) do tipo da interface API-AP a serem retornadas aos APs como resposta às chamadas de funções da API. Quando a API obtém a resposta completa de um serviço solicitado, a API retira a mensagem correspondente desta fila, preenche o restante da mensagem de resposta com a informação obtida, e envia ao AP (ver também seção 3.2.2.6);

b - INTERFACE_API_MMS: que contém as primitivas enviadas pela API para a Máquina de Protocolo do MMS, as quais requisitam primitivas de confirmação, ou podem ser retransmitidas;

c - WAIT_QUEUE: que contém mensagens (totalmente preenchidas) do tipo da interface API-AP, que ainda não foram entregues aos APs. Este caso ocorre porque as funções relacionadas a estas mensagens foram chamadas assincronamente, e os APs não chamaram ainda a função WAIT.

TABELAS ADICIONAIS:

Devido ao fato da comunicação entre a API e os APs ser realizada através de troca de mensagens, e a sincronização entre a API e os APs ser realizada por semáforos, a API possui mais duas tabelas:

a - PORTA_AP_DESTINO: que associa cada AE_LABEL (identificador de invocação na API) com a porta de comunicação do AP correspondente;

b - SEMAFORO_AP_DESTINO: que associa cada AE_LABEL (identificador de invocação na API) com o semáforo do AP correspondente.

3.2.2.6 - ALOCAÇÃO DE MEMÓRIA

As mensagens trocadas entre a API e o AP e entre a API e o MMS apresentam estruturas diferentes. Alguns campos da mensagem API-AP não são transferidos para o MMS, pois têm significado apenas para a API, tais como o AE_LABEL, o RETURN_EVENT_NAME e o RETURN_CODE. O mesmo ocorre com alguns campos da mensagem API-MMS, que não são transferidos para o AP, pois não têm significado para o mesmo, como por exemplo, os campos VERSION_NUMBER_LIST e INVOKE_ID.

A diferença entre estes dois tipos de mensagens não é apenas questão de preâmbulo, ou seja, uma mensagem não é igual à outra a menos de certos campos iniciais e gerais, mas sim a diferença está na estrutura interna das mensagens. Os campos AE_LABEL e VERSION_NUMBER_LIST são exemplos citados que confirmam esta diferença na estrutura interna.

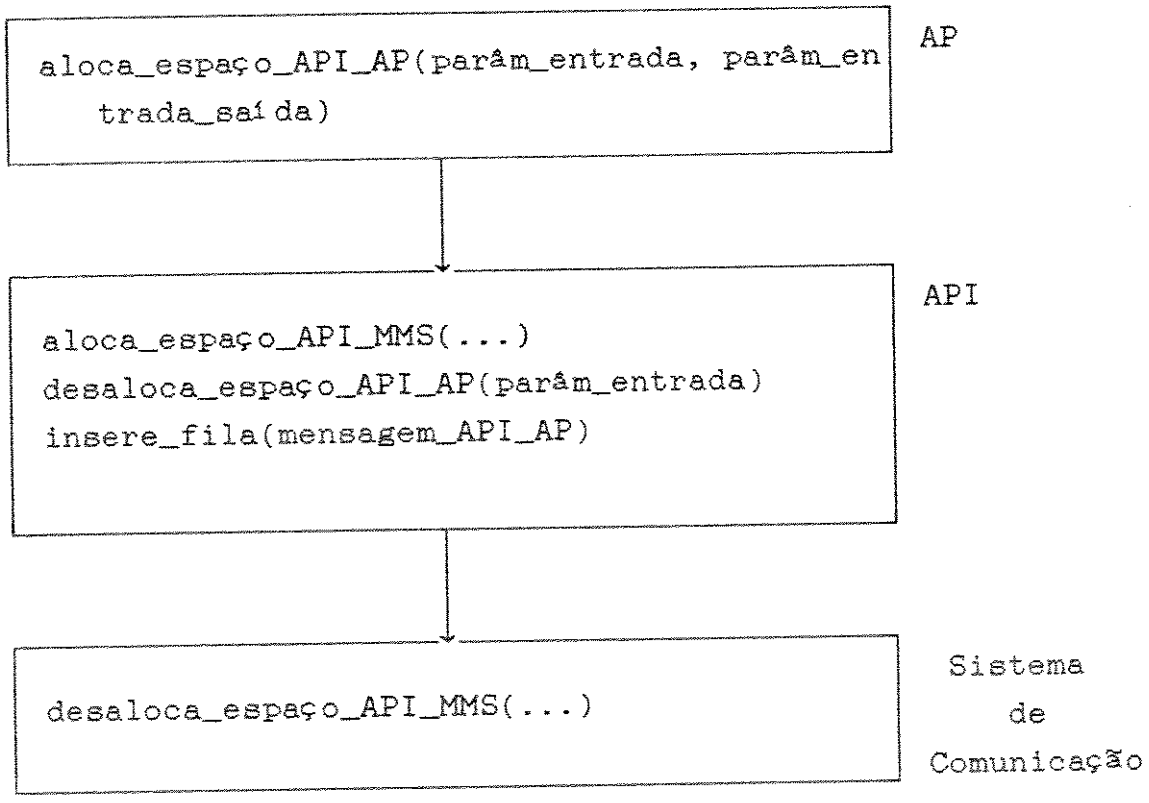
Desta forma, conclui-se que deve-se ter espaços de memória distintos para as mensagens API-AP e API-MMS. Quando o AP solicita um serviço da API através de uma mensagem API-AP, a API não pode aproveitar o espaço de memória desta mensagem para construir a mensagem API-MMS correspondente, que é uma primitiva de requisição ou resposta de serviço MMS.

Esta característica é oposta ao que ocorre normalmente, quando mensagens são transmitidas através das camadas do Modelo de Referência OSI/ISO [SV089]. Neste modelo, uma camada, para construir a sua PDU (Unidade de Dado do Protocolo), adiciona à sua SDU (Unidade de Dado do Serviço) recebida da camada superior (que corresponde à PDU desta camada) o seu preâmbulo num espaço já alocado pela camada mais superior do sistema de comunicação. A camada superior aloca espaço para os preâmbulos de todas as camadas inferiores, além de espaço para a sua informação.

Outra solução típica consiste na camada superior alocar espaço apenas para o seu preâmbulo, além de espaço para a sua informação.

Nesta solução, o sistema de comunicação constroi uma lista ligada de preâmbulos, à medida que a mensagem transita pelas camadas.

Outro fato que leva ao não reaproveitamento do espaço de memória da mensagem API-AP para a geração da mensagem API-MMS é a existência de serviços de alto nível na API. Isto implica que uma mensagem API-AP pode gerar diversas mensagens API-MMS.

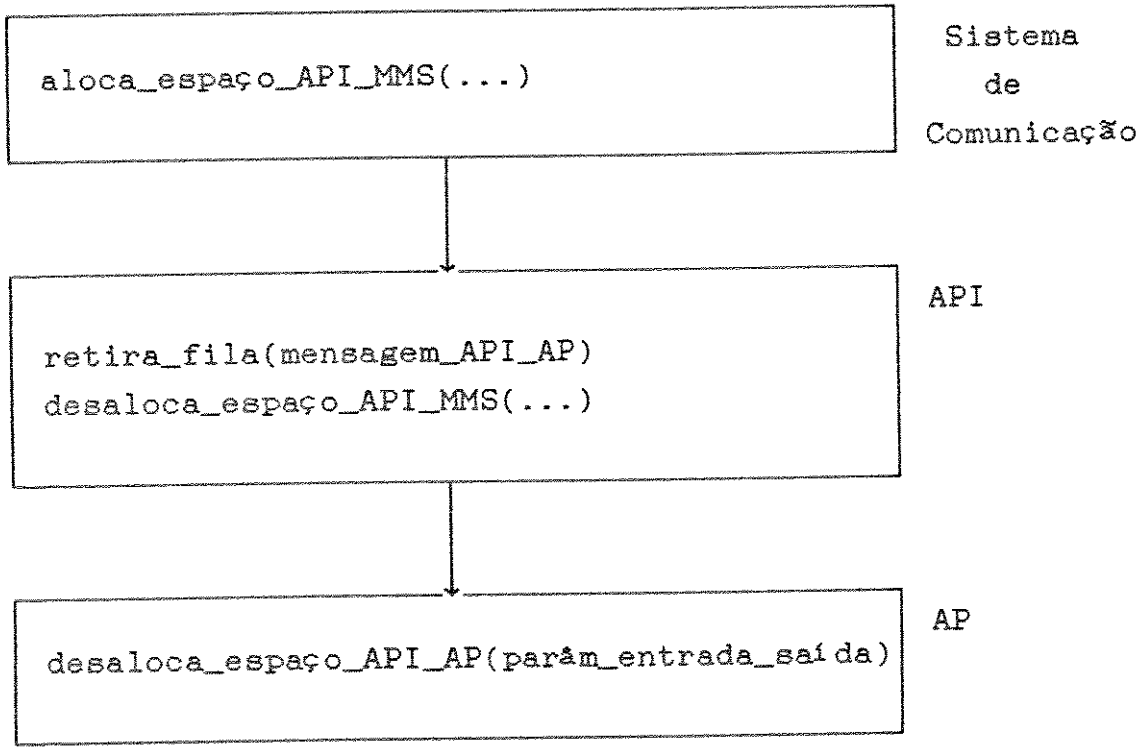


(a) - Mensagem no sentido AP -> Sistema de Comunicação

Figura 3.5 - Alocação de Memória

A figura 3.5 mostra o gerenciamento de memória por parte dos APs, da API e do Sistema de Comunicação, quando uma mensagem transita no sentido AP -> Sistema de Comunicação (a), e vice-versa (b).

Quando o AP deseja solicitar um serviço da API, o AP aloca memória para a mensagem API-AP que engloba os parâmetros de entrada e os parâmetros de entrada/saída. Para a mensagem API-AP não existem parâmetros só de saída pela própria especificação [MAP88b].



(b) - Mensagem no sentido Sistema de Comunicação -> AP

Figura 3.5 - Alocação de Memória (cont.)

Depois, a API aloca memória para a mensagem API-MMS de acordo com a implementação do sistema de comunicação. Duas opções são possíveis. A API aloca memória para todos os preâmbulos do sistema mais para a mensagem API-MMS, ou aloca memória somente para a mensagem API-MMS e o sistema de comunicação constroi uma lista ligada de preâmbulos. Para o SISDI-MAP, a alocação é realizada do primeiro modo. Se o serviço solicitado é de alto nível, a API pode alocar memória para diversas mensagens API-MMS.

Em seguida, a API mapeia os parâmetros da mensagem API-AP nos parâmetros das mensagens API-MMS. Se apenas uma mensagem API-MMS deve ser gerada, ou se todas as mensagens API-MMS podem ser geradas neste instante, a API pode liberar a memória alocada para os parâmetros de entrada da mensagem API-AP, e inserir a mensagem API-AP apenas com os

parâmetros de entrada/saída na fila INTERFACE_API_AP para esperar a resposta do par remoto. Se nem todas as mensagens API-MMS podem ser geradas neste instante, a API insere a mensagem API-AP inteira na fila INTERFACE_API_AP, pois os parâmetros de entrada poderão ser mapeados em parâmetros de mensagens API-MMS futuras. Quando a última mensagem API-MMS for gerada, então a API libera a memória alocada para os parâmetros de entrada da mensagem API-AP, e insere a mensagem API-AP com apenas os parâmetros de entrada/saída na fila INTERFACE_API_AP.

Uma camada inferior do sistema de comunicação libera a memória da mensagem API-MMS.

No sentido inverso, procedimentos similares são realizados. Uma camada inferior do sistema de comunicação aloca memória para a mensagem API-MMS.

Então, a API mapeia os parâmetros da mensagem API-MMS para os parâmetros de entrada/saída da mensagem API-AP, e libera a memória alocada para a mensagem API-MMS. Se apenas uma mensagem API-MMS deveria ser obtida como confirmação, ou se esta mensagem API-MMS recebida é a última que deveria ser obtida como confirmação de um serviço de alto nível solicitado anteriormente, a API retira a mensagem API-AP, que contém apenas os parâmetros de entrada/saída, da fila INTERFACE_API_AP.

O AP libera posteriormente a memória alocada para a mensagem API-AP que, como visto, contém apenas os parâmetros de entrada/saída.

No SISDI-MAP, a alocação e a liberação de memória são realizadas através de chamadas de primitivas do núcleo de tempo real.

CAPÍTULO 4

MODELO DE APRESENTAÇÃO: INTERFACE DE APLICAÇÃO x CONCEITOS DE "SOCKET", AGENTE DE USUARIO, "BINDING" E OPERAÇÃO REMOTA

4.1 - INTRODUÇÃO

Neste capítulo consideram-se algumas Interfaces de Aplicação embutidas em Protocolos de Aplicação definidos pela ISO, como por exemplo, o UA ("User Agent") do Protocolo X-400 ([CCITT400] e [ISO10021]) e o DUA ("Directory User Agent") do Protocolo DS ("Directory Service") [ISO9594]. Como foi visto no capítulo 1, o conceito de API é um conceito MAP/TOP, e não é utilizado pela ISO. Contudo, alguns conceitos da ISO, como por exemplo, o UE ("User Element") e o IF ("Interaction Function"), têm similaridades com o conceito de API. Nos exemplos dos protocolos X-400 e DS, analisa-se a similaridade do conceito de UA ("User Agent"), que estes protocolos possuem (conceito este, da ISO), com o conceito de API.

Neste capítulo também analisa-se o conceito de "socket" definido pelo UNIX de Berkeley [STE90], que suporta os Protocolos TCP/IP ("Transmission Control Protocol / Internet Protocol") ([POS81b] e [POS81a]). Será visto que o "socket" é uma API para a camada de transporte com características simples.

Na seção 4.4 comenta-se o caso de um sistema de comunicação com API, existindo na camada de aplicação o Protocolo ROSE ("Remote Operations Service Element") [ISO9072]. O que a API ganha em termos de facilidade com a existência do Protocolo ROSE será discutido.

No final do capítulo é proposto um Modelo de Apresentação de API, que reflete a tendência de uso de Interfaces de Aplicação nos sistemas de comunicação atuais.

4.2 - SISTEMA UNIX: CONCEITO DE "SOCKET"

Como foi visto, no capítulo 1, uma API depende do sistema operacional sendo utilizado e da linguagem de programação. Para o sistema operacional UNIX, existem duas APIs definidas para comunicação:

a- Berkeley Sockets: definido inicialmente no Sistema UNIX 4.1cBSD para o Vax [STE90];

b- TLI ("Transport Layer Interface"): introduzido inicialmente com a Versão 3.0 do "System V" [ATT89].

As duas APIs foram desenvolvidas para a linguagem C.

Uma associação de comunicação pode ser representada por uma quintupla:

<protocolo, endereço local, processo local, endereço remoto, processo remoto>,

onde:

a- protocolo: define o contexto da aplicação;

b- endereço local e endereço remoto: especificam os identificadores da rede e do nó, dos nós local e remoto, respectivamente;

c- processo local e processo remoto: identificam os processos específicos nos nós local e remoto, que estão envolvidos na conexão, respectivamente.

O "socket" é um ponto final de comunicação, ou seja, uma meia associação representada por <protocolo, endereço local, processo local> ou <protocolo, endereço remoto, processo remoto>, para a camada de transporte. Um par de "sockets" corresponde à definição de associação de transporte. O conceito de "socket" é similar ao conceito de T-SAP da OSI [IS07498].

Através de um "socket" local, um processo de aplicação local pode transferir dados, ou seja, enviar e receber mensagens para/de um "socket" remoto associado a um ou mais processos de aplicação remotos (Figura 4.1).

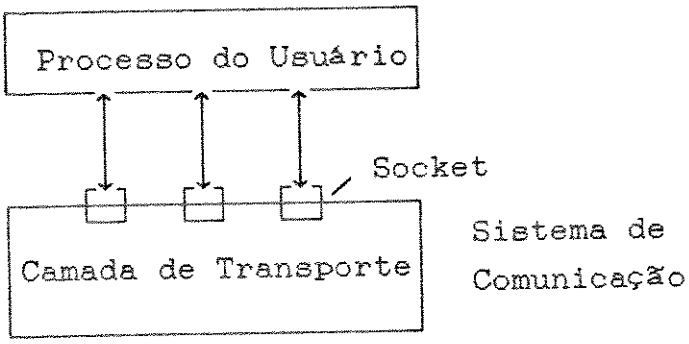


Figura 4.1 - Utilização de "Sockets"

O processo de aplicação local pode também passar o descritor do "socket" local para outro processo de aplicação local. Desta maneira, dois APs locais podem ser conectados a um único "socket".

A definição de um "socket" (uma estrutura de dados dentro do sistema operacional) implica na alocação de filas para envio e recepção de mensagens.

A Interface de "sockets" é baseada em chamadas ao sistema, ou seja, para criar um "socket", iniciar uma conexão ou realizar qualquer operação da Interface, o AP deve realizar uma chamada ao sistema. A Interface TLI é baseada em funções, ou seja, a Interface é uma biblioteca de funções que realizam a definição de um ponto final de comunicação, o estabelecimento de uma conexão e os demais serviços. Portanto, o AP, para realizar uma determinada operação, chama uma função da Interface TLI. A Interface deve ser ligada ao AP através do editor ligador.

A figura 4.2 apresenta a comparação dos serviços oferecidos entre "sockets" e TLI. As figuras de 4.2 a 4.6 são apresentadas (ou são variações) das apresentadas em [STE90].

		SOCKETS	TLI
Servidor	Alocar espaço		t_alloc()
	Criar ponto final de comunicação	socket()	t_open()
	Mapear endereço	bind()	t_bind()
	Especificar fila	listen()	
	Esperar por conexão	accept()	t_listen()
	Obter novo fd		t_open()
			t_bind()
			t_accept()
Cliente	Alocar espaço		t_alloc()
	Criar ponto final de comunicação	socket()	t_open()
	Mapear endereço	bind()	t_bind()
	Conectar ao servidor	connect()	t_connect()
	Transferir dado	read() write() recv() send()	read() write() t_rcv() t_snd()
	Datagramas	recvfrom() sendto()	t_rcvudata() t_sndudata()
	Terminar	close() shutdown()	t_close() t_sndrel() t_snddis()

Figura 4.2 - Comparação entre "Sockets" e TLI

Os "sockets" podem ser de dois tipos: orientados à conexão e sem conexão. A figura 4.3 apresenta as chamadas típicas ao sistema para o protocolo orientado à conexão, e a figura 4.4 para o protocolo sem conexão.

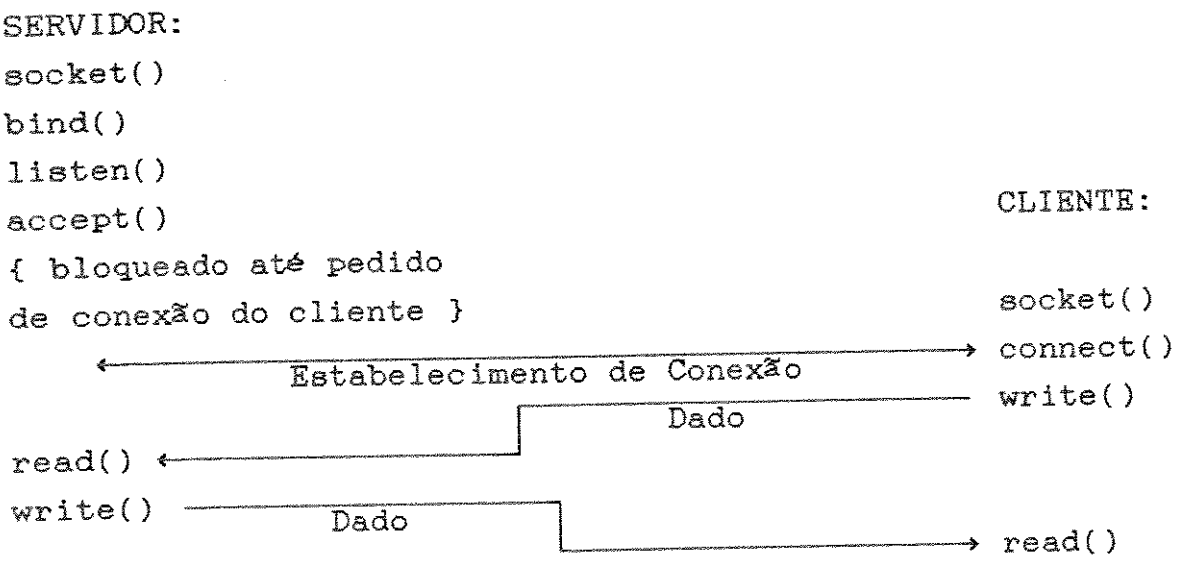


Figura 4.3 - Chamadas ao Sistema relacionadas com os "Sockets" para Protocolo Orientado à Conexão

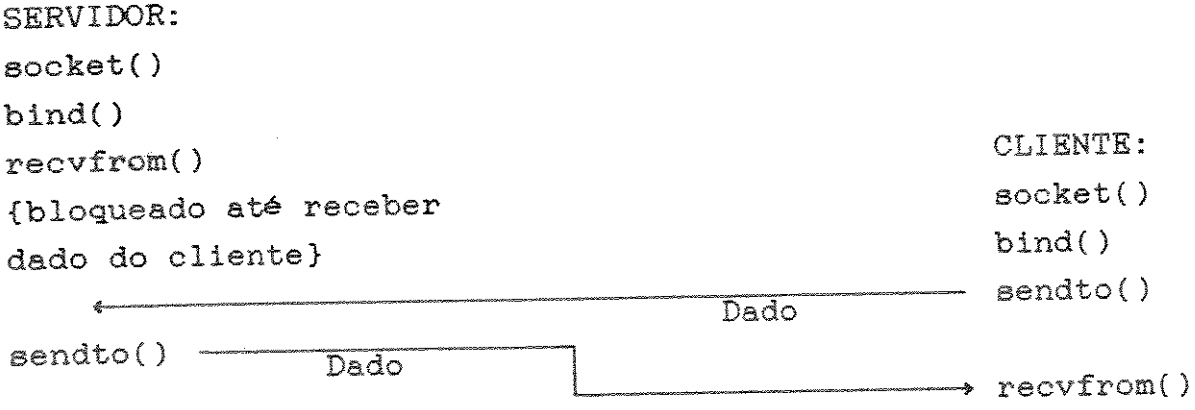


Figura 4.4 - Chamadas ao Sistema relacionadas com os "Sockets" para Protocolo sem Conexão

A figura 4.5 apresenta para cada parâmetro de uma associação (representada por uma quintupla) qual chamada do sistema o preenche, para os seguintes casos: estação servidora com protocolo orientado à conexão, estação cliente com protocolo orientado à conexão, estação servidora com protocolo sem conexão e estação cliente com protocolo

sem conexão.

	Protocolo	End e Proc Local	End e Proc Remoto
Serv/Orient Con	socket()	bind()	listen(), accept()
Cl/Orient Con	socket()	connect()	connect()
Serv/Sem Con	socket()	bind()	recvfrom()
Cl/Sem Con	socket()	bind()	sendto()

Figura 4.5 - Chamadas ao Sistema relacionadas com "Sockets" e Parâmetros da Associação

A figura 4.6 mostra a implementação de "sockets". Os "sockets" são implementados dentro do "kernel" do UNIX. Os protocolos de comunicação (TCP/UDP/IP) também pertencem ao "kernel" do UNIX.

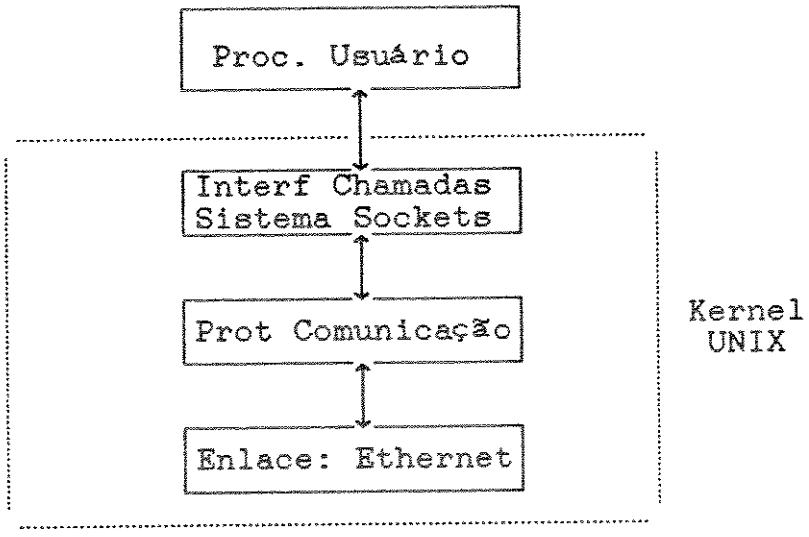


Figura 4.6 - Implementação de "Sockets"

O conceito de "sockets" possui vantagens e desvantagens em relação à API proposta no capítulo 1, as quais na realidade, refletem para os diferentes casos de aplicação, qual o conceito que é melhor ser utilizado.

VANTAGENS:

Um AP, ao utilizar um "socket", tem as seguintes vantagens:

a- Como o "socket" é uma interface para a camada de transporte, o sistema de comunicação não tem as camada de aplicação, apresentação e sessão. Isto simplifica o sistema de comunicação, apesar de perder os serviços oferecidos normalmente por estas camadas. Contudo, se o AP deseja apenas enviar e receber mensagens genéricas para/de um AP remoto, os serviços das camadas de aplicação, apresentação e sessão podem ser desnecessários;

b- O "socket" permite acesso direto à camada de transporte por parte do AP. O "socket" funciona como uma "porta" de entrada/saída ao "kernel" do UNIX, que engloba a camada de transporte. Não existe uma interface sofisticada entre o AP e o sistema de comunicação. Portanto, o sistema de comunicação é simplificado;

c- A manipulação de um "socket" é feita através de chamadas ao sistema. Isto implica que as estruturas relacionadas a um "socket" estão dentro do "kernel" do UNIX. Portanto, a manipulação de "sockets" é eficiente;

d- Um "socket" pode estar conectado a mais de um AP. Este fato implica numa maior flexibilidade das aplicações que utilizam o sistema de comunicação.

DESVANTAGENS:

Um AP, ao utilizar um "socket", ao invés da API proposta, tem as seguintes desvantagens:

a- O "socket" só pode ser utilizado num ambiente em que os sistemas local e remoto tenham como sistema operacional o UNIX. O uso da API proposta não tem esta restrição;

b- Através de um "socket", um AP pode enviar e receber mensagens genéricas. Se o AP deseja utilizar um protocolo de aplicação próprio ao seu tipo de processamento, o AP não pode utilizar os "sockets", pois estes implicam num sistema de comunicação sem a camada de aplicação. A mesma restrição ocorre se o AP deseja utilizar algum

serviço específico oferecido pela camada de apresentação ou sessão;

c- Para um sistema de comunicação baseado em "sockets", não existem os conceitos de:

- Serviço confirmado: Pode-se apenas ler e escrever mensagens de/em "sockets" locais/remotos. Se um AP deseja ter confirmação do serviço executado, é de responsabilidade dos APs envolvidos providenciarem a confirmação;

- Serviço de alto nível;

- Serviço respondedor: Não é possível preparar a API do tipo "socket" para decidir sobre a aceitação ou não de primitivas de indicação não solicitadas;

- Resposta automática: Não é possível programar a API do tipo "socket", para responder automaticamente a primitivas de indicação que chegam solicitando serviços;

- Evento associado a um pedido de serviço: Desta maneira, não é possível solicitar diversos serviços a um mesmo "socket" sem ficar bloqueado, e poder esperar posteriormente pela resposta de um dos diversos serviços pedidos. Contudo, a chamada ao sistema SELECT permite verificar, se num conjunto de "sockets" conectados ao AP, algum destes "sockets" pode ser lido ou escrito. Como exemplo, esta chamada ao sistema pode ser utilizada pelo AP que tem várias conexões estabelecidas, e espera receber mensagens em diversas destas conexões. Se o AP escolhesse um "socket" para ler, e este não tivesse mensagem disponível, o AP ficaria bloqueado, apesar de que outros "sockets" pudessem ter mensagens disponíveis. Se o AP chamar a operação SELECT, sobre um conjunto de "sockets", o AP fica bloqueado até receber mensagem em pelo menos um dos "sockets" selecionados;

d- Como um "socket" pode estar conectado a mais de um AP, é de responsabilidade dos APs envolvidos controlar o compartilhamento do recurso comum ("socket");

e- Devido ao fato dos protocolos de comunicação (TCP/UDP/IP) pertencerem ao "kernel" do UNIX, quando deseja-se trocar um ou mais dos protocolos de comunicação (por exemplo, por protocolos de comunicação do padrão OSI), é necessário mudar o "kernel" do UNIX.

CONCLUSÃO:

Portanto, com relação ao "socket", pode-se concluir que:

a- É uma API, pois o "socket" oferece uma interface ao AP, para acessar o sistema de comunicação;

b- É uma API para a camada de transporte, pois o "socket" oferece ao AP acesso à camada de transporte. Pode-se extrapolar o conceito de "socket" para outras camadas do sistema de comunicação. Por exemplo, para sistemas simples pode-se ter "sockets" para a camada de enlace. Neste caso, o sistema de comunicação teria apenas as camadas de enlace e física. É o caso correspondente para os "sockets" ao caso das estações mini-MAP no Projeto MAP [MAP87d]. Neste caso de estações simples com a existência de "sockets" no sistema de comunicação, a camada de enlace deve ser robusta;

c- É uma API simplificada. Como foi visto no ítem c das desvantagens dos "sockets" em relação à API proposta, o "socket" não possui a maioria dos conceitos da API proposta. Contudo, como foi visto nos ítens das vantagens, o "socket" é eficiente quando deseja-se apenas transferir dados, ou seja, enviar e receber mensagens genéricas para/de APs.

4.3 - PROTOCOLO X-400 E PROTOCOLO DE SERVIÇO DE DIRETÓRIO: CONCEITO DE AGENTE DE USUÁRIO

Os Protocolos X-400 ([CCITT400] e [ISO10021])] (para troca de mensagens eletrônicas) e de Serviço de Diretório [ISO9594] possuem nas suas estruturas um elemento chamado Agente de Usuário (UA - User Agent - no X-400 e DUA - Directory User Agent no Serviço de Diretório), que é o elemento ao qual o AP interage.

PROTOCOLO X-400:

No campo de sistemas com protocolos para troca de mensagens eletrônicas (MHS - Message Handling Systems), o CCITT e a ISO têm trabalhado conjuntamente de maneira a gerarem padrões com textos

similares. O padrão CCITT é o X-400, e o da ISO é o 10021 (MOTIS - Message Oriented Text Interchange Systems). A figura 4.7 apresenta o modelo funcional do MHS.

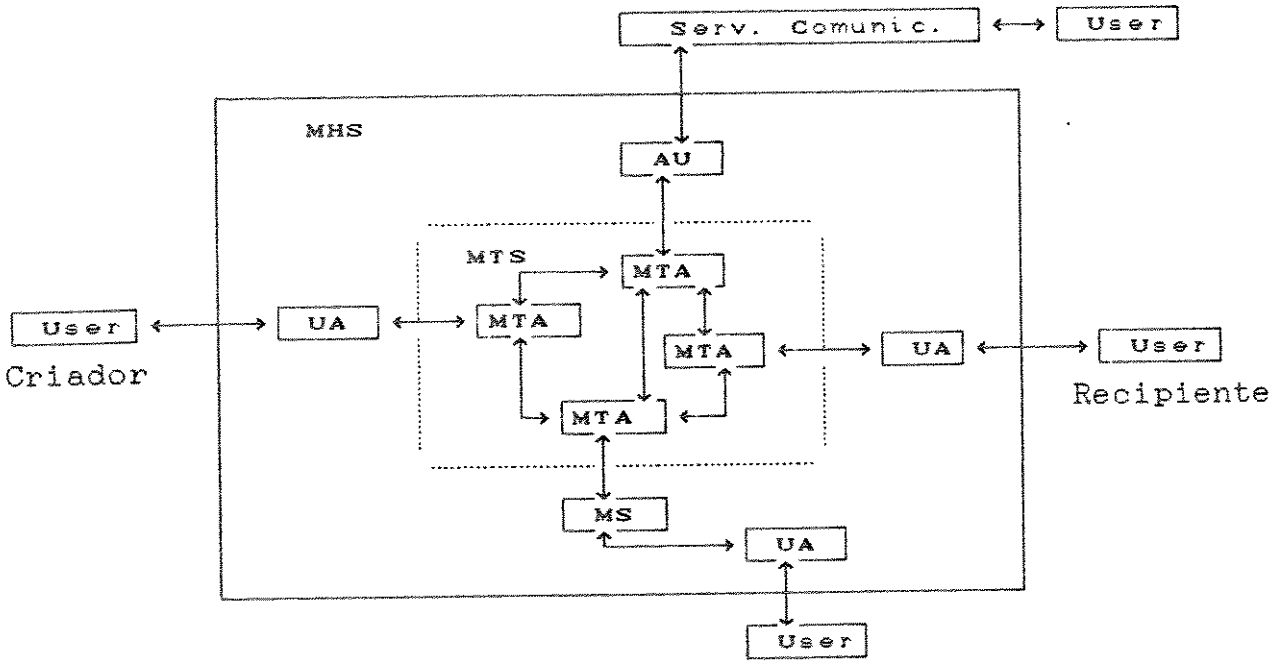


Figura 4.7 - Modelo Funcional do MHS

Os elementos básicos do modelo são:

- Criador: usuário que envia uma mensagem para um outro usuário;
- Recipiente: usuário que recebe uma mensagem de outro usuário;
- UA -"User Agent": permite ao usuário compor e submeter uma mensagem para o MTS, receber e processar uma mensagem do MTS, e manipular seu "mailbox";
- MTA -"Message Transfer Agent": em cooperação com entidades similares, transporta as mensagens através do sistema de transferência de mensagens;
- MTS -"Message Transfer System": composto de todos os MTAs individuais, é um sistema distribuído que provê o serviço de transferência;

- MS -"Message Store": provê serviços de armazenamento e gerenciamento de mensagens aos UAs;

- AU -"Access Unit": providencia uma porta para outro sistema de comunicação, como por exemplo, um sistema postal;

- MHS -"Message Handling System": composto pelos MTAs, UAs, MSs e AUs.

O UA tem algumas similaridades com os conceitos da API proposta. Segundo [HEN90], as funcionalidades do UA são:

a- realizar a interação de submissão de mensagens com o MTS;

b- realizar a interação de entrega de mensagens com o MTS;

c- executar serviços de composição e manipulação de mensagens (o usuário pode utilizar ferramentas locais, tais como: editores, processadores de texto, "spelling checkers", formatadores de documentos, entre outros);

d- recuperar e processar mensagens recebidas anteriormente;

e- opcionalmente pode ter facilidades de gerenciamento de base de dados de mensagens (por exemplo, busca de mensagem por chave).

As funcionalidades locais dos UAs não fazem parte da especificação do protocolo, pois são de interesse apenas dos usuários. A seguir são feitos alguns comentários sobre a relação entre o UA (ou UA e MS) e a API proposta:

a- O UA não é uma API, pois está incluída na camada de aplicação. Contudo, como foi visto anteriormente, pode-se introduzir funcionalidades locais aos UAs. Estas funcionalidades adicionais interessam aos usuários. Portanto, nos UAs tem-se funcionalidades de camada de aplicação e de processo de aplicação, o que caracteriza uma Interface de Aplicação. Contudo, os dois conjuntos de funcionalidades estão bem separados: procedimentos do protocolo e ferramentas locais. Em resumo, o UA não é uma API, porém tem funcionalidades próximas a de uma API pois, na realidade, o UA permite o acesso ao MTS (Sistema de Transferência de Mensagens);

b- Como uma API, o UA possui os conceitos de:

- Serviços Confirmados: o criador pode requerer que, quando o recipiente receber a mensagem enviada por ele, uma confirmação seja retornada;

- Serviços de alto nível: o criador pode enviar uma mesma mensagem para vários recipientes, solicitando um único serviço do UA;

c- Ao contrário da API proposta, o UA é muito específico da aplicação, ou seja, dificilmente pode-se generalizar a sua definição para outros tipos de aplicação. Por exemplo, o UA tem ferramentas locais próprias;

d- Se entre o UA e o MTS for introduzido o MS (por exemplo, no caso do MTA associado ao UA estar num computador diferente do que está o UA):

- O MS pode ser programado pelo UA para realizar algumas funções automaticamente. Por exemplo, o MS pode auto-enviar mensagens entregues que satisfaçam algum critério. Esta funcionalidade é similar à que as unidades funcionais IFA e IP da API proposta têm;

- O MS armazena as mensagens manipuladas pelo UA. Portanto, o MS tem alguma funcionalidade similar à da unidade funcional VDRDB da API proposta, no tocante ao acesso ao Sistema de Arquivos Local;

e- Ao contrário do que normalmente ocorre com a API proposta, o UA pode estar num computador diferente do computador que contém o MTA ou MS (Sistema de Transferência de Mensagens). Isto implica na existência de um protocolo de comunicação entre o UA e o MS.

O protocolo de comunicação entre um UA e um MS é composto por três SASEs (Figura 4.8):

- MSSE -"Message Submission Service Element": para suportar serviços de submissão de mensagens;

- MRSE -"Message Retrieval Service Element": para suportar serviços de recuperação de mensagens;

- MASE -"Message Administration Service Element": para suportar serviços de administração de mensagens.

O protocolo é assimétrico, sendo o UA a estação cliente, e o MS a estação servidora. O UA e o MS devem controlar interações entre os diversos SASEs da camada de aplicação. Por exemplo, o UA, numa associação, a partir da recuperação de uma mensagem antiga, pode querer submeter uma nova mensagem. As interações associadas aos diversos SASEs, a nível de interface de aplicação, correspondem às funcionalidades das unidades TAOCF e RAOCF da API proposta.

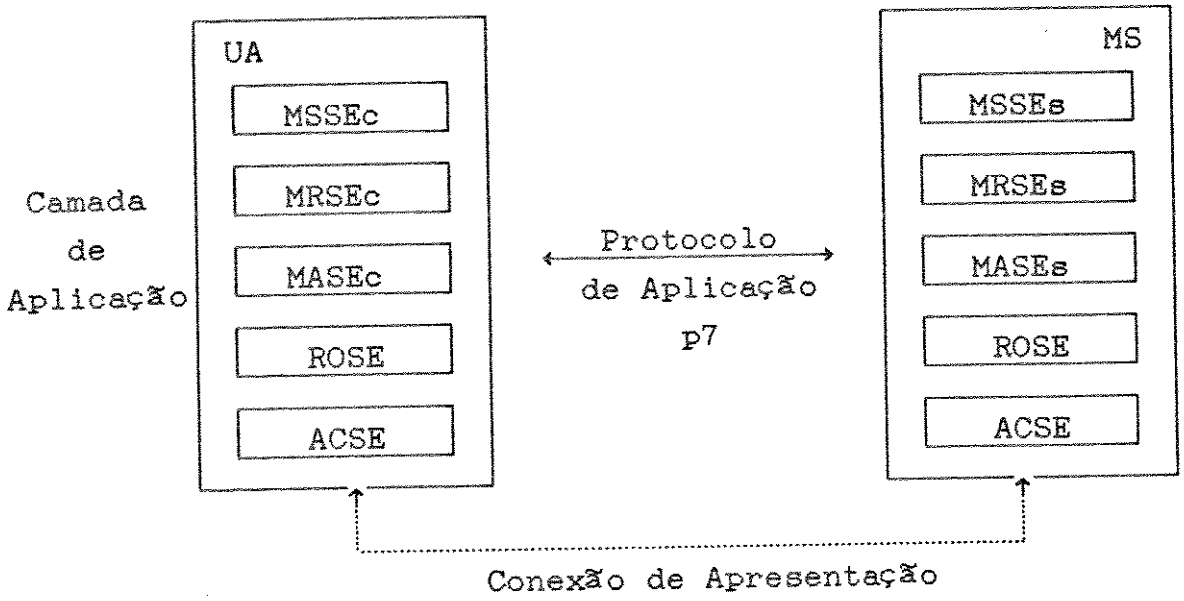


Figura 4.8 - Modelo de Protocolo de Acesso a um MS

A existência dos três SASEs nos objetos UA e MS implica na possibilidade de existência de três conexões entre os dois objetos, e de três portas (pontos de conexão entre objetos) no UA e três no MS (Figura 4.9).

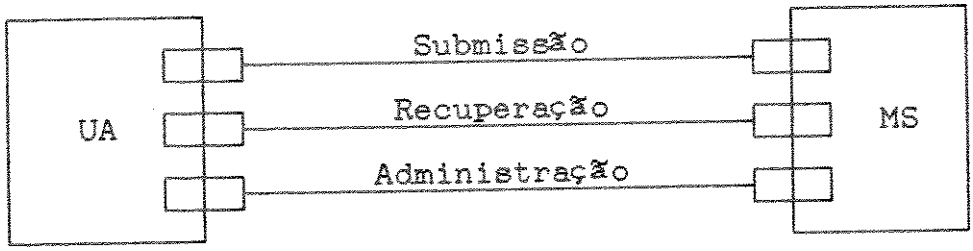


Figura 4.9 - Portas na Conexão UA / MS

PROTOCOLO DS:

O Protocolo DS (Serviço de Diretório) [ISO9594] também tem o conceito de UA (Agente de Usuário) de maneira equivalente ao Protocolo

X-400. O padrão CCITT do Serviço de Diretório é a Recomendação X-500, e o padrão ISO é o documento [ISO9594]. O Serviço de Diretório permite ao usuário ler, buscar e, em alguns casos, modificar as informações armazenadas no Diretório. O Diretório contém informações sobre os objetos acessíveis na rede.

A figura 4.10 apresenta o modelo funcional do DS. O Serviço de Diretório contém quatro participantes, dos quais três aparecem no modelo funcional. Os participantes são:

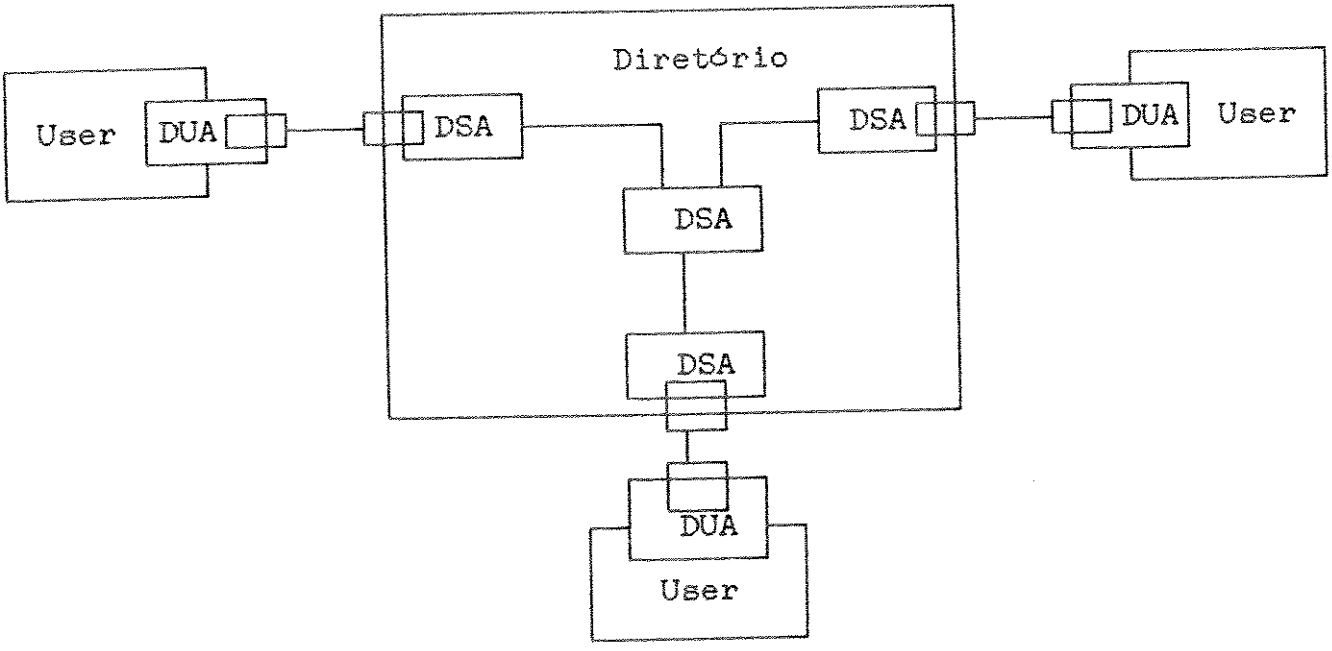


Figura 4.10 - Modelo Funcional do DS

- Usuário do Diretório (User): pessoa ou processo de aplicação que requer os serviços de Diretório;
- DUA -"Directory User Agent": interface que permite ao usuário acessar o Diretório através do mapeamento ("binding") a um dos pontos de acesso do DSA associado a este UA;
- DSA -"Directory Service Agent": componente do Diretório. Os diversos DSAs cooperam entre si, para providenciarem um Serviço de Diretório Integrado. O DSA acessa o DIB para recuperar as informações requeridas;
- DIB -"Directory Information Base": Base de informação

sobre os objetos acessíveis na rede.

O protocolo de comunicação entre o DUA e o DSA é composto por três SASEs como mostra a figura 4.11. Um dos SASEs oferece serviços relacionados com operações de leitura, outro com operações de busca, e finalmente o outro com operações de modificação. A existência dos três SASES nos objetos DUA e DSA implica na possibilidade de existência de três conexões entre os dois objetos, associadas a pontos de acesso no DSA (Figura 4.12).

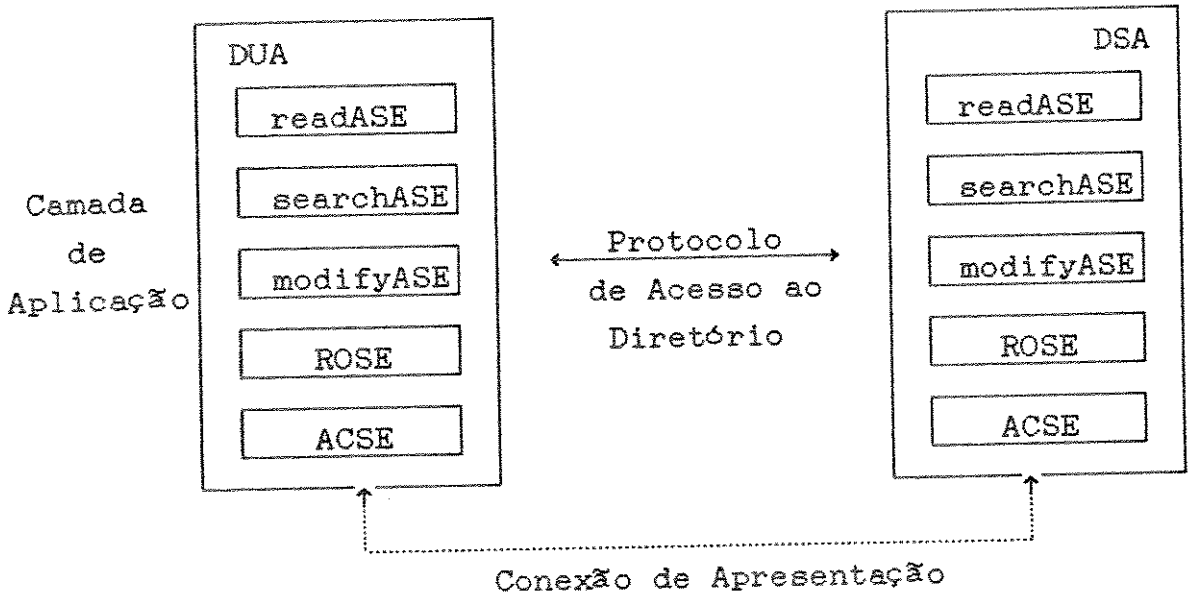


Figura 4.11 - Modelo de Protocolo de Acesso ao Diretório

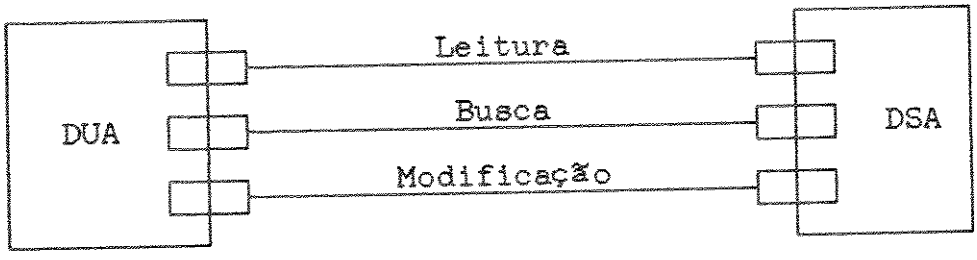


Figura 4.12 - Modelo de Serviço de Acesso ao Diretório

O conceito de DUA é muito próximo do conceito de UA para o X-400.

Na realidade, o conceito é o mesmo, a menos dos pontos relacionados às características próprias do protocolo. Por exemplo, o DS não necessita de ferramentas locais para suportar seus serviços. O X-400 necessita. Portanto, o UA do X-400 possui ferramentas locais adicionais. O DUA não possui tais ferramentas.

Desta maneira, o DUA também tem similaridades com os conceitos da API proposta, pois de fato o DUA é uma interface entre o usuário e o Diretório, permitindo ao primeiro o acesso ao segundo. Contudo, o DUA não é uma API como a proposta, pois pertence à camada de aplicação, tendo inclusive três protocolos de aplicação.

Análise similar à que foi feita na comparação entre o UA e a API proposta pode ser feita para a comparação entre o DUA e a API proposta. Em particular, foi visto que o protocolo de comunicação entre o DUA e o DSA possui três SASEs. O DUA e o DSA devem controlar as interações entre os diversos SASEs na camada de aplicação. Por exemplo, o usuário, a partir da leitura de uma entrada do Diretório, pode querer modificá-la. As interações associadas aos diversos SASEs, a nível de interface de aplicação, correspondem às funcionalidades das unidades TAOCF e RAOCF da API proposta.

4.4 - PROTOCOLO ROSE: CONCEITOS DE "BINDING" E OPERAÇÃO REMOTA

O Protocolo de Aplicação ROSE ("Remote Operations Service Element") [ISO9072] é utilizado em ambientes de sistemas abertos distribuídos, para suportar aplicações interativas. Nestas aplicações, uma Entidade de Aplicação envia uma requisição de serviço a uma outra Entidade de Aplicação remota que realiza o serviço, e envia uma resposta para a Entidade de Aplicação local. A um serviço realizado desta maneira, dá-se o nome de Operação Remota. O Protocolo ROSE suporta cinco classes de Operações Remotas:

- Classe 1: Operações Síncronas, reportando sucesso ou falha (resultado ou erro);
- Classe 2: Operações Assíncronas, reportando sucesso ou falha (resultado ou erro);
- Classe 3: Operações Assíncronas, reportando somente falha

(erro), se houver;

- Classe 4: Operações Assíncronas, reportando somente sucesso (resultado), se houver;

- Classe 5: Operações Assíncronas, sem resposta.

Nas operações assíncronas, o solicitante dos serviços pode continuar a requisitar outros serviços sem obter resposta dos serviços anteriores. Nas operações síncronas, o requisitante de serviços espera pela resposta do serviço solicitado, para depois requisitar novo serviço.

O Protocolo ROSE possui os seguintes serviços:

a- RO_INVOKE: Serviço que habilita a Entidade de Aplicação local a requisitar uma operação a ser realizada numa Entidade de Aplicação remota;

b- RO_RESULT: Serviço que habilita a Entidade de Aplicação, que realizou uma operação remota, retornar a resposta positiva desta operação à Entidade de Aplicação que requisitou a operação;

c- RO_ERROR: Serviço que habilita a Entidade de Aplicação, que foi solicitada a realizar uma operação remota, retornar a resposta negativa desta operação à Entidade de Aplicação que requisitou a operação;

d- RO_REJECT_U: Serviço que habilita uma Entidade de Aplicação a rejeitar uma requisição ou resposta de uma outra Entidade de Aplicação, se o Usuário do ROSE detectou um problema;

e- RO_REJECT_P: Serviço que habilita o Provedor do ROSE informar ao Usuário do ROSE sobre um problema detectado.

O Protocolo ROSE define o uso de operações ligadas, que são formadas por uma operação pai e uma ou mais operações filhas. Se a operação pai é requisitada pela AE 1, para ser executada na AE 2, as operações filhas são requisitadas durante a realização da operação pai pela AE 2, para serem executadas na AE 1. Uma operação filha pode ser uma operação pai de outro conjunto de operações ligadas.

A especificação [ISO9072] também define uma notação para Operações Remotas para especificação de ASEs e Contextos de Aplicação, e um conjunto de macros que constituem a interface de operação com o Contexto de Aplicação. As macros definidas são:

- BIND: contém uma operação para estabelecimento de uma associação de aplicação;

- UNBIND: contém uma operação para liberação de uma associação de aplicação;

- OPERATION: contém um conjunto de operações remotas para serem requisitadas;

- ERROR: contém um conjunto de erros associados a operações remotas.

A figura 4.13 apresenta o modelo de um Contexto de Aplicação envolvendo Operações Remotas (operação de BIND, operação de UNBIND e operações específicas). O Elemento de Usuário (UE) estabelece uma associação invocando a operação de BIND. A operação de BIND é mapeada num serviço ACSE ou RTSE. Após o estabelecimento da associação, o UE pode invocar operações específicas (dos SASEs). Estas operações são mapeadas nos serviços do ROSE. O UE, quando desejar liberar a associação, invoca a operação de UNBIND. Esta operação é mapeada num serviço ACSE ou RTSE.

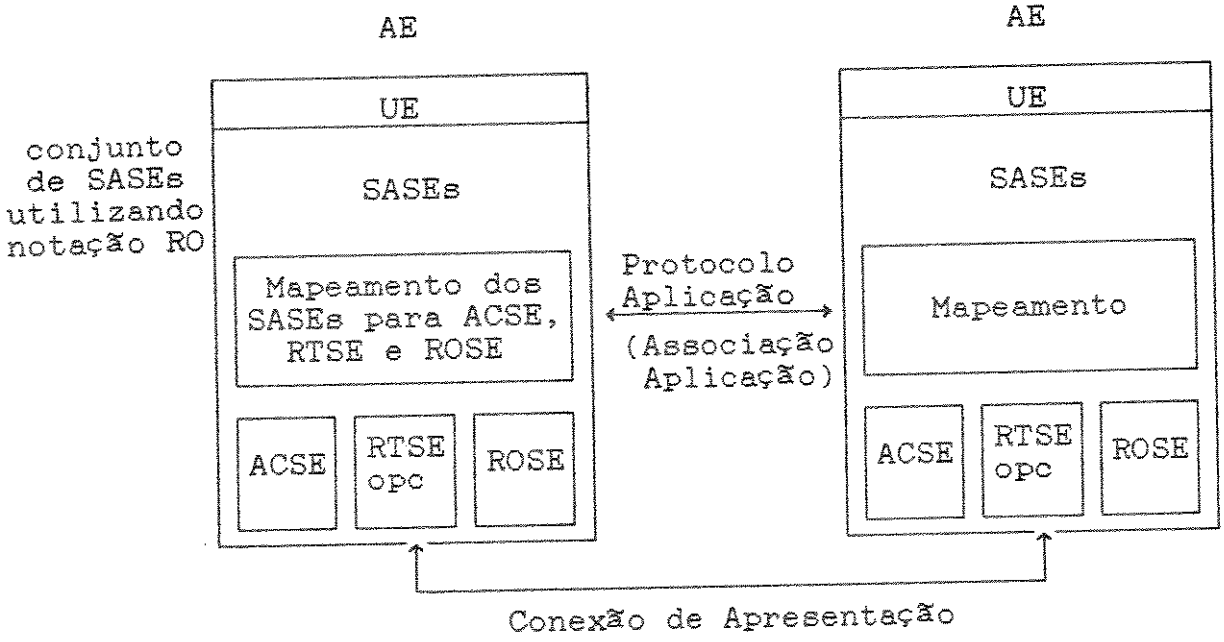


Figura 4.13 - Modelo de Contexto de Aplicação envolvendo Operações Remotas

Se um sistema de comunicação possui API e o Protocolo ROSE na camada de aplicação, a API pode facilitar o processamento da camada de aplicação, e o ROSE pode facilitar o processamento da API:

a- As funcionalidades do Elemento de Usuário (UE) podem ser absorvidas pela API (como visto na seção 1.2.1). Neste caso, a API invoca as operações de BIND e UNBIND, além de invocar as operações específicas dos SASEs;

b- A função de mapeamento dos serviços SASEs para os serviços dos protocolos ACSE, RTSE e ROSE apresentada na figura 4.13 corresponde a uma das funcionalidades do SACF apresentada na seção 1.2.4, e pode, portanto, ser executada em parte pela API. Por exemplo, a API ao invocar as operações de BIND e UNBIND, pode solicitar estes serviços diretamente aos Protocolos ACSE ou RTSE;

c- A especificação de Operação Remota define Contextos de Aplicação e ASEs, e permite o estabelecimento de uma associação através da operação de BIND, cujo Contexto de Aplicação possui diversos ASEs. Este fato é utilizado pela API para oferecer serviços aos APs. Certas interações entre os ASEs são controladas a nível de camada de aplicação. Isto facilita o processamento da API;

d- O Protocolo ROSE pode ser invocado pelos diversos SASEs que compõem o Contexto da Aplicação. Desta maneira, o Protocolo ROSE provê um certo controle das interações entre os diversos SASEs. Por exemplo, num conjunto de operações ligadas, as operações filhas podem ser as associadas a SASEs diferentes. Esta funcionalidade do Protocolo ROSE corresponde às funcionalidades do SACF e do MACF da camada de aplicação. Como foi visto na seção 1.2.4, se as funcionalidades do SACF e do MACF forem executadas na camada de aplicação, este fato facilita o processamento da API (ou vice-versa);

e- O Protocolo ROSE oferece, como a API, mas a nível de camada de aplicação:

- Serviços confirmados e não confirmados: dependendo da classe de operação, reporta-se ou não o resultado e/ou o erro da operação;

- Serviços de alto nível: uma operação pai pode gerar na entidade par uma seqüência de operações filhas;

- Serviços síncronos e assíncronos: dependendo da classe de operação.

Como o ROSE oferece estes serviços a um nível diferente da API, o ROSE simplifica a API, mas não a substitui.

Portanto, pode-se concluir que a API e o Protocolo ROSE possuem funcionalidades em comum, mas aplicadas em níveis diferentes (API: interface de aplicação; ROSE: camada de aplicação). Desta maneira, se a API executa algumas destas funcionalidades comuns, a camada de aplicação (protocolo ROSE) é simplificada. Se a camada de aplicação (protocolo ROSE) executa algumas funcionalidades comuns, a API é simplificada.

Desta forma, a existência conjunta do Protocolo ROSE na camada de aplicação e da API num sistema de comunicação é adequada, pois um completa o outro.

4.5 - MODELO DE APRESENTAÇÃO DE API: VISÃO EXTERNA

Na seção 1.4.5 foi proposto um Modelo Geral de API, a partir dos conceitos já conhecidos e levando em consideração as características dos diversos protocolos de aplicação existentes. O Modelo Geral proposto é composto por blocos funcionais (que executam funções específicas e bem definidas) e comunicações (ligações) entre os mesmos. Este Modelo corresponde à "visão interna" da API, ou seja, define a estrutura interna da API.

Nesta seção será proposto um Modelo de Apresentação de API que corresponde à "visão externa" da API, ou seja, o Modelo define a visão que o AP (Processo Aplicativo) tem dos serviços oferecidos pela API. Para este Modelo, a API é uma "caixa-preta" da qual não se conhece sua estrutura interna. Contudo, a "visão externa" da API depende obviamente do suporte interno aos serviços oferecidos, mas não depende de como este suporte é implementado. O Modelo Geral de API (seção 1.4.5) justamente propõe uma maneira de como este suporte interno pode ser estruturado.

O Modelo de Apresentação da API é mostrado na figura 4.14, e possui as seguintes características:

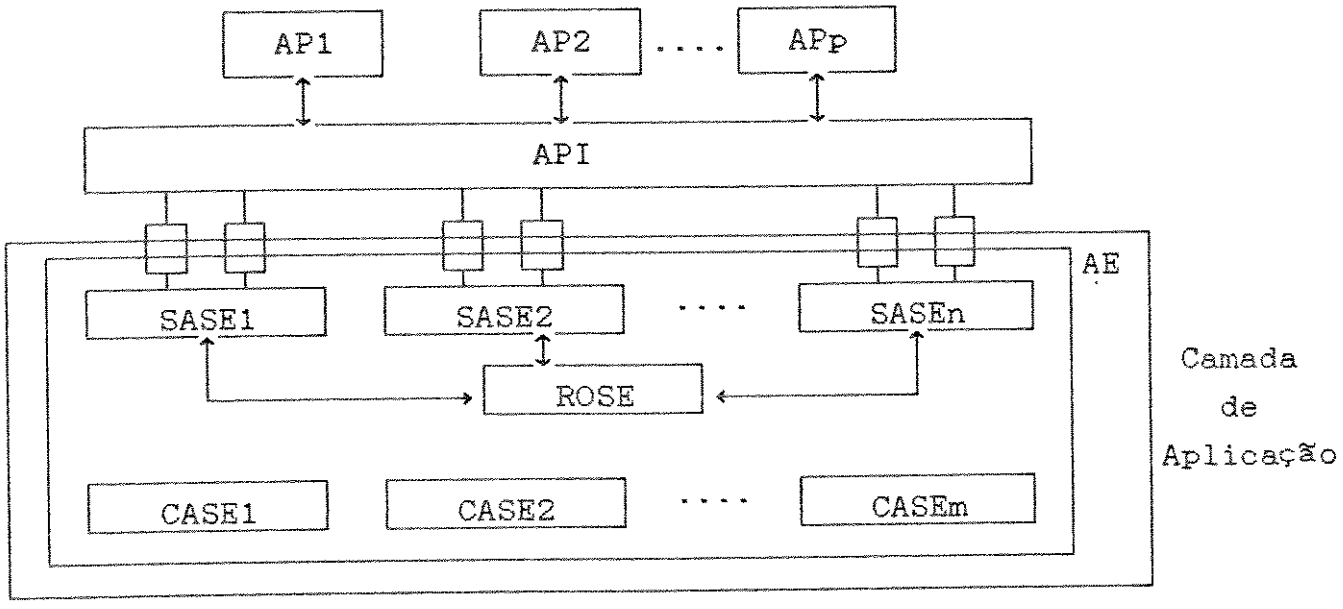


Figura 4.14 - Modelo de Apresentação de API

API ÚNICA:

a- O sistema de comunicação possui uma única API, independentemente de quantos ASEs e SASEs tenha a Entidade de Aplicação. Por exemplo, se a AE possui os SASEs RDA e TP, existirá uma única API que deverá oferecer os serviços de ambos os protocolos de aplicação ao usuário, e realizar o controle da interação entre os dois protocolos a nível de interface;

LINGUAGEM DE PROGRAMAÇÃO:

b- A API depende da linguagem de programação dos APs, pois os APs chamam funções da biblioteca da API. Contudo, esta dependência é apenas em relação à chamada de funções, ou seja, como são passados e retornados (empilhados e desempilhados) os parâmetros das funções. Portanto, pode-se flexibilizar esta restrição, impondo que os APs empilhem os parâmetros de acordo com a forma de sua utilização pela

API, independentemente da linguagem de programação do AP.

Por exemplo, a linguagem Pascal empilha os parâmetros efetivos de uma chamada de função da esquerda para a direita, enquanto a linguagem C empilha os parâmetros efetivos da direita para a esquerda.

Se a API foi projetada para a linguagem C, e o AP está escrito em Pascal, o AP deve empilhar os parâmetros na ordem em que a API os utilize corretamente, ou seja, o AP deve declarar os parâmetros efetivos na ordem inversa dos parâmetros da função na especificação da API. Se a função f da API tem os parâmetros a, b e c, no AP a chamada à função f deve ser f(c,b,a). Desta maneira, pode-se flexibilizar a dependência da API sobre a linguagem de programação do AP para linguagens similares, tais como Pascal, C e Modula-2;

SOLUÇÃO INTERMEDIÁRIA ENTRE A API PROPOSTA E O "SOCKET":

c- A API proposta na seção 1.4.5 possui vários blocos funcionais, que dão suporte para serviços oferecidos externamente, tais como: serviços confirmados, serviços de alto nível, serviços respondedores e serviços de resposta automática, entre outros. Esta API é bastante robusta para os conceitos de interface de aplicação e protocolos de aplicação existentes.

Por outro lado, pode-se extrapolar o conceito de "socket" para a camada de aplicação, como sendo um ponto final de comunicação para a camada. Desta forma, o "socket" corresponderia a um ponto de acesso a um SASE, ao invés de corresponder a um ponto de acesso ao protocolo de transporte. O primeiro componente da tripla <protocolo, endereço, processo>, que define um "socket", indicaria a qual SASE está associado o "socket". O "socket"definido desta maneira é uma API para a camada de aplicação.

Contudo, como foi visto na seção 4.2, esta API é simples, não oferecendo muitos serviços aos APs, tais como: serviços confirmados, de alto nível, respondedores e de resposta automática, entre outros. De fato, a API do tipo "socket" não possui a maioria dos blocos funcionais da API proposta.

Portanto, de um lado tem-se a API proposta robusta, e do outro o

"socket" simples. A solução do Modelo de Apresentação de API proposto consiste em:

- Implementar a API como uma única instância ou tendo várias instâncias, dependendo da complexidade do sistema local;
- Adotar o conceito de "socket" como ponto final de comunicação da camada de aplicação. Filas associadas aos "sockets" existirão para troca de primitivas entre a API e os SASEs;
- Adotar os conceitos dos blocos funcionais da API proposta como suporte interno aos serviços oferecidos aos APs;
- Permitir soluções intermediárias entre a API proposta completa e o "socket" simples. Por exemplo, pode-se desejar que uma API ofereça serviços confirmados e de alto nível, mas não ofereça serviços respondedores e de resposta automática. Para obter esta "visão externa" da API, o suporte interno deve ter os blocos funcionais que realizam os serviços confirmados e de alto nível, e não ter os blocos funcionais que realizam os serviços respondedores e de resposta automática. Em geral, para construir estas soluções intermediárias, precisa-se retirar blocos funcionais e comunicações entre blocos funcionais da API proposta.

API MODULAR:

d- As estações, que utilizam o sistema de comunicação, podem ser mais simples ou mais complexas. Os APs das estações mais simples, que desejam utilizar um protocolo de aplicação, podem chamar apenas um subconjunto das funções da API especificada para o protocolo em questão.

Para resolver este caso, o Modelo de Apresentação proposto permite que as funções oferecidas aos APs sejam divididas e oferecidas em unidades funcionais, como no Modelo de API do Projeto MAP/TOP. As funções, que pertencem a uma mesma unidade funcional, têm uma certa relação no tocante a sua aplicabilidade.

Desta forma, uma API, para um dado protocolo de aplicação, pode implementar as funções de todas as unidades funcionais, enquanto outra pode implementar as funções apenas de algumas delas. Para suportar

esta estrutura modular externa, alguns blocos funcionais internos à API devem também ter estrutura modular;

DIVERSOS TIPOS DE "SOCKETS":

e- As especificações de protocolos da camada de aplicação muitas vêzes dividem os diferentes tipos de serviços em SASEs diferentes. Como foi visto na seção 4.3, o MHS possui os SASEs MSSE, MRSE e MASE, e o Serviço de Diretório os SASEs readASE, searchASE e modifyASE. Estão também surgindo aplicações que requerem na AE mais de um SASE: por exemplo, o acesso às Bases de Dados Remotas pode requerer os SASEs RDA e TP.

Nestes casos, é necessário para a API ter pontos de acesso nos diversos SASEs, ou seja, portas de acesso ou "sockets" específicos para comunicação com os diversos SASEs. Portanto, a API pode solicitar a criação de diversos "sockets", que correspondem aos diversos pontos finais de comunicação dos diversos SASEs, ou seja, dos diversos serviços oferecidos pela camada de aplicação. Para o estabelecimento de conexão (associação) através de um "socket" (ou um conjunto de "sockets"), a API pode oferecer aos APs uma operação de BINDING no sentido utilizado pelo protocolo ROSE, que corresponde à operação de CONNECT para um "socket" na estação cliente. Quatro operações de BINDING são propostas:

UMA ASSOCIAÇÃO - UM SASE:

1- BINDING(protocolo, socket_local, socket_remoto): permite o estabelecimento de uma conexão (associação) entre o <socket_local> e o <socket_remoto> (Figura 4.15).

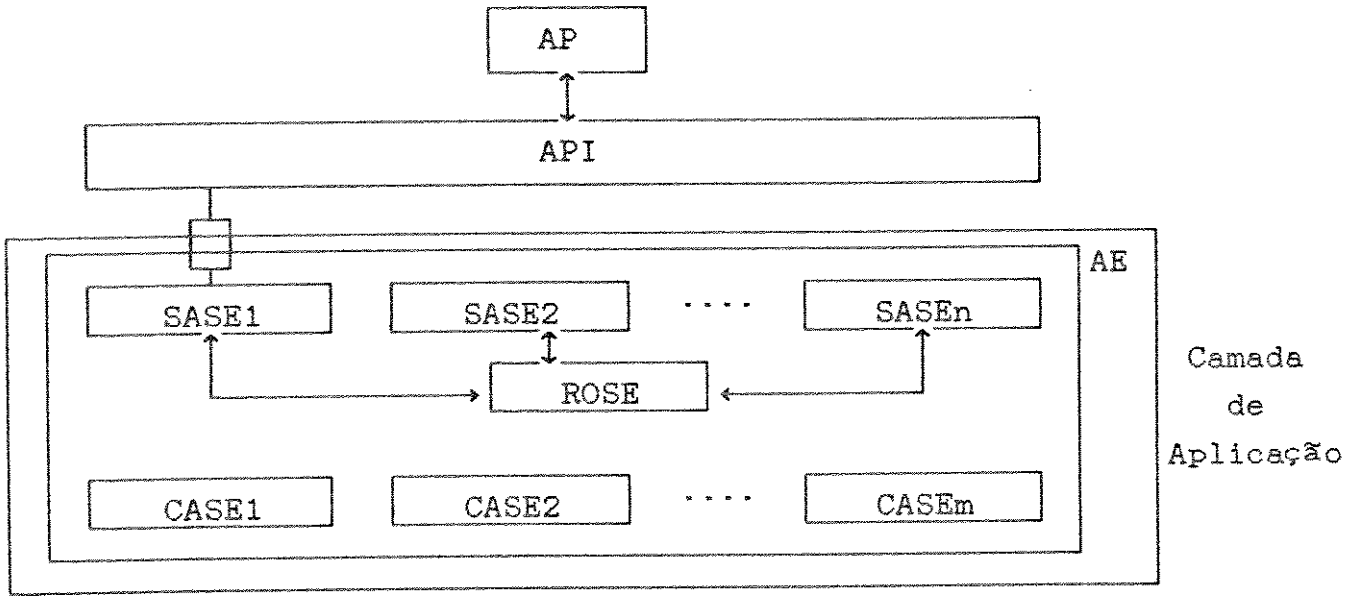


Figura 4.15 - Primeira Operação de BINDING: Uma Associação que utiliza um SASE

Se o parâmetro <protocolo> contém um protocolo de aplicação, a API solicita a criação do <socket_local> com o SASE correspondente. Se o parâmetro <protocolo> é NULL, significa que o "socket" a ser utilizado localmente já foi criado pela chamada de outra função da API, e é o "socket" do parâmetro <socket_local> apresentado na forma <identificador de socket>. O parâmetro <socket_remoto> pode ser apresentado sob a forma <identificador de socket> ou <endereço remoto, processo remoto>, e deve estar relacionado com o protocolo de aplicação apresentado no parâmetro <protocolo> (se este parâmetro é diferente de NULL) ou com o protocolo de aplicação associado com o <socket_local> (se o parâmetro <protocolo> é igual a NULL).

Esta operação de BINDING corresponde ao estabelecimento de uma associação entre dois APs, utilizando um único protocolo de aplicação do tipo SASE. Através desta associação, o AP pode requisitar serviços não confirmados, confirmados, de alto nível, respondedores e de resposta automática, desde que somente um SASE esteja envolvido;

UMA ASSOCIAÇÃO - DIVERSOS SASEs:

2- BINDING(lista_de(protocolo, socket_local, socket_remoto)): permite o estabelecimento de uma conexão (associação) a nível de interface de aplicação entre os "sockets" locais e os "sockets" remotos (Figura 4.16).

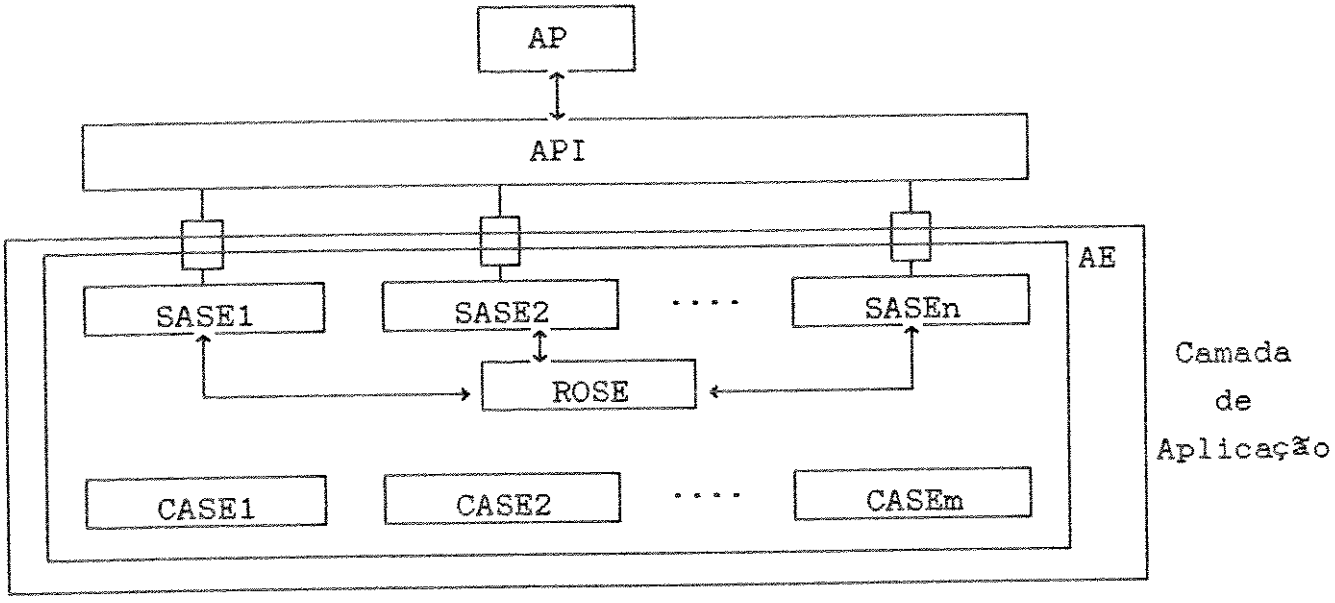


Figura 4.16 - Segunda Operação de BINDING: Uma Associação que utiliza diversos SASEs

O número de "sockets" locais é igual ao número de "sockets" remotos, e cada par define uma associação de "sockets". A associação a nível de interface engloba diversas associações de "sockets". Só pode existir um par de "sockets" (local e remoto) relacionado com cada SASE numa mesma associação da API. Portanto, dada uma primitiva a ser enviada ou recebida, a API sabe a que "socket" deve enviar, ou de que "socket" deve receber. É o "socket" relacionado com o SASE a que pertence a primitiva.

As diversas maneiras de preenchimento dos parâmetros <protocolo, socket_local, socket_remoto> pelos APs são as mesmas que foram apresentadas para a operação anterior.

Esta operação de BINDING corresponde ao estabelecimento de uma associação entre dois APs, utilizando mais de um protocolo de aplicação do tipo SASE. Através desta associação, o AP pode requisitar serviços não confirmados, confirmados, de alto nível, respondedores e de resposta automática, envolvendo vários SASEs;

DIVERSAS ASSOCIAÇÕES - UM SASE:

3- BINDING(protocolo, lista_de(socket_local), lista_de(socket_remoto)): permite o estabelecimento de diversas conexões (associações) a nível de interface, relacionadas com um mesmo SASE, entre os "sockets" locais e os "sockets" remotos (Figura 4.17).

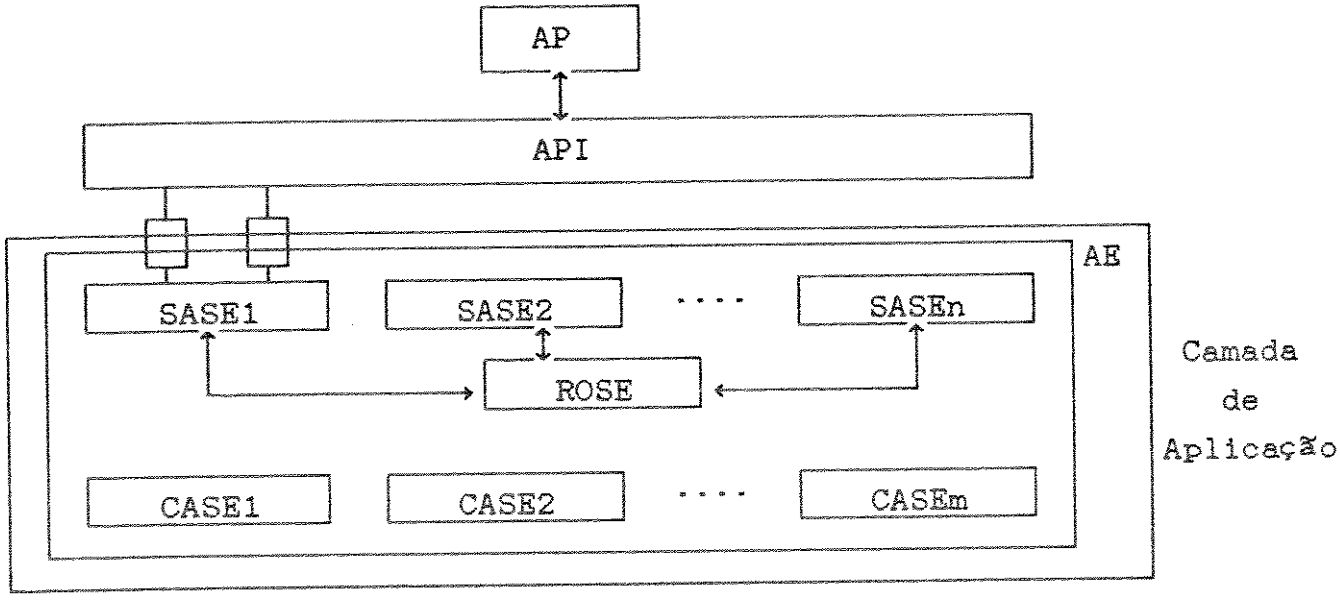


Figura 4.17 - Terceira Operação de BINDING: Diversas Associações que utilizam o mesmo SASE

O número de "sockets" locais deve ser igual ao número de "sockets" remotos, e cada par define uma associação de "sockets" e uma associação a nível de interface. Se o parâmetro <protocolo> contém um protocolo de aplicação, a API solicita a criação dos "sockets" locais com o SASE correspondente. Se o parâmetro <protocolo> é NULL,

significa que os "sockets" a serem utilizados localmente já foram criados por outras chamadas de funções da API, e são os "sockets" do parâmetro <lista_de <socket_local>>, onde cada "socket" é apresentado na forma <identificador de socket>. Todos os "sockets" locais devem estar relacionados com o mesmo SASE. Os "sockets" do parâmetro <lista_de <socket_remoto>> podem ser apresentados sob a forma <identificador de socket> ou <endereço remoto, processo remoto>, e devem estar relacionados com o protocolo de aplicação apresentado no parâmetro <protocolo> (se este parâmetro é diferente de NULL), ou com o protocolo de aplicação associado com os "sockets" locais (se o parâmetro <protocolo> é igual a NULL).

Esta operação de BINDING corresponde ao estabelecimento de uma invocação entre o AP local e o SASE local relacionado, e ao estabelecimento através desta invocação de diversas conexões (associações) entre o AP local e diversos APs remotos, utilizando um único protocolo de aplicação do tipo SASE. Se a operação for realizada com sucesso, deve retornar o identificador da invocação e os identificadores das diversas associações. Através da invocação estabelecida, o AP pode requisitar serviços não confirmados, confirmados, de alto nível, respondedores e de resposta automática, envolvendo diversas associações, mas relacionadas com um único SASE.

Através da invocação estabelecida, o AP pode solicitar serviços da API do tipo:

- Enviar uma mensagem através de uma associação específica (O AP conhece os identificadores das associações);
- Receber uma mensagem através de uma associação específica;
- Enviar uma mesma mensagem em todas as associações da invocação (O AP conhece o identificador da invocação);
- Receber uma única mensagem de qualquer associação da invocação (O identificador da associação, pela qual a API receber a mensagem, deve ser retornado);
- Receber uma mensagem de cada associação da invocação.

DIVERSAS ASSOCIAÇÕES - DIVERSOS SASEs:

4- BINDING(lista_de(lista_de(protocolo, socket_local, socket_remoto))): permite o estabelecimento de diversas conexões (associações) a nível da interface de aplicação entre os "sockets" locais e os "sockets" remotos, envolvendo diversos SASEs em cada conexão (Figura 4.18).

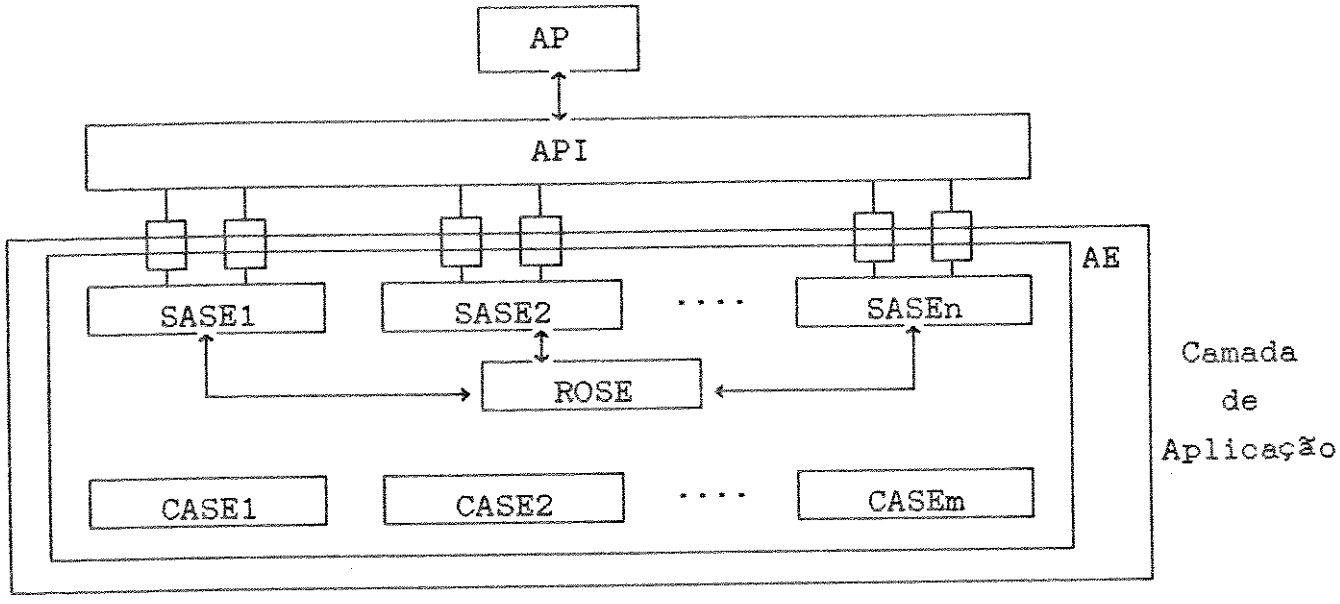


Figura 4.18 - Quarta Operação de BINDING: Diversas Associações que utilizam diversos SASEs

Uma lista <lista_de <protocolo, socket_local, socket_remoto>> define uma conexão (associação) a nível de interface de aplicação, que engloba diversas associações de "sockets". Cada par <socket_local, socket_remoto> define uma associação de "sockets". Só pode existir um par de "sockets" (local e remoto) relacionado com cada SASE numa mesma associação da API.

As diversas maneiras de preenchimento dos parâmetros <protocolo, socket_local, socket_remoto> pelos APs são as mesmas que foram apresentadas para a primeira operação.

Esta operação de BINDING corresponde ao estabelecimento de uma

invocação entre o AP local e a AE local, que contém diversos SASEs, e ao estabelecimento através desta invocação de diversas conexões (associações) entre o AP local e diversos APs remotos, onde cada conexão tem como contexto de aplicação diversos dos SASEs pertencentes à AE. Se a operação for realizada com sucesso, deve retornar o identificador da invocação e os identificadores das diversas associações. Através da invocação estabelecida, o AP pode requisitar serviços não confirmados, confirmados, de alto nível, respondedores e de resposta automática, envolvendo diversas associações, onde cada associação está relacionada com diversos SASEs.

Através da invocação estabelecida, o AP pode solicitar serviços da API do tipo dos citados na operação anterior, com a vantagem que um serviço pode estar relacionado com um SASE, o seguinte relacionado com outro SASE, e assim por diante.

A API não oferece nenhum serviço que implica no envio de uma mensagem relacionada com um SASE através de uma associação, e no envio de outra mensagem relacionada com outro SASE através de outra associação da invocação.

Para esta operação de BINDING, o caso de enviar uma mesma mensagem em todas as associações da invocação significa enviar uma mesma mensagem em todas as associações da invocação, que tenham no seu contexto de aplicação o SASE relacionado com a mensagem.

Se for imposto para esta operação de BINDING, que todas as associações de uma invocação têm que possuir o mesmo contexto de aplicação, ou seja, têm que estar relacionadas com o mesmo subconjunto de SASEs da AE, a sintaxe desta operação de BINDING pode ser simplificada para:

BINDING(lista_de(protocolo, lista_de(socket_local), lista_de(socket_remoto)))

onde, a lista <lista_de <protocolo, lista_de <socket_local>, lista_de <socket_remoto>>> define os protocolos de aplicação que compõem o contexto das associações. Para cada parâmetro <protocolo> o número de "sockets" locais deve ser igual ao número de "sockets" remotos, e cada par define uma associação de "sockets". Cada associação assim definida pertence a uma associação diferente a nível

de interface de aplicação.

As diversas maneiras de preenchimento dos parâmetros <protocolo, lista_de <socket_local>, lista_de <socket_remoto>> pelos APs são as mesmas que foram apresentadas para a operação anterior. Para a lista <lista_de <protocolo, lista_de <socket_local>, lista_de <socket_remoto>>>, cada protocolo de aplicação (SASE) só pode ser definido uma vez no parâmetro <protocolo>. Devido ao fato de que todas as associações têm o mesmo contexto de aplicação, todas as listas <lista_de <socket_local>> e <lista_de <socket_remoto>> devem ter o mesmo número de elementos, independentemente de qual seja o parâmetro <protocolo> associado.

COMENTÁRIOS SOBRE AS OPERAÇÕES DE "BINDING":

No Modelo de Apresentação de API proposto devem existir quatro operações de UNBINDING, que liberam as associações dos "sockets" ou dos conjuntos de "sockets" envolvidos numa associação ou invocação a nível de interface de aplicação. Cada uma destas operações de UNBINDING está relacionada com uma das operações de BINDING vistas anteriormente.

As operações de UNBINDING relacionadas com as operações 1 e 2 de BINDING têm como parâmetro de entrada o identificador da associação retornado pela operação de BINDING correspondente. As operações de UNBINDING relacionadas com as operações 3 e 4 de BINDING têm como parâmetro de entrada o identificador da invocação retornado pela operação de BINDING correspondente.

As operações 3 e 4 de BINDING estabelecem uma invocação a nível de API entre o AP local e a AE local, e através desta invocação estabelece diversas associações a nível de API. Somente se todas as associações forem estabelecidas, que a operação de BINDING é realizada com sucesso. Portanto, estas operações de BINDING devem ser utilizadas pelos APs, quando as associações, pertencentes à invocação, estão relacionadas entre si a nível de interface de aplicação.

As operações de BINDING propostas suportam diversos SASEs na AE. Desta forma, o Modelo de Apresentação de API proposto estimula a

divisão dos serviços da camada de aplicação em diversos SASEs (o que implica na existência de diferentes tipos de "sockets"), como ocorre, por exemplo, nos casos do MHS e DS vistos anteriormente.

PROTOCOLO ROSE:

f- O Modelo de Apresentação de API proposto sugere a introdução do Protocolo ROSE na camada de aplicação, principalmente quando esta camada for composta por diversos SASEs. Como foi visto na seção 4.4, a existência do Protocolo ROSE simplifica a API, pois o ROSE possui um certo controle das interações entre os diversos SASEs a nível da camada de aplicação. Devido ao fato do Modelo proposto estimular a existência de diversos SASEs na camada de aplicação (existência de diferentes tipos de "sockets"), o Protocolo ROSE também é útil para receber as diversas primitivas dos SASEs, e tratá-las como operações remotas, independentemente de qual SASE enviou a primitiva. O conceito de operações ligadas do Protocolo ROSE facilita o oferecimento, por parte da API, de serviços de alto nível.

CONCLUSÃO

É necessário apresentar aos Processos Aplicativos (APs) os serviços de comunicação de maneira amigável ("friendly"). Os protocolos de aplicação têm normalmente máquinas de estados simples. Contudo, estes protocolos são complexos em relação às primitivas de serviço, em geral em grande número e com estruturas complexas de parâmetros. Desta forma, tem-se de um lado um AP que deseja solicitar serviços de uma maneira simples, e do outro uma camada de aplicação que oferece serviços complexos.

A Interface de Aplicação (API) é introduzida como uma interface entre o AP e a camada de aplicação, para diminuir ("suavizar") esta diferença de complexidade, oferecendo ao AP os serviços de comunicação de maneira amigável. De fato, apesar da introdução da Interface de Aplicação, muito trabalho para acessar os serviços de comunicação ainda permanece para o AP. Afinal, o AP é que deve fornecer todas as diretrizes para a comunicação que este AP deseja efetuar. Contudo, a Interface de Aplicação pode realizar algumas tarefas que, em princípio, seriam de responsabilidade do AP, ajudando-o.

Alguns exemplos de tais tarefas comentados neste trabalho são:

a - Verificar a correção dos parâmetros passados pelo AP, quando o AP deseja invocar algum serviço da camada de aplicação;

b - Esperar e verificar a correção das "confirmações" para as diversas "requisições" de serviços;

c - Particionar um serviço de alto nível em diversos serviços mais simples;

d - Esperar por "indicações" (serviços solicitados remotamente);

e - Responder, sem intervenção do AP, a um serviço solicitado remotamente;

f - Mapear os objetos virtuais dos protocolos de aplicação nos objetos reais existentes na estação e vice-versa. Um exemplo de objeto é um Sistema de Arquivos;

g - Controlar a interação entre protocolos de aplicação, a nível de usuário (AP), em aplicações que requerem a utilização interrelacionada de diversos protocolos de aplicação;

h - Invocar os serviços da camada de aplicação, sem que o AP fique bloqueado à espera das "confirmações" (respostas enviadas remotamente) dos serviços;

i - Auxiliar a construção e a interpretação das estruturas dos objetos dos protocolos de aplicação (como por exemplo, uma lista de variáveis de rede para o Protocolo MMS);

j - Auxiliar a construção dos parâmetros das primitivas de serviço dos protocolos de aplicação. Alguns parâmetros podem ser opcionais, em outros o AP pode não estar interessado, e para alguns o AP pode desejar que, em todas as invocações de serviços dos protocolos de aplicação, tenham sempre o mesmo valor.

A comunidade de usuários dos serviços de comunicação sempre requereu funções de alto nível de um sistema de comunicação, pois assim os usuários podem resolver os seus serviços de comunicação com menos chamadas e com chamadas mais simples [RUP89]. Por exemplo, os usuários do Protocolo MMS sempre desejaram serviços de mais alto nível neste protocolo. A Interface de Aplicação para este protocolo resolve em parte esta requisição. A funcionalidade c da Interface oferece estes serviços de alto nível para os APs.

A funcionalidade d fornece ao AP uma grande vantagem: O AP só é interrompido pela Interface de Aplicação, se este AP programou a Interface para tal, mesmo para recepção de indicações de serviços. Desta forma, conclui-se que é o usuário que controla o sistema de comunicação, e não o sistema de comunicação que controla o usuário.

Este trabalho propõe soluções para as funcionalidades e, f e g. Estas soluções surgem como extensões ao Modelo de Interface de Aplicação apresentado em [MAP88b].

A funcionalidade h é obtida pela existência de funções de gerenciamento de eventos na Interface de Aplicação. A utilização destas funções não é obrigatória, mas tem a vantagem de permitir aos APs chamarem assincronamente a Interface de Aplicação.

A funcionalidade i é obtida pela existência de funções auxiliares

na Interface de Aplicação. Algumas conclusões sobre o seu uso foram apresentadas na seção 3.2.2.3. As funções auxiliares facilitam também a manutenção dos APs. Por exemplo, trocar o nome de uma lista de variáveis torna-se simples, pois a estrutura da lista de variáveis é a mesma. Portanto, não é necessário modificar todas as funções auxiliares, utilizadas na construção da lista, para obter a nova lista de variáveis.

A funcionalidade j é obtida pela existência de DCBs como parâmetros das funções da Interface de Aplicação. Algumas conclusões sobre o seu uso foram apresentadas na seção 1.3.1.

Outras vantagens de ter a Interface de Aplicação nos sistemas de comunicação são:

a - A Interface provê portabilidade aos APs para diferentes ambientes de implementação;

b - A Interface reduz custos de treinamento de programadores, que utilizam a interface em ambientes múltiplos;

c - A Interface estimula a geração de novos APs (sistemas aplicativos) que utilizam sistemas de comunicação, pois a Interface fornece uma biblioteca padronizada de funções de comunicação (muito bem definida);

d - Com a definição do conceito de Interfaces de Serviços ([TSC90] e [WOL90]), a Interface de Aplicação pode ser vista como uma Interface de Serviços específica para comunicação, que pode ser utilizada diretamente pelos APs ou por outras Interfaces de Serviços.

A seguir comentam-se algumas conclusões desta tese.

CONCLUSÕES RELATIVAS AO MODELO GERAL:

Em relação ao Modelo Geral de Interfaces de Aplicação proposto, pode-se concluir:

a - É necessário introduzir algumas funcionalidades à API definida pelo Projeto MAP/TOP, para que a API possa ser generalizada para os protocolos de aplicação e os padrões de formatos de dados existentes atualmente. Este trabalho propõe os seguintes blocos funcionais (funcionalidades):

- TAOCF: Controla a interação dos diversos ASEs a nível de Interface de Aplicação, quando o AP envia uma requisição ou resposta;

- RAOCF: Controla a interação dos diversos ASEs a nível de Interface de Aplicação, quando a camada de aplicação envia uma primitiva de indicação ou confirmação;

- IP: Processa serviços solicitados por primitivas de indicação aceitas, quando o AP autoriza a Interface de Aplicação para tal;

- VDRDB: Realiza o mapeamento entre os dispositivos virtuais e os dispositivos reais e vice-versa. Além disso, cria e manipula formatos de dados padrões.

Os blocos funcionais TAOCF e RAOCF executam a funcionalidade g citada anteriormente, enquanto que o IP realiza a funcionalidade e. O bloco funcional VDRDB executa a funcionalidade f. Uma grande vantagem do ambiente OSI/ISO é a criação e o uso de objetos virtuais. O VDRDB permite a manipulação de objetos virtuais a nível de Interface de Aplicação;

b - Interfaces de Aplicação mais simples podem ser construídas, utilizando o Modelo Geral proposto com a eliminação de alguns blocos funcionais e/ou algumas comunicações entre blocos, de acordo com os aspectos específicos de cada estação.

CONCLUSÕES RELATIVAS À PROPOSTA DE IMPLEMENTAÇÃO:

Em relação à proposta de implementação do Modelo Geral de Interfaces de Aplicação proposto, pode-se concluir:

a - Metodologias clássicas de desenvolvimento de "software" podem ser utilizadas para a construção de uma Interface de Aplicação;

b - As partes da Interface de Aplicação, que dependem do sistema operacional, são muito bem localizadas. Ao trocar o sistema operacional que dá suporte à Interface de Aplicação, as mudanças na Interface não são complexas;

c - A Interface de Aplicação pode ser implementada como uma única tarefa ou possuindo diversas tarefas. No segundo caso, uma

possibilidade é ter uma tarefa para cada bloco funcional do Modelo Geral. Neste último caso, a seção 1.3.1 comenta quantas instâncias devem ter alguns blocos funcionais (por exemplo, uma instância por AE ou uma instância por associação).

No caso da Interface de Aplicação ser implementada numa tarefa, pode existir uma única instância desta tarefa, uma instância por invocação de AE ou uma instância por associação. A primeira solução pode ser utilizada em estações simples, enquanto que a segunda é uma solução intermediária que, à primeira vista, parece ser adequada para a opção da Interface de Aplicação ter mais de uma instância;

d - Várias soluções conhecidas de sincronização e comunicação de processos podem ser utilizadas para a criação do ambiente multitarefa requerido pela Interface de Aplicação.

CONCLUSÕES RELATIVAS AO MODELO GERAL E À PROPOSTA DE IMPLEMENTAÇÃO:

Em relação ao Modelo Geral de Interfaces de Aplicação proposto e à proposta de implementação deste Modelo, pode-se concluir:

a - As Interfaces de Aplicação devem ser modeladas em função dos protocolos de aplicação utilizados e da complexidade das estações. De acordo com o protocolo de aplicação que se deseja implementar, a Interface de Aplicação deverá ou não implementar certos blocos funcionais, dependendo do tipo da estação, cliente ou servidora (modelo utilizado em muitos protocolos de aplicação) e das particularidades operacionais (por exemplo, se a Interface de Aplicação responde ou não automaticamente às requisições remotas). A complexidade da estação implica na implementação de todas, ou de parte das funções suportadas pela Interface de Aplicação (módulos de funções), de acordo com o tipo de aplicação que estas estações possuem.

A implementação do SISDI-MAP é realizada a partir da modelagem dos blocos funcionais e da existência de módulos de funções comentadas acima.

CONCLUSÕES RELATIVAS À IMPLEMENTAÇÃO NO SISDI-MAP:

Em relação à implementação da Interface de Aplicação no SISDI-MAP, pode-se concluir:

a - Cada bloco funcional do Modelo Geral possui código que:

- É igual para todas as funções da Biblioteca da Interface de Aplicação;
- É igual para um grupo de funções da Biblioteca da Interface de Aplicação;
- É particular a uma função da Biblioteca da Interface de Aplicação.

Este fato influi na estrutura dos procedimentos dos blocos funcionais, sejam os blocos tarefas ou procedimentos da tarefa Interface de Aplicação. O aproveitamento desta característica da Interface pode simplificar a sua implementação;

b - Pode-se elaborar uma estrutura de dados não muito complexa para a Interface de Aplicação. Apenas com tabelas (vetores) e filas (internas e de interface com os APs e a camada de aplicação) pode-se construir uma estrutura de dados adequada para a Interface;

c - Devido à localização dos blocos funcionais TAOCF e RAOCF introduzidos no Modelo Geral, para controlarem a interação entre os diversos ASEs a nível de Interface, estes blocos facilitam a interação entre a Interface e o usuário (AP), funcionando como um pequeno módulo de comunicação com o usuário;

d - Para algumas funções de baixo nível da Interface de Aplicação, pode-se modificar o esquema de alocação de memória para as mensagens, de tal maneira que não sejam necessárias duas alocações (realizadas pelo usuário (AP) e pela Interface). Para tal, é necessário que as diferenças entre os tipos de mensagens das interfaces API-AP e API-Camada de Aplicação sejam apenas questão de preâmbulo, e que no restante, as duas mensagens tenham os mesmos campos na mesma ordem;

e - Para estações simples que utilizam poucos recursos da Interface de Aplicação, e possuem sempre os mesmos APs que, por sua vez, são em pequeno número e confiáveis, a verificação da correção dos

parâmetros das funções da Interface pode ser diminuida. Isto simplifica a Interface de Aplicação.

CONCLUSÃO FINAL:

Desta tese, pode-se concluir:

a - Os sistemas de comunicação devem possuir Interfaces de Aplicação, ou algo similar, com a intenção de obterem interfaces amigáveis para os usuários, devido à complexidade das primitivas dos diversos protocolos de aplicação. Exemplos vistos foram: API do Projeto MAP/TOP, interface "socket" para a camada de transporte do sistema UNIX, Agente de Usuário nos Protocolos X-400 e de Serviço de Diretório e a Interface de Aplicação proposta nesta tese;

b - A complexidade da Interface de Aplicação depende das aplicações (APs). Em certos casos, os Processos Aplicativos podem requerer diversos serviços da Interface: de alto nível, confirmados, de espera de indicações, respondedores, de resposta automática, de mapeamento entre objetos virtuais e objetos reais, entre outros. Outras vêzes, os Processos Aplicativos podem requerer apenas o envio e o recebimento de mensagens genéricas.

No primeiro caso, a Interface de Aplicação pode ser construída de acordo com o Modelo Geral proposto na seção 1.4.5. No segundo caso, a Interface de Aplicação pode ser similar a um "socket". No caso intermediário, em que os Processos Aplicativos requerem somente alguns dos serviços citados acima, a Interface de Aplicação pode ser construída de acordo com o Modelo Geral, onde os blocos funcionais relacionados com os serviços desejados permanecem, e os outros blocos são suprimidos.

O Modelo de Apresentação de Interfaces de Aplicação proposto no capítulo 4 permite estas soluções intermediárias entre o Modelo Geral e a Interface do tipo "socket";

c - A Interface de Aplicação deve refletir a divisão de protocolos de aplicação mais complexos (por exemplo, os Protocolos X-400 e de Serviço de Diretório) em diversos SASEs. Os blocos funcionais TAOCF e RAOCF representam, na Interface de Aplicação, esta

característica da camada de aplicação. O Modelo de Apresentação permite esta divisão;

d - A Interface de Aplicação e o Protocolo ROSE têm funcionalidades comuns, mas aplicadas em níveis diferentes. Desta forma, se a Interface de Aplicação executar algumas destas funcionalidades comuns, a camada de aplicação é simplificada, e vice-versa. O Modelo de Apresentação estimula a utilização do Protocolo ROSE na camada de aplicação;

e - Para os Processos Aplicativos, pode ser adequada a existência de associações multipares, ou seja, associações entre um e diversos nós. A Interface de Aplicação pode incorporar este conceito, dependendo do protocolo de transporte utilizado. O Modelo de Apresentação mostra as operações de "binding" necessárias para a implementação de associações multipares, num ambiente em que os protocolos de aplicação podem estar divididos em diversos SASEs.

CONTRIBUIÇÕES DESTE TRABALHO:

Os pontos relevantes desta tese são:

a - Proposta do Modelo Geral de Interface de Aplicação (figura 1.28), construído para os Protocolos MMS, FTAM, TP e RDA, e para padrões gráficos, de definição de produtos e de documentos de escritório, sendo geral para aplicações típicas e atuais de Interfaces de Aplicação (seção 1.4) ([MAD90a], [MAD90b], [MAD90c] e [MAD90d]);

b - Análise comparativa entre o conceito de Interface de Aplicação (API) e os conceitos similares e/ou relacionados: "socket", Agente de Usuário, "binding" e Operação Remota (capítulo 4) [MAD90d];

c - Proposta do Modelo de Apresentação de Interface de Aplicação (figura 4.14), que define a visão que o AP tem dos serviços oferecidos, baseado na tendência atual dos sistemas de comunicação utilizarem os conceitos citados em b (capítulo 4);

d - Proposta das operações de "binding" de associações e invocações, para a utilização de diversos SASEs. Desta forma, pode-se ter não somente a comunicação par-a-par, mas também a comunicação multipar (capítulo 4);

e - Proposta de implementação de Interfaces de Aplicação, baseada na existência de módulos de funções e de blocos funcionais (capítulo 2) ([MAD88], [MAD90a] e [MAD90b]);

f - Análise de diversos ambientes multitarefas que podem suportar a Interface de Aplicação (capítulo 2);

g - Implementação de um subconjunto das funções da Interface de Aplicação para o Protocolo MMS no SISDI-MAP. Esta implementação é a parte prática deste trabalho e das dissertações [COR90a] e [SOU90]. A implementação é baseada em a, e e f (capítulo 3) ([PAG89], [COR90b], [MAD90a], [MAD90b], [MAD90c] e [MAD90d]);

h - Enquadramento do conceito de Interface de Aplicação nos conceitos da ISO (seção 1.2) [MAD90c].

SUGESTÕES

A seguir são apresentadas algumas sugestões para continuação deste trabalho.

a - A parte prática deste trabalho é a construção da API para o SISDI-MAP, que é um sistema didático. Portanto, para o SISDI-MAP, o tempo de resposta não é o aspecto fundamental. Contudo, para um sistema não didático é importante conhecer os tempos de resposta para os serviços oferecidos pela API;

b - A API no SISDI-MAP está construída e integrada apenas para um subconjunto de funções da Biblioteca, como foi visto anteriormente. Pode-se continuar este trabalho, apesar da implementação atual possuir pelo menos algumas funções de cada tipo;

c - O SISDI-MAP pode ser migrado para o ambiente do sistema operacional UNIX. Alguns comentários sobre a migração de uma API para o ambiente UNIX foram feitos na seção 2.5. Desta forma, pode-se comparar a eficiência da API no ambiente UNIX em relação ao ambiente DOS;

d - O SISDI-MAP possui apenas as camadas de apresentação e aplicação, além da Interface de Aplicação. A existência de várias estações é simulada num único nó. Seria interessante a implementação do SISDI-MAP em diversas estações com a comunicação real entre estas. Desta forma, o comportamento da API poderia ser analisado não apenas para um sistema didático. As estações poderiam ter diferentes níveis de complexidade. Dependendo da complexidade da estação, um ou mais módulos de funções da Biblioteca poderiam ser implementados.

Para obter a comunicação entre estações, necessita-se das outras camadas do Modelo de Referência OSI/ISO que o SISDI-MAP não implementa. Contudo, duas dissertações de mestrado ([INA90] e [RAU90]) apresentam e implementam subconjuntos das funções das camadas de apresentação e sessão, de maneira compatível com o SISDI-MAP. Se o SISDI-MAP migrar para o ambiente UNIX (ver c), incorporando as camadas de apresentação e sessão comentadas acima, o SISDI-MAP estará pronto

para comunicar com a camada de transporte, utilizando como sistema operacional o UNIX.

As redes locais internas da UNICAMP possuem estações de trabalho SUNs, que executam o sistema operacional UNIX, e utilizam os protocolos de transporte e rede TCP/IP ([POS81b] e [POS81a]). Implementado nestas redes locais, o SISDI-MAP poderá comunicar com a camada de transporte TCP. Contudo, o TCP apresenta algumas diferenças em relação à classe 4 do protocolo de transporte da ISO (classe mais completa) ([ISO8072] e [ISO8073]). Desta forma, é necessário a construção de uma interface entre a camada de sessão e o protocolo TCP/IP. Contudo, soluções de conversões entre os protocolos TCP e de transporte da ISO já foram apresentadas [TAN88];

e - A API, assim como os diversos protocolos do SISDI-MAP, pode ser implementada utilizando-se uma linguagem orientada a objetos. Alguns comentários sobre a utilização de linguagens orientadas a objetos na implementação de protocolos e interfaces de aplicação foram feitos na seção 2.4. Um estudo detalhado desta utilização pode ser realizado;

f - Por simplicidade, existe apenas uma instância da API no SISDI-MAP. Um estudo comparativo poderia ser realizado para outros casos: uma instância da API para cada invocação de Entidade de Aplicação e uma instância da API para cada associação.

A API no SISDI-MAP é implementada como uma única tarefa, também por simplicidade. Um estudo sobre a implementação da API possuindo diversas tarefas pode ser realizado. Por exemplo, cada bloco funcional do Modelo Geral proposto na figura 1.28 pode ser uma tarefa distinta;

g - Um tópico atual de pesquisa em sistemas de comunicação é o da comunicação multipar, em contrapartida com a comunicação par-a-par utilizada hoje. A seção 4.5 apresentou uma proposta de comunicação multipar, para os sistemas de comunicação existentes (protocolos de transporte existentes). Um estudo mais detalhado pode ser realizado;

h - Este trabalho analisou alguns conceitos, relacionados com o de API, existentes nos sistemas de comunicação atuais. Um estudo

com novos conceitos que poderão surgir, também relacionados com o de API, pode ser realizado.

TENDÊNCIAS

A seguir são apresentados alguns comentários sobre a tendência atual de utilização do conceito de Interface de Aplicação e de conceitos relacionados:

a - A tendência atual é dos sistemas de comunicação possuirem uma Interface de Aplicação, ou algo similar. Contudo, não se pode ter uma Interface de Aplicação muito robusta numa estação simples. Exemplo típico é a existência da API para o Protocolo MMS em estações simples, que utilizam poucas primitivas do MMS. Neste caso, a implementação do Protocolo MMS é simples (quase que apenas repassa primitivas entre a API e a camada inferior), mas a implementação da API pode ser custosa.

Devem ser formuladas soluções intermediárias para as Interfaces de Aplicação, em função dos diversos usos da interface.

Entretanto, as funcionalidades de uma Interface de Aplicação são desejadas mesmo em sistemas simples, e podem ser implementadas. Por exemplo, as estações mini-MAP (estações simples em ambientes da manufatura) [MAP87d] possuem as camadas física, de enlace e aplicação, além da API. Isto significa que para estas estações simples foram retiradas as funcionalidades das camadas de rede, transporte, sessão e apresentação, mas foram adicionadas as funcionalidades da Interface de Aplicação.

De certa forma, o Processo de Aplicação requer uma representação na camada de aplicação e vice-versa, devido às interações entre ambos. No ambiente OSI/ISO, o Elemento do Usuário e as Funções de Interação executam estas representações, respectivamente. Com a introdução da Interface de Aplicação, que contém Regiões do Programa de Usuário e do Provedor de Serviços, as representações citadas são absorvidas pela Interface de Aplicação;

b - Com a definição de protocolos de aplicação mais robustos (Serviço de Diretório, MHS, entre outros), nota-se a tendência de divisão de cada um destes protocolos em diversos SASEs. Além disto,

nota-se a tendência das aplicações, que requerem serviços de rede, utilizarem mais de um protocolo de aplicação específico. Por exemplo, a aplicação de Acesso às Bases de Dados Remotas pode requerer o uso dos SASEs RDA e TP, além dos CASEs ROSE, ACSE e CCR. Em ambos os casos, apesar da aplicação requerer diversos SASEs, a tendência é a utilização de uma única Interface de Aplicação;

c - Com a crescente utilização de Sistemas Distribuídos, novas aplicações, que requerem a comunicação de um nó com diversos nós, estão surgindo. Por exemplo, um AP pode desejar efetuar uma transação distribuída, ou seja, uma transação em que estejam envolvidos diversos outros nós. A tendência é não ter apenas associações par-a-par, mas também multipar, pela qual um nó está conectado com diversos nós através da definição de uma única associação.

d - Apesar da Interface de Aplicação aumentar a portabilidade dos Processos de Aplicação, e ajudar estes Processos em algumas tarefas que seriam, em princípio, de responsabilidade dos Processos, a Interface de Aplicação não resolve o problema da portabilidade na sua totalidade, e não realiza todas as tarefas que o Processo Aplicativo desejaria. Por exemplo, a migração de um AP escrito em Pascal para um ambiente de comunicação, que possui uma Interface de Aplicação escrita em C, pode necessitar a construção de uma interface entre o AP e a Interface de Aplicação, para colocar os parâmetros das chamadas das funções da Interface de Aplicação nas posições corretas. Uma tendência é introduzir novas funcionalidades na Interface de Aplicação, para aumentar a portabilidade dos APs e/ou para executar outras tarefas de interesse dos Processos Aplicativos;

e - Com o aparecimento de protocolos de aplicação mais robustos (Serviço de Diretório, MHS, RDA, entre outros), nota-se também a tendência de utilização do Protocolo ROSE na camada de aplicação. Desta forma, os diversos SASEs, que compõem um protocolo de aplicação, utilizam o ROSE tratando todos os seus serviços como Operações Remotas.

BIBLIOGRAFIA

[ATT89] AT&T
"UNIX System V Release 3.2 - Network Programmer´s Guide"
Prentice Hall, 1989

[BEV89] BEVER, M.
"OSI Application Layer - Entwicklungsstand und Tendenzen"
IBM European Networking Center, Technical Report No. 43.8901, 1989

[BOE86] BOEHM, B.
"A Spiral Model of Software Development and Enhancement"
Software Engineering Notes, Vol. 11, n. 4, 1986, pp. 22-42

[BOL87] BOLOGNESI, T. e BRINKSMA, E.
"Introduction to the ISO Specification Language LOTUS"
Computer Networks and ISDN Systems, Vol. 14, 1987, pp. 25-59

[BRI72] BRINCH HANSEN, P.
"Structured Multiprogramming"
Comm. ACM, 15, 7, pp. 574-578, Julho 1972

[BRI73] BRINCH HANSEN, P.
"Operating System Principles"
Prentice Hall, 1973

[BUD87] BUDKOWSKI, S. e DEMBINSKI, P.
"An Introduction to Estelle: A Specification Language for Distributed Systems"
Computer Networks and ISDN Systems, Vol. 14, 1987, pp. 3-23

[CCITT400] CCITT - Comité Consultatif International de Télégraphique

et Téléphonique
Recomendação X-400
"Message Handling System"
1988

[COR90a] CORRÊA, 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"
UNICAMP/FEE/DCA, Julho 1990

[COR90b] CORRÊA, J. N.; SOUSA, V. L. P., MADEIRA, E. R. M. e MENDES, M. J.
"Aspectos de Programação e Uso do MMS"
I Seminário sobre Redes de Comunicação Industrial - SOBRACON, São Paulo, Setembro 1990

[DIJ65] DIJKSTRA, E. W.
"Cooperating Sequential Process"
The Netherlands, 1965

[DIJ71] DIJKSTRA, E. W.
"Hierarchical Ordering of Sequential Processes"
Acta Inf. 1,2, pp. 115-138, 1971

[DWY87] DWYER, J. e Ioannou, A.
"MAP and TOP: Advanced Manufacturing Communications"
Korgan Page, 1987

[FER] FERNANDES, I. A.
Dissertação de Mestrado em andamento
UNICAMP/FEE/DCA

[HAR81] HARLAND, D. M.
"On Facilities for Interprocess Communication"

Information Processing Letters, Vol. 12, n. 5, Outubro 1981
pp. 221-226

[HEN90] HENSHALL, J. e SHAW, S.
"OSI Explained: End-to-end Computer Communication Standards"
Second Edition, Ellis Horwood, 1990

[HOA72] HOARE, C. A. R.
"Towards a Theory of Parallel Programming"
Operating System Techniques, Academic Press, N. Y., 1972

[HOA78] HOARE, C. A. R.
"Communicating Sequential Processes"
Comm. ACM, Vol. 21, n. 8, Agosto 1978, pp. 666-677

[ICH79] ICHBIAH, J. D.
"Preliminary ADA Reference Manual"
Sigplan Notices, Vol. 14, n. 6, Junho 1979

[INA90] INAZUMI, P. C. M.
"Aspectos de Especificação e Implementação da Camada de Apresentação do Padrão MAP utilizando o ambiente EPOS"
UNICAMP/FEE/DCA, Dezembro 1990

[ISO1494] ISO/TC97/SC21 N1494
"Application Layer Structure"
Setembro 1986

[ISO7498] ISO IS 7498
"Information Processing Systems - Open Systems Interconnection - Basic Reference Model"
1984

[ISO8072] ISO DIS 8072
"Information Processing Systems - Open Systems Inter-

connection - Transport Service Definition"

[ISO8073] ISO DIS 8073
"Information Processing Systems - Open Systems Interconnection - Transport Protocol Specification"

[ISO8571] ISO DIS 8571
"Information Processing Systems - Open Systems Interconnection - "File Transfer, Access and Management"
Parte 1: "General Introduction"
Parte 2: "The Virtual Filestore"
Parte 3: "File Service Definition"
Parte 4: "File Protocol Specification"

[ISO8649] ISO IS 8649
"Information Processing Systems - Open Systems Interconnection - Association Control: Service Definition"
1988

[ISO8650] ISO IS 8650
"Information Processing Systems - Open Systems Interconnection - Association Control: Protocol Specification"
1988

[ISO8802] ISO 8802.2 (idêntico ao IEEE 802.2)
"Local Area Networks - Logical Link Control"

[ISO9066] ISO DIS 9066
"Information Processing Systems - Text Communication - "Reliable Transfer"
Parte 1: "Model and Service Definition"
Parte 2: "Protocol Specification"

[ISO9072] ISO DIS 9072
"Information Processing Systems - Text Communication -

Remote Operations"
Parte 1: "Model, Notation and Service Definition"
Parte 2: "Protocol Specification"
1987

[ISO9506a] ISO DP 9506 - Parte 1
"Manufacturing Message Specification - Service Specification"
Maio, 1987

[ISO9506b] ISO DP 9506 - Parte 2
"Manufacturing Message Specification - Protocol Specification"
Maio, 1987

[ISO9545] ISO DIS 9545
"Information Processing Systems - Open Systems Interconnection - Application Layer Structure"
1989

[ISO9579] ISO DP 9579
"Information Processing Systems - Open Systems Interconnection - Remote Database Access Service and Protocol"

[ISO9594] ISO DIS 9594
"Information Processing Systems - The Directory"

[ISO9804] ISO DIS 9804.2
"Information Processing Systems - Open Systems Interconnection - Commitment, Concurrency and Recovery Service Element: Service Definition"
1988

[ISO9805] ISO DIS 9805.2
"Information Processing Systems - Open Systems Inter-

connection - Commitment, Concurrency and Recovery Service Element: Protocol Specification"
1988

[ISO10021] ISO IS 10021
"Information Processing Systems - Text Communication - MOTIS: Message Oriented Text Interchange Systems"

[ISO10026] ISO DP 10026
"Information Processing Systems - Open Systems Interconnection - "Distributed Transaction Processing"
Parte 1: "Model"
Parte 2: "Service Definition"
Parte 3: "Protocol Specification"
1988

[JAC89] JACINTHO, D. C. A.
"Aspectos de Especificação e Implementação da Estrutura de Mensagens de um Sistema Didático do Protocolo MMS"
UNICAMP/FEE/DCA, Dezembro 1989

[JOH90] JOHANNSEN, W. e LAMERSDORF, W.
"An Open Systems Architecture for Transaction Supported Distributed Database Applications"
IBM Deutschland, European Networking Center, 1990

[LIM] LIMA, J. M. S.
Dissertação de Mestrado em andamento
UNICAMP/FEE/DCA

[MAD85] MADEIRA, E. R. M.
"Implementação do Protocolo X-25 num Concentrador de Comunicações baseado no 8088"
Dissertação de Mestrado, UNICAMP/IMECC/DCC, Maio 1985

[MAD88] MADEIRA, E. R. M. e MENDES, M. J.
"Interface de Programas de Aplicação para o Protocolo MMS (RS-511)"
III CONAI - Congresso Nacional de Automação Industrial, São Paulo, Setembro 1988

[MAD90a] MADEIRA, E. R. M.; CORRÊA, J. N.; SOUSA, V. L. P. e MENDES, M. J.
"Modelagem de Interfaces de Aplicação e Exemplo de Implementação para o Protocolo MMS"
8o Simpósio Brasileiro de Redes de Computadores, Campinas, Abril 1990

[MAD90b] MADEIRA, E. R. M. e MENDES, M. J.
"Application Interface Model for Communication Software and an MMS Protocol Implementation Example"
Colloque International - CIM90 - Productique et Integrations, Bordeaux, França, Junho 1990

[MAD90c] MADEIRA, E. R. M. e MENDES, M. J.
"An Application Interface Model for Communication Software"
IEEE Global Telecommunications Conference - GLOBECOM'90
San Diego, Estados Unidos, Dezembro 1990

[MAD90d] MADEIRA, E. R. M.
"An Application Interface Model for Communication Systems"
Notas de Apresentação - GMD/FOKUS
Berlim, Alemanha, Dezembro 1990

[MAP87a] MAP/TOP
"MMS Application Interface Specification"
Março 1987

[MAP87b] MAP/TOP
"TOP: Technical Office Protocols"

Versão 3.0 - "Implementation Release-Subject to Errata Changes", Abril 1987

[MAP87c] MAP/TOP
"MAP: Manufacturing Automation Protocol"
Versão 3.0, Julho 1987

[MAP87d] MAP/TOP
MAP 3.0 - Capítulo 13: "The MAP Enhanced Performance Architecture"
Julho 1987

[MAP88a] MAP/TOP
"FTAM Application Interface Specification"
Abril 1988

[MAP88b] MAP/TOP
"Application Interface Model and Specification Requirements"
Junho 1988

[MAP88c] MAP/TOP
"Connection Management Interface Specification"
Junho 1988

[MAP88d] MAP/TOP
"Application Interface Support Functions"
Junho 1988

[MAP88e] MAP/TOP
"MMS Application Interface Specification"
Junho 1988

[MAP88f] MAP/TOP
"Private Communications Application Interface Specificaton"
Junho 1988

[MCG88] MCGUFFIN, L. J.; REID, L. O. e SPARKS, S. R.
"MAP/TOP in CIM Distributed Computing"
IEEE Network, Vol. 2, n. 3, Maio 1988, pp. 23-31

[MEL86a] MELENDEZ, W. A. e PETERSEN, E. L.
"The Upper Layers of the ISO/OSI Reference Model (Part I)"
Computer Standards & Interfaces, Vol. 5, 1986, pp. 13-46

[MEL86b] MELENDEZ, W. A. e PETERSEN, E. L.
"The Upper Layers of the ISO/OSI Reference Model (Part II)"
Computer Standards & Interfaces, Vol. 5, 1986, pp. 65-77

[MEN89] MENDES, M. J.
"Comunicação Fabril e o Projeto MAP/TOP"
IV EBAI, Santiago Del Estero, Argentina, Janeiro 1989

[MOR88] MORGAN, E.
"Through MAP/TOP to CIM"
Moore & Matthes, 1988

[NEH90] NEHMER, J. e MATTERN, F.
"Service Modeling in Distributed Operating Systems"
Proceedings 2nd IEEE Workshop on Future Trends of Distributed Computing Systems, Cairo, Egito, Setembro 1990, pp. 215-221

[PAG89] PAGLIONI, A. J.; JACINTHO, D. C. A.; MADEIRA, E. R. M.; FERNANDES, I. A.; CORRÊA, J. N.; LIMA, J. M. S.; ZABEU, M. C.; SOUSA, V. L. P. e MENDES, M. J.
"SISDI-MAP: Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP"
Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos, Florianópolis - SC, Setembro 1989

[PAG91] PAGLIONI, A. J.

"Aspectos de Especificação e Implementação da Estrutura de Mensagens da Máquina de Protocolo ACSE para o SISDI-MAP"
UNICAMP/FEE/DCA, Junho 1991

[PAP86] PAPPE, S.; EFFELSBERG, W e LAMERSDORF, W.
"Database Access in Open Systems"
IBM Deutschland, European Networking Center, 1986

[POS81a] POSTEL, J.
"Internet Protocol"
RFC 791, Setembro 1981

[POS81b] POSTEL, J.
"Transmission Control Protocol"
RFC 793, Setembro 1981

[RAU90] RAUL MENDEZ, C.
"Aspectos de Implementação do Protocolo de Sessão utilizando Ferramentas CASE"
UNICAMP/FEE/DCA, Dezembro 1990

[ROY70] ROYCE, W. W.
"Managing the Development of Large Software Systems - Concepts and Techniques"
Proceedings WESCON, Agosto 1970

[RUP89] RUPPRECHT, G.
"OSI-Produkte: Einheitliche Benutzerschnittstelle"
Computer Magazin, Outubro 1989, pp. 50-53

[SOU90] SOUSA, V. L. P.
"Aspectos de Especificação e Implementação da Interface de Aplicação para o MMS: Serviços de Acesso a Variáveis"
UNICAMP/FEE/DCA, Julho 1990

[STA87] STACY, A. H.
"The MAP Book: An Introduction to Industrial Networking"
Industrial Network Incorporated, 1987

[STE90] STEVENS, W. R.
"UNIX Network Programming"
Prentice Hall, 1990

[SVO89] SVOBODOVA, L.
"Implementing OSI Systems"
IEEE Journal on Selected Areas in Communications, Vol. 7, n. 7, Setembro 1989, pp. 1115-1130

[TAK90] TAKAHASHI, T. e LIESENBERG, H. K. E.
"Programação Orientada a Objetos"
VII Escola de Computação, São Paulo, 1990

[TAN88] TANENBAUM, A. S.
"Computer Networks"
Second Edition, Prentice Hall, 1988

[TSC90] TSCHAMMER, V.; WOLISZ, A. e HALL, J.
"Support for Cooperation and Coherence in an Open Service Environment"
Proceedings 2nd IEEE Workshop on Future Trends of Distributed Computing Systems, Cairo, Egito, Setembro 1990, pp. 222-228

[WEB90] WEBER, H. e WOSNITZA, H.
"Protocol Engineering for Application Layer Protocols"
Proceedings 2nd IEEE Workshop on Future Trends of Distributed Computing Systems, Cairo, Egito, Setembro 1990, pp. 315-324

[WEG79] WEGNER, P.
"Programming with ADA: An Introduction by Means of Graduated Examples"

Sigplan Notices, Vol. 14, n. 12, Dezembro 1979

[WIR77] WIRTH, N.
"Modula: A Language for Modular Programming"
Software Practice and Experience, 7, 1, pp. 3-35,
Janeiro 1977

[WIR80] WIRTH, N.
"Modula 2"
Relatório n. 36, Institut fur Informatic, 1980

[WOL90] WOLISZ, A. e TSCHAMMER, V.
"Service Provider Selection in an Open Services Environment"
Proceedings 2nd IEEE Workshop on Future Trends of Distributed
Computing Systems, Cairo, Egito, Setembro 1990, pp. 229-235

[ZAB89] ZABEU, M. C.
"Um Modelo para Configuração de Núcleos para Tempo Real"
UNICAMP/FEE/DCA, Dezembro 1989

APÊNDICE A

PROJETO MAP/TOP E PROTOCOLO MMS

PROJETO MAP/TOP:

O Projeto MAP ("Manufacturing Automation Protocol") foi criado pela General Motors em 1980, com o objetivo de definir mecanismos e procedimentos de interconexão de equipamentos programáveis para a automação industrial. A meta era facilitar a interconexão das "ilhas de automação" existentes, para construir sistemas integrados da manufatura.

O Projeto TOP ("Technical and Office Protocol") foi criado pela Boeing, e tem objetivos semelhantes aos do MAP, mas voltados para a automação de escritórios. Os primeiros esforços da Boeing datam do início da década de 80.

Uma promoção conjunta da General Motors com a Boeing na AUTOFACT em 1985, consistindo na exposição dos Projetos MAP e TOP, uniu os dois projetos num só, denominado MAP/TOP. O Projeto MAP/TOP segue o Modelo OSI da ISO, e define os protocolos a serem adotados para cada uma das camadas. Para a camada de aplicação, o Projeto MAP define o uso dos seguintes protocolos: DS (Serviço de Diretório), FTAM, NM ("Network Management") e MMS. O Projeto TOP define: FTAM, VT ("Virtual Terminal"), MHS (Correio Eletrônico), NM e DS. Maiores detalhes sobre o Projeto MAP/TOP podem ser encontrados em [MAP87c], [MAP87b], [MEN89], [STA87], [DWY87] e [MOR88].

PROTOCOLO MMS:

No Projeto MAP, o componente principal é o Protocolo MMS, que suporta a nível de camada de aplicação a transmissão de mensagens entre equipamentos programáveis, num ambiente de manufatura e controle de processos. Devido ao fato dos equipamentos programáveis num

ambiente fabril serem geralmente simples e especializados, o MMS fornece uma maneira relativamente simples, para estes equipamentos se comunicarem.

Os serviços MMS são organizados em unidades funcionais. Cada unidade não apresenta muitas primitivas. Porém, o MMS define várias unidades funcionais, e portanto, o MMS possui várias primitivas. Os parâmetros das diversas primitivas podem ser simples ou complexos, dependendo muitas vêzes do desejo do usuário do MMS. Isto ocorre devido à existência de parâmetros obrigatórios e opcionais, e devido à existência de parâmetros do tipo registro de tamanho variável, que podem ser de tipos simples ou de tipos estruturados complexos.

O MMS modela os equipamentos programáveis através de VMDs - Dispositivos Virtuais da Manufatura. Os VMDs tratam, de uma forma abstrata, dos elementos estruturais e dos objetos, que compõem os dispositivos reais. Cada VMD contém pelo menos um domínio. Um domínio representa um subconjunto de recursos alocados, de forma estática ou dinâmica (por exemplo: memória, entrada/saída, funções específicas de controle, entre outros). Um VMD, além de um ou mais domínios, possui uma função executiva, estações de operador e uma memória virtual de arquivo, como pode ser visto na figura A.1. Esta figura apresenta a

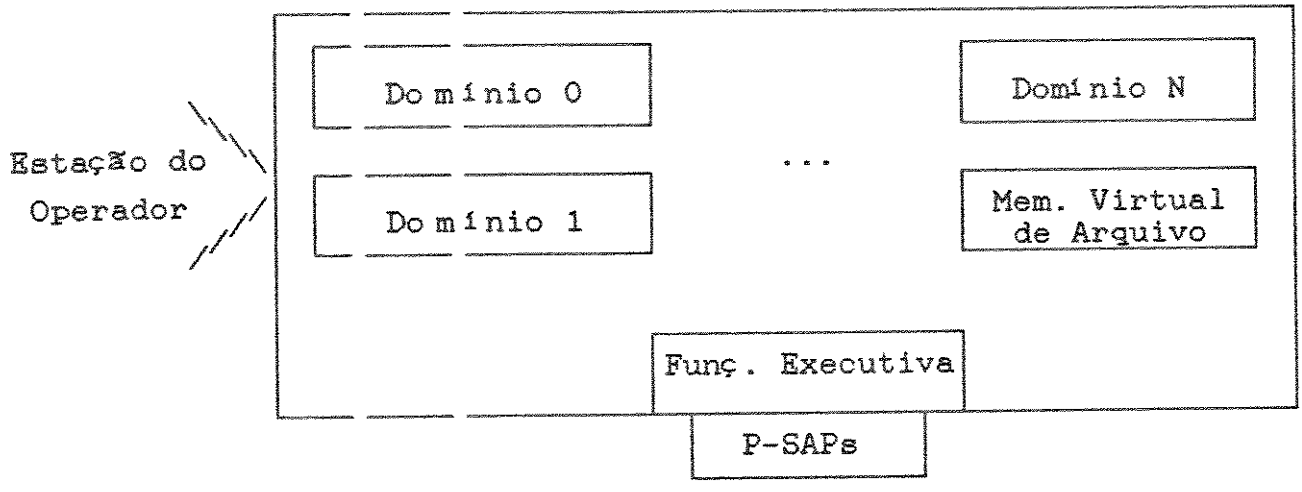


Figura A.1 - Modelo de VMD

estrutura do VMD. Os dois últimos conceitos têm funções óbvias, e a função executiva administra o acesso aos recursos do VMD.

Na implementação de um nó servidor do MMS é necessário providenciar um mapeamento do Dispositivo Virtual da Manufatura (VMD) para os aspectos funcionais do DispositivoReal da Manufatura e vice-versa.

As unidades funcionais definidas no MMS são: Gerenciamento de Contexto, Suporte de VMD, Gerenciamento de Domínios, Gerênciamento de Invocação de Programa, Acesso às Variáveis, Gerenciamento de Semáforos, Comunicação com Operador, Gerenciamento de Eventos, Gerenciamento de Jornal e Gerenciamento de Arquivos.

Maiores detalhes sobre o Protocolo MMS podem ser encontrados na sua especificação ([ISO9506a] e [ISO9506b]), e também em [MEN89].

APÊNDICE B

PRIMITIVAS DE SERVIÇO

As camadas do Modelo RM - OSI/ISO (Modelo de Referência para Interconexão de Sistemas Abertos da ISO) são compostas por Entidades. Uma Entidade comunica com o seu usuário (com o seu provedor), ou seja, com uma Entidade da camada superior (inferior), através de primitivas. Uma Entidade comunica com a Entidade par (pertencente a uma mesma camada) na estação remota, através de PDUs (Unidades de Dados do Protocolo). Existem quatro tipos de primitivas e PDUs: de requisição, resposta, indicação e confirmação (Figura B.1).

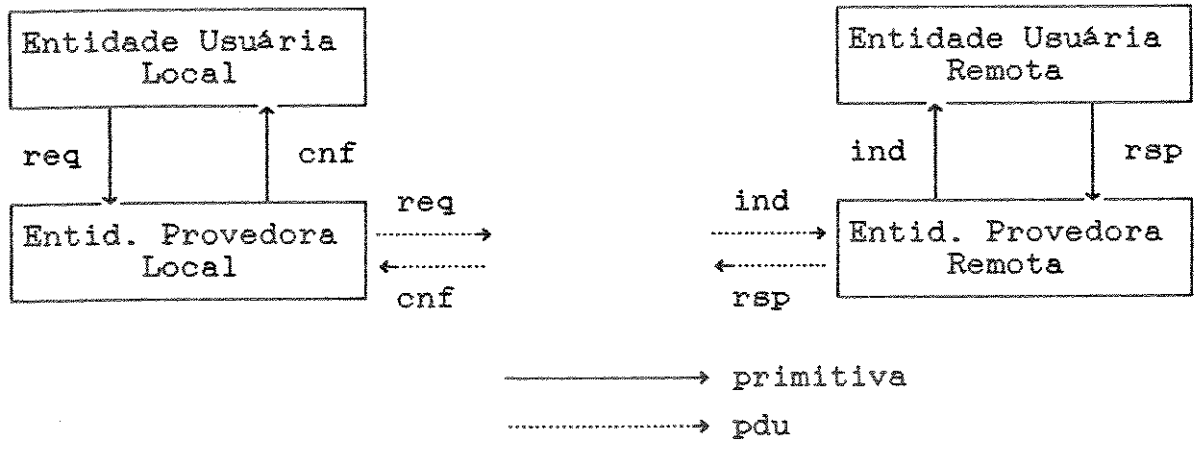


Figura B.1 - Troca de Primitivas e PDUs entre Entidades

Uma Entidade envia uma primitiva de requisição (PDU de requisição), quando deseja solicitar um serviço da Entidade Provedora (Entidade par).

Uma Entidade recebe uma primitiva de indicação (PDU de indicação), para ser informada sobre um evento da Entidade Provedora (Entidade par).

Uma Entidade envia uma primitiva de resposta (PDU de resposta)

para a Entidade Provedora (Entidade par), para responder a um evento.

Uma Entidade recebe uma primitiva de confirmação (PDU de confirmação) da Entidade Provedora (Entidade par), para ser informada sobre a sua requisição de serviço.

Serviços confirmados possuem primitivas (PDUs) de requisição, indicação, resposta e confirmação. Serviços não confirmados possuem primitivas (PDUs) de requisição e indicação.

Primitivas (PDUs) de resposta e confirmação podem ser positivas ou negativas.

APÊNDICE C

API PARA O PROTOCOLO FTAM

A API para o Protocolo FTAM é especificada pelo Projeto MAP/TOP [MAP88a], e apresenta serviços associados aos serviços definidos na parte 3 do padrão FTAM [ISO8571], além de serviços similares aos tipicamente disponíveis num sistema local. Os primeiros são executados pelas funções de baixo nível da API, enquanto que os últimos são executados pelas funções de alto nível da API. [MAP88a] define a API apenas para estações clientes que enviam requisições, e não recebem indicações.

A API para o FTAM utiliza os serviços de gerenciamento de contexto apresentados em [MAP88c], e define parâmetros e processamento específicos, relacionados ao FTAM, para as funções da Tabela C-I. As funções da Tabela C-II também são utilizadas pela API do FTAM, mas estas funções não têm parâmetros específicos do FTAM, e não realizam processamento específico relacionado ao FTAM.

As funções de gerenciamento de contexto STOP LISTEN, LISTEN, ANSWER e RELEASE RESPONSE não são utilizadas pela API do FTAM, pois estas funções são para uso de estações servidoras.

ft_connect
ft_rrequest
ft_abort
ft_ireceive

Tabela C-I - Funções de Gerenciamento de Contexto com Parâmetros Específicos do FTAM

ft_aeactivation
ft_aedeactivation

Tabela C-II - Funções de Gerenciamento de Contexto sem Parâmetros Específicos do FTAM

A API possui funções livres de contexto e sensíveis ao contexto. As funções sensíveis ao contexto requerem conhecimento da máquina de protocolo do FTAM, pois num dado instante, dependendo do estado da máquina, pode-se ou não requisitar um serviço específico. As funções livres de contexto não requerem nenhum conhecimento especial da máquina de protocolo, pois realizam uma ação autônoma e completa.

Os serviços livres de contexto são de alto nível, pois uma única função da API invoca várias primitivas do FTAM. Um exemplo de função livre de contexto é a função FILE COPY, que copia um arquivo de um sistema de arquivos em outro sistema de arquivos sem que seja preciso chamar nenhuma outra função da API. A chamada da função FILE COPY pode ser feita de um terceiro nó da rede, diferente dos nós fonte e destino da cópia do arquivo. A Tabela C-III apresenta as funções livres de contexto da API.

ft_fcopy	File Copy
ft_fmove	File Move
ft_fdelete	File Delete
ft_frattributes	File Read Attributes
ft_fcattributes	File Change Attributes

Tabela C-III - Funções de Alto Nível Livres de Contexto

A função FILE COPY da API deve invocar as seguintes primitivas do Protocolo FTAM, para uma operação realizada com sucesso:

a - F_INITIALIZE.req: para estabelecer a primeira conexão

com o nó fonte (também é invocada a primitiva A_ASSOCIATE.req do ACSE);

b - F_INITIALIZE.req: para estabelecer a segunda conexão com o nó destino (também é invocada a primitiva A_ASSOCIATE.req do ACSE);

c - F_BEGIN_GROUP.req, F_SELECT.req, F_READ_ATTRIB.req, F_OPEN.req e F_END_GROUP.req: utilizando a primeira conexão, para abrir o arquivo fonte;

d - F_BEGIN_GROUP.req, F_CREATE.req, F_OPEN.req e F_END_GROUP.req: utilizando a segunda conexão, para abrir o arquivo destino;

e - F_WRITE.req: utilizando a segunda conexão, para iniciar a escrita do arquivo destino;

f - F_READ.req: utilizando a primeira conexão, para iniciar a leitura do arquivo fonte;

g - F_DATA.inds: através da primeira conexão, para receber os dados que compõem o arquivo fonte;

h - F_DATA.reqs: utilizando a segunda conexão, para enviar os dados que irão compor o arquivo destino;

i - F_DATA_END.ind: através da primeira conexão, para receber a indicação de final da seqüência de dados que compõe o arquivo fonte;

j - F_DATA_END.req: utilizando a segunda conexão, para indicar o final da seqüência de dados que irá compor o arquivo destino;

k - F_TRANSFER_END.req: utilizando a primeira conexão, para indicar o término da fase de transferência de dados;

l - F_TRANSFER_END.req: utilizando a segunda conexão, para indicar o término da fase de transferência de dados;

m - F_BEGIN_GROUP.req, F_CLOSE.req, F_CHANGE_ATTRIB.req, F_DESELECT.req e F_END_GROUP.req: utilizando a segunda conexão, para terminar o uso do arquivo destino;

n - F_BEGIN_GROUP.req, F_CLOSE.req, F_DESELECT.req e F_END_GROUP.req: utilizando a primeira conexão, para terminar o uso do arquivo fonte;

o - F_TERMINATE.req: para terminar a segunda conexão (também é invocada a primitiva A_RELEASE.req do ACSE);

p - F_TERMINATE.req: para terminar a primeira conexão (também é invocada a primitiva A_RELEASE.req do ACSE).

A função FILE COPY da API tem como um dos parâmetros de entrada o AE_LABEL da invocação da Entidade de Aplicação a ser utilizada. Dois valores são possíveis:

a - Um valor de AE_LABEL já existente. Neste caso, o AP chamou anteriormente a função FT_AEACTIVATION da API;

b - O valor NULL. Neste caso, a API gera um AE_LABEL para definir uma invocação a ser utilizada apenas para este serviço de FILE COPY.

Pode-se ver pelo exemplo, que as funções livres de contexto da API para o FTAM são bastante complexas. Contudo, estas funções oferecem serviços aos APs de uma maneira amigável, pois toda a complexidade da seqüência de invocações das diversas primitivas do FTAM é realizada de modo transparente aos APs.

Os serviços sensíveis ao contexto podem ser de baixo ou alto nível, dependendo de quantas primitivas do FTAM são invocadas por função da API, se apenas uma ou várias, respectivamente.

Um exemplo de função de baixo nível é a função READ da API, que invoca unicamente a primitiva F-READ.req do FTAM. Um exemplo de função de alto nível é a função FILE OPEN da API, que invoca as primitivas F-SELECT.req (ou F-CREATE.req, dependendo do parâmetro de estado do arquivo) e F-OPEN.req do FTAM. A Tabela C-IV mostra as funções de alto nível sensíveis ao contexto da API, e a Tabela C-V mostra as funções de baixo nível sensíveis ao contexto da API.

ft_fopen	File Open
ft_fclose	File Close

Tabela C-IV - Funções de Alto Nível Sensíveis ao Contexto

ft_select	Select
ft_deselect	Deselect
ft_open	Open
ft_close	Close
ft_create	Create
ft_delete	Delete
ft_rattribute	Read Attribute
ft_cattribute	Change Attribute
ft_read	Read
ft_write	Write
ft_erase	Erase
ft_locate	Locate
ft_sdata	Send Data
ft_rdata	Receive Data
ft_edata	Data End
ft_etransfer	Transfer End
ft_bgroup	Begin Group
ft_egroup	End Group
ft_cancel	Cancel
ft_rcancel	Cancel Response

Tabela 2-V - Funções de Baixo Nível Sensíveis ao Contexto

APÊNDICE D

API PARA COMUNICAÇÃO PRIVADA

A API para comunicação privada é especificada pelo Projeto MAP/TOP [MAP88f], e provê aos APs acesso direto ao ACSE e aos serviços da camada de apresentação definidos pelo MAP. Desta maneira, a API suporta a comunicação entre Entidades de Aplicação que trocam informação, utilizando o contexto de aplicação "Private".

A API utiliza os serviços de gerenciamento de contexto apresentados em [MAP88c], e define parâmetros específicos para as funções CONNECT, LISTEN, ANSWER, RELEASE REQUEST, RELEASE RESPONSE, ABORT e INDICATION RECEIVE. Esta última função recebe, além das indicações de A_RELEASE, A_ABORT e P_ABORT (relativas ao ACSE), indicações de P_DATAs (relativas à camada de apresentação).

Pode ser necessária mais de uma função INDICATION RECEIVE, para o AP receber informação associada com um único P_DATA.ind da camada de apresentação. Um parâmetro do tipo booleano compõe, junto com o parâmetro de dado do usuário, a estrutura referente à indicação de P_DATA na função INDICATION RECEIVE. O parâmetro do tipo booleano informa se este dado do usuário é ou não o último dado associado ao P_DATA.

A API define também o serviço de transferência de dados privados, que pode ser utilizado depois do estabelecimento da conexão, e permite o acesso direto à camada de apresentação. Este seviço é oferecido por uma única função não confirmada de baixo nível, que invoca a primitiva P_DATA.req da camada de apresentação.

Pode ser que o AP necessite chamar mais do que uma vez a função de transferência de dados, para que a API invoque um único P_DATA.req da camada de apresentação. O P_DATA.req conterá todos os campos de dados do usuário de todas as funções chamadas (caso inverso ao que foi descrito para a função INDICATION RECEIVE). A função de transferência de dados também possui um parâmetro do tipo booleano, para informar à API, se o parâmetro de dados do usuário é ou não o último, que irá

compor o P_DATA.req associado.

APÊNDICE E

FUNÇÕES DA API PARA O PROTOCOLO MMS

Este apêndice apresenta as funções da API do Projeto MAP/TOP [MAP88e] para o Protocolo MMS. As funções da unidade funcional de Gerenciamento de Contexto foram comentadas na seção 3.2.1. As Tabelas de E-I a E-IX apresentam as funções para cada uma das demais unidades: Suporte aos VMDs (Dispositivos Virtuais da Manufatura), Gerenciamento de Domínios, Gerenciamento de Invocação de Programa, Acesso às Variáveis, Gerenciamento de Semáforos, Comunicação com o Operador, Gerenciamento de Eventos, Gerenciamento de Jornal e Gerenciamento de Arquivos, respectivamente. A Tabela E-X apresenta as funções auxiliares.

mm_status	Status
mm_gnlist	Get Name List
mm_identify	Identify
mm_gclist	Get Capability List

Tabela E-I - Serviços de Suporte aos VMDs

mm_idsequence	Initiate Download Sequence
mm_dsresponse	Download Segment Response
mm_tdresponse	Terminate Download Response
mm_iusequence	Initiate Upload Sequence
mm_usegment	Upload Segment
mm_tusequence	Terminate Upload Sequence
mm_ddresponse	Domain Download Response
mm_duresponse	Domain Upload Response
mm_ldcontent	Load Domain Content
mm_sdcontent	Store Domain Content
mm_ddelete	Domain Delete
mm_gdattribute	Get Domain Attribute
mm_fupload	File Upload
mm_fdownload	File Download

Tabela E-II - Serviços de Gerenciamento de Domínios

mm_cpinvocation	Create Program Invocation
mm_pidelete	Program Invocation Delete
mm_strt	Start
mm_stop	Stop
mm_resume	Resume
mm_rst	Reset
mm_kill	Kill
mm_gpinvocation	Get Program Invocation

Tabela E-III - Serviços de Gerenciamento de Invocação de Programa

mm_read	Read
mm_rresponse	Read Response
mm_write	Write
mm_wresponse	Write Response
mm_rinformation	Report Information
mm_gaattribute	Get Access Attribute
mm_glvariable	Get Local Variable
mm_gvresponse	Get Varible Response
mm_dvariable	Define Variable
mm_dlvariable	Define Local Variable
mm_dvresponse	Define Variable Response
mm_glscattered	Get Local Scattered
mm_lvdelete	Local Variable Delete
mm_dllist	Define Local List
mm_gllist	Get Local List
mm_lldelete	Local List Delete
mm_dltype	Define Local Type
mm_gltype	Get Local Type
mm_ltdelete	Local Type Delete

Tabela E-IV - Serviços de Acesso à Variável

mm_tcontrol	Take Control
mm_ltcontrol	Local Take Control
mm_tcresponse	Take Control Response
mm_sdequeue	Semaphore Dequeue
mm_rcontrol	Relinquish Control
mm_lrcontrol	Local Relinquish Control
mm_rcresponse	Relinquish Control Response
mm_dlsemaphore	Define Local Semaphore
mm_lsdelete	Local Semaphore Delete
mm_rsstatus	Report Semaphore Status
mm_lsstatus	Local Semaphore Status
mm_ssresponse	Semaphore Status Response
mm_rpsemaphore	Report Pool Semaphore
mm_lpstatus	Local Pool Status
mm_psresponse	Pool Status Response
mm_rsentry	Report Semaphore Entry
mm_lsentry	Local Semaphore Entry
mm_seresponse	Semaphore Entry Response

Tabela E-V - Serviços de Gerenciamento de Semáforo

mm_input	Input
mm_output	Output

Tabela E-VI - Serviços de Comunicação com o Operador

mm_gasummary	Get Alarm Summary
mm_aenotification	Acknowledge Event Notification
mm_gecondition	Get Event Condition
mm_recondition	Report Event Condition

Tabela E-VII - Serviços de Gerenciamento de Eventos

mm_wjournal	Write Journal
mm_rjournal	Read Journal
mm_ijournal	Initialize Journal
mm_cjournal	Create Journal
mm_jdelete	Journal Delete
mm_rjstatus	Report Journal Status

Tabela E-VIII - Serviços de Gerenciamento de Jornal

mm_fopen	File Open
mm_fread	File Read
mm_fclose	File Close
mm_frname	File Rename
mm_fdelete	File Delete
mm_fdirectory	File Directory
mm_ofile	Obtain File
mm_fget	File Get

Tabela E-IX - Serviços de Gerenciamento de Arquivos

mm_mlist	Make List
mm_nmtype	New MMS Type
mm_nscomponent	New Structure Component
mm_naaccess	New Alternate Access
mm_nvspecification	New Variable Specification
mm_nsaccess	New Scattered Access
mm_nlmapping	New Local Mapping
mm_nnrepresentation	New Network Representation
mm_nvassociation	New Variable Association
mm_ndata	New Data
mm_necontent	New Entry Content
mm_njvariable	New Journal Variable
mm_ndcapability	New Domain Capability
mm_vget	V-Get
mm_vput	V-Put
mm_leextract	List Element Extract
mm_imtype	Interpret MMS Type
mm_iscomponent	Interpret Structure Component
mm_iaaccess	Interpret Alternate Access
mm_ivspecification	Interpret Variable Specification
mm_ivassociation	Interpret Variable Association
mm_idrepresentation	Interpret Data Representation
mm_isaccess	Interpret Scattered Access
mm_ijentry	Interpret Journal Entry
mm_ijvariable	Interpret Journal Variable
mm_iasummary	Interpret Alarm Summary
mm_dhandle	Duplicate Handle
mm_fhandle	Free Handle

Tabela E-X - Funções Livre de Contexto