Ernandes Lopes Bezerra

Aspectos da Implantação do Protocolo CMIS/CMIP em Ambiente ISODE

Dissertação submetida ao corpo docente da Coordenação dos Cursos de Pós-Gradução em Engenharia Elétrica da Universidade Federal da Paraíba - Campus II como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

Joberto Sérgio Barbosa Martins, Dr.-Ing., UFPB
Orientador

Campina Grande, Paraíba, Brasil

©Ernandes Lopes Bezerra, 1994

B574a Bezerra, Ernandes Lopes.
Aspectos da implantação do protocolo CMIS/CMIP em ambiente ISODE / Ernandes Lopes Bezerra. - Campina Grande, 1994.
128 f.

Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1994.
"Orientação : Prof. Dr. Joberto Sérgio Barbosa Martins".
Referências.

1. Engenharia Elétrica. 2. Redes de Computadores. 3. Protocolo CMIS/CMIP. 4. Ambiente ISODE. 5. Dissertação - Engenharia Elétrica. I. Martins, Joberto Sérgio Barbosa. II. Universidade Federal da Paraíba - Campina Grande (PB). III. Título

CDU 621:004.7(043)

Aspectos da Implantação do Protocolo CMIS/CMIP em Ambiente ISODE

Ernandes Lopes Bezerra

Dissertação de Mestrado aprovada em 23/11/1994

Joberto Sérgio Barbosa Martins, Dr.-Ing., UFPB
Orientador

Edilson Ferneda, Dr.,UFPB
Componente da Banca

Maria de Fátima Q. Vieira Turnell, Ph.D.,UFPB
Componente da Banca

Campina Grande, Paraíba, Brasil, dezembro/1994

Dedico este trabalho a Selma Bezerra

Agradecimentos

Gostaria de expressar meus sinceros votos de gratidão as seguintes pessoas e instituições:

Aos meus pais, Ernande e Selma.

Aos professor Pedro Nicolletti, pelo auxilio na instalação do ISODE no laboratório do DSC.

Ao professor Carlos Westphal pelo incetivo, nas minhas visitas a UFRGS.

Ao professor e orientador Joberto Sérgio Martins pela compreensão e incentivo.

A todos que contribuiram de alguma forma para a realização deste trabalho: Washington, Silvana, Joseana, Ângelo e demais companheiros do LAPS.

Ao CNPq, pelo auxílio financeiro nos primeiros meses do trabalho.

A Universidade Federal da Paraíba-Campus II, pela oportunidade oferecida.

Resumo

Neste documento é apresentada a implementação do protocolo para gerenciamento de rede CMIS/CMIP (Common Management Information Service / Common Management Information Protocol) [1] [2] seguindo a padronização da ISO (International Standards Organization) e utilizando o ambiente ISODE (The ISO Development Environment) [3] [4]. Relata-se igualmente neste documento a experiência obtida com a utilização do ambiente ISODE como plataforma de desenvolvimento de protocolos e sistemas distribuídos.

O modelo de implementação do CMIS/CMIP obtido foi baseado no ambiente ROS e nos compiladores ROSY e PEPSY fornecidos pelo ISODE.

A especificação do protocolo CMIS/CMIP é toda baseada no conceito de operações remotas, considerando o uso dos serviços do ACSE para o estabelecimento e controle de associação, e dos serviços de operações remotas do ROSE. Estas características da especificação direcionou a escolha da plataforma de trabalho para uma que já possuísse um ambiente de programação baseado em operações remotas.

O ambiente ROS, contido no ISODE, foi escolhido para a implementação devido ao mesmo conter os serviços de operações remotas ISO e ainda fornecer um ambiente para implementação bastante amigável, incluindo a geração de rotinas e tabelas para a codificação e decodificação de PDUs.

Em relação ao ISODE, foram utilizadas interfaces distintas visando identificar a mais apropriada para implementação foco CMIS/CMIP.

Neste documento, além da apresentação da implantação realizada para o CMIS/CMIP, faz-se uma discussão e análise das facilidades de desenvolvimento de protocolos do ISODE e uma crítica das limitações encontradas na sua utilização como plataforma de desenvolvimento.

Abstract

In this documment is presented the implementation of the network management protocol CMIS/CMIP (Common Management Information Service / Common Management Information Protocol) [1] [2] according to ISO (International Standards Organization) and using the ISODE (The ISO Development Environment) [3] [5]. Is related in this documento too, the experience gained with the utilization of ISODE environment how a platform for development of protocol and distributed systems.

The implementation model reached for CMIS/CMIP was based in ROS environmet and on ROSY and PEPSY compilers supplied by the ISODE.

Everything in the CMIS/CMIP specification is based in the concept of remote operation related with the use of ACSE and ROSE services. This features of the specification directed us to a development platform that had a programming environment based in remote operation too.

The ROS environment contained in ISODE was chosen to the implementation of CMIS/CMIP because it has the ISO remote operations and generation os rotines and tables related with encode and decode PDUs.

Several interfaces of ISODE was used to identify the more apropriated to CMIS/CMIP.

In this document is discussed and analised the facilities with protocols development using ISODE with a valuation about the limitations of the ISODE how a development platform.

Sumário

Lista de Figuras

Lista de Abreviaturas

1. ACSE - Association Control Service Element
2. ASE - Application Service Element
3. ASN.1 - Abstract Syntax Notation.1
4. ATM - Asynchronous Transfer Mode
5. BER - Basic Encoding Rules
6. CMIP - Commom Management Information Protocol
7. CMIPDU - Common Management Information Protocol Data Unit
8. CMIPM - Common Management Information Protocol Machine
9. CMIS - Common Management Information Service
10. CMISE - Common Management Information Service Element
11. FDDI - Fiber Distributed Data Interface
12. FTAM - File Transfer Access and Management
13. IEC - International Electrotechnical Commission
14. IEEE - Institute of Electrical and Electronics Engineers
15. IP - Internet Protocol

16. ISO - International Standards Organization

17. ISODE - ISO Development Environment

18. LAN - Local Area Network

19. LLC - Logical Link Control

20. MAC - Medium Access Control

21. MIB - Management Information Base

22. NFS - Network File System

23. OSI - Open System Interconnection

24. PDU - Protocol Data Unit

25. PEPSY - Presentation Element Parser Structure Generator

26. ROSE - Remote Operation Service Element

27. ROSY - Remote Operations Stub Generator

28. RPC - Remote Procedure Call

29. SAP - Service Access Point

30. SMAE - System Management Application Entity

31. SMI - Structure of Management Information

32. SNMP - Simple Network Management Protocol

33. TCP - Transmission Control Protocol

34. WAN - Wide Area Network

Capítulo 1

Introdução

1.1 Motivação

As redes de computadores são formadas por enlaces de comunicação, sistemas de cabeação, equipamentos para transmissão e comutação, e software que possibilita aos computadores conectados à rede se comunicarem uns com os outros.

O aumento da importância das redes tem gerado a necessidade de acesso eficiente a um número crescente de sistemas de informação, localizados tanto dentro como fora das organizações. Esta tendência provoca uma descentralização cada vez maior dos sistemas de informações envolvidos [6].

Na fase inicial de sua utilização as redes de computadores eram de um único fornecedor, que normalmente desenvolvia o projeto de todo o sistema. As redes se resumiam então à conexão dos equipamentos mais periféricos aos sistemas de computadores, normalmente de grande porte ("mainframes"). Os enlaces de dados consistiam de conexões ponto-a-ponto entre os computadores. Para os enlaces locais usava-se algum tipo de cabeação, e para os de longa distância, eram usados circuitos telefônicos alugados. Nesta fase de evolução as velocidades envolvidas eram bastante baixas (alguns Kbps).

A tecnologia das redes de computadores evoluiu em diversas direções. À título de exemplo tem-se, dentre outros:

- Os meios de transmissão são utilizados de forma mais eficiente, (par trançado e fibra óptica são exemplos típicos);
- As tecnologias de rede evoluíram em termos de velocidade (Ethernet, FDDI,...) e, em alguns casos, em termos de integração de serviços (ATM, DQDB).
- Os equipamentos evoluíram em desempenho e agregação funcional.
- Difusão da tecnologia com acesso abordável para os seus usuários.

Durante a evolução da tecnologia das Redes de Computadores, foi dada uma grande enfase ao desenvolvimento de protocolos heterogêneos que permitissem a comunicação entre sistemas de computadores. Um importante passo neste sentido foi alcançado com o surgimento da rede ARPAnet, criada nos EUA, que introduziu os protocolos TCP/IP, como arquitetura comum para a interconexão de computadores heterogêneos [7] [8]. Um outro grande passo da evolução tecnológica das redes foi o desenvolvimento de um padrão internacional através do Modelo de Referência para a Interconexão de Sistemas Abertos (RM-OSI) da ISO (International Organisation for Standardisation) [9].

Com a utilização das redes e o crescimento da capacidade dos computadores, começaram a surgir outras necessidades:

- Transferência e acesso transparente de arquivos (FTAM, NFS, ...);
- Compartilhamento de periféricos (impressoras, servidor de arquivos, ...);
- Interfaces APIs (Aplication Programming Interfaces) para programação distribuída;
- Mecanismos de programação transparentes (RPC,...) em rede;
- Outros.

Finalmente, tem-se como mais uma etapa da evolução, a necessidade de gerenciamento da rede e de seus sistemas de informação.

De maneira geral, quanto maior e mais complexa a rede, maior a necessidade de gerenciamento eficiente de seus componentes. O gerenciamento adequado é uma ação

complementar imprescindível à rede que objetiva garantir a sua operação e manutenção de forma eficiente.

considerando o contexto global de uso das redes, a solução de gerenciamento adotada deve:

- Ser suficientemente genérica no sentido de poder gerenciar sistemas distintos e heterogêneos;
- Ser flexível no sentido de poder acompanhar a evolução tecnológica;
- Preservar a idéia de padronização que viabiliza o gerenciamento heterogêneo.

1.2 Objetivos do Trabalho

A grande importância do gerenciamento de redes para a sua operação e manutenção levou ao desenvolvimento do trabalho aqui exposto.

O escopo do trabalho tem como objetivos principais:

- A implementação do protocolo padrão de gerência de redes CMIS/CMIP da ISO, considerando que no início do trabalho não existia nenhuma implementação disponível do mesmo;
- A experimentação prática do ambiente de desenvolvimento ISODE em ambiente SUN/UNIX para o desenvolvimento de protocolos.

Este trabalho se encaixa igualmente no escopo dos trabalhos já desenvolvidos e em desenvolvimento da área de gerência do Grupo de Redes de Computadores (GRC-UFPB). Entre os trabalhos já desenvolvidos encontramos uma implementação do protocolo FTAM [10] e uma metodologia para modelagem de uma base de informações de gerenciamento para o Modelo OSI [11], que juntamente com o trabalho aqui descrito podem servir de base para outros projetos na área de gerência de redes.

A ação do GRC visa globalmente o desenvolvimento de plataformas de gerenciamento SNMP e CMIS/CMIP (protocolos, agentes, gerentes e MIBs) para as redes de alta velocidade com ênfase na tecnologia ATM.

1.3 Descrição do Conteúdo dos Capítulos

- **Capítulo 1** - Faz uma abordagem inicial sobre a área de redes de computadores situando a necessidade de gerenciamento de rede.
- **Capítulo 2** - Descreve o Modelo de Referência OSI/ISO, abordando principalmente a camada de aplicação e introduz o modelo de gerenciamento OSI.
- **Capítulo 3** - Descreve o ambiente de desenvolvimento ISODE e aborda os procedimentos usados para definição de um novo serviço dentro do ISODE, seu esquema de endereçamento, o seu modelo de implementação e as ferramentas que facilitam o trabalho de implementação no ISODE.
- **Capítulo 4** - Decreve os serviços e o protocolo do CMIS/CMIP e sua interação com outros componentes da camada de aplicação. É feita uma comparação entre os serviços do CMIS/CMIP com o SNMP.
- **Capítulo 5** - Aborda a implementação do CMIS/CMIP no ambiente ISODE, descrevendo as interfaces utilizadas, os procedimentos e uma avaliação crítica da experiência obtida para o desenvolvimento de protocolos.
- **Capítulo 6** - É o capítulo de conclusão, onde são identificandos os resultados e os pontos relevantes do trabalho dentro do contexto de gerência de redes.

Capítulo 2

Gerência de Redes

2.1 Introdução

O gerenciamento de redes pode ser definido como o planejamento, a monitoração, a contabilização e o controle das atividades e recursos de rede.

Este capítulo é dedicado ao Modelo de Gerenciamento de Rede OSI, definido pela ISO e parte integrante do Modelo de Referência OSI.

Um gerenciamento de redes baseado em sistemas abertos, reduz o custo de interface entre os diversos processos e permite a troca uniforme de mensagens de informações de gerenciamento, possibilitando que cada sistema em particular consiga analisar cada campo de informação contida nas mensagens de gerenciamento.

Antes da introdução do gerenciamento OSI, foco deste trabalho, é feita uma descrição suscinta do Modelo de Referência OSI/ISO no sentido de possicionar o assunto dentro da sua estrutura.

2.2 Modelo de Referência OSI

Um sistema é considerado aberto quando pode se comunicar com outros sistemas de outros fabricantes. Os sistemas que obedecem aos padrões ditados pelo Modelo de Referência OSI/ISO para sua comunicação com outros sistemas são considerados abertos.

Os protocolos de rede usam um modelo em camadas hierárquicas, cada um possuindo um conjunto bem definido de funções e uma interface com as camadas acima e abaixo bem especificada. A organização em camadas hierárquicas simplifica a concepção dos protocolos e a atualização da tecnologia usada em uma dada camada, sem afetar as camadas vizinhas [12] [9].

Segundo A.S.Tanembaum [13], o número de sete camadas foi escolhido para compor o Modelo de Referência OSI em função dos seguintes princípios:

- Uma camada deve ser criada quando for necessário um novo nível de abstração.
- Cada camada deve executar um conjunto bem definido de funções.
- A interface deve ser especificada de forma a minimizar o fluxo de informações entre as camadas.
- O número de camadas deve ser escolhido de forma a não levar a coexistência de funções distintas em uma mesma camada.
- Deve ser possível atualizar uma camada sem afetar nenhuma outra.

As funções executadas pelas camadas de protocolos podem ser divididas em funções orientadas para a rede de comunicação e funções orientadas para a aplicação do usuário, criando desta forma ambientes operacionais distintos [14]. Dentre estes ambientes, identificamos os seguintes:

- A Rede: trata dos protocolos e padrões relacionados com os diferentes tipos de redes de comunicação de dados que podem ser utilizadas.

- A Arquitetura OSI: trata a rede juntamente com os protocolos e padrões de alto nível orientados à aplicação, de forma a permitir que programas e aplicações do usuário (sistemas finais) possam se comunicar uns com os outros.

- Sistemas Finais: Trata o software e os serviços proprietários, que se utilizam da arquitetura OSI (ambiente OSI) para excutar tarefas relacionadas com o processamento distribuído de informações.

No modelo hierárquivo, cada camada opera de acordo com um protocolo especificado, possibilitando a troca de mensagens contendo informações de controle e dados do usuário entre as entidades pares deste protocolo, localizados nas pilhas dos sistemas remotos [15].

Cada camada fornece um conjunto bem definido de serviços para a camada imediatamente superior e utiliza os serviços oferecidos pela camada imediatamente inferior. A interação entre as camadas ocorre através dos pontos de acesso de serviço SAPs (Service Access Point), o que possibilita a troca de primitivas de serviço entre os níveis. Desta forma, a camada (N) se comunica com as camadas (N-1) e (N+1), possibilitando a comunicação lógica ou virtual com entidades de camada (N) pares em sistemas distintos. Cada camada se comunica com a camada par similar em um sistema remoto, de acordo com o protocolo definido para a camada específica, através da troca de Unidades de Dados de Protocolo (UDP) ou PDU (Protocol Data Unit), consistindo em uma comunicação virtual (Figura 2.1).

Um serviço representa um conjunto bem definido de funções oferecidas a uma entidade usuária, fornecido por uma entidade provedora, de uma camada qualquer.

Cada protocolo é especificado por dois tipos de documentos:

- Um que define o serviço oferecido à camada superior, o usuário do serviço;

- Outro que especifica as Unidades de Dados do Protocolo, geralmente em uma sintaxe abstrata, como é o caso da ASN.1.

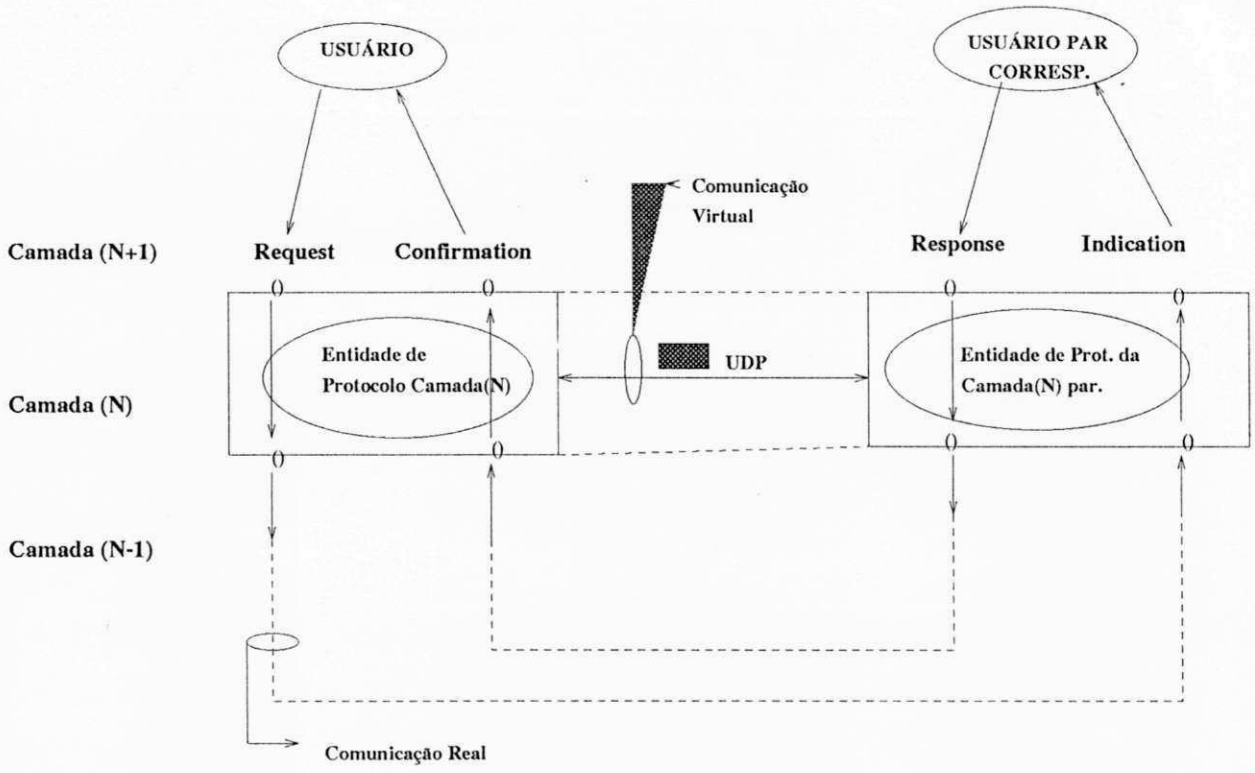

Figura 2.1: Primitivas de Serviço vs UDPs.

Os serviços geralmente são especificados na forma de primitivas de serviço, cada uma das quais com um conjunto de parâmetros de serviço. A especificação do protocolo envolve a definição precisa de cada PDU que será usada para comunicação com a entidade de protocolo par no sistema remoto. O protocolo define também os serviços que serão usados da camada inferior. Normalmente é feita a descrição de cada operação específica da entidade do protocolo através de um método de descrição formal, geralmente ASN.1 (Abstract Syntax Notation One) [16] [17].

Os serviços fornecidos por cada camada podem ser serviços confirmados ou não confirmados. São considerados confirmados aqueles serviços que possuem algum tipo de resultado (ex: se a associação foi estabelecida ou rejeitada, etc). Um serviço é solicitado através de uma primitiva do tipo "REQUEST". Os dados desta primitiva são codificados e passados para a camada inferior como dados do usuário de uma outra primitiva de serviço relacionada com a camada inferior. A PDU formada à partir

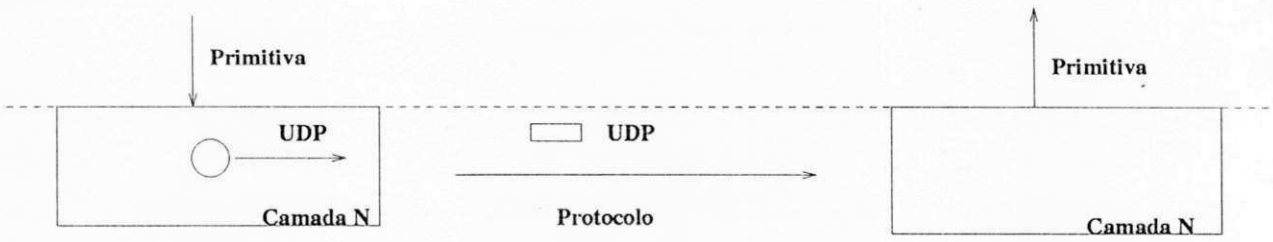

Figura 2.2: Troca de Primitivas de Serviço.

da primitiva inicial chegando na camada par do sistema remoto como dados contidos em uma primitiva do tipo "INDICATION". No caso de serviços não confirmado significa o fim da operação. Para os serviços confirmados, a entidade que recebe uma primitiva do tipo "INDICATION" deve enviar uma primtiva tipo "RESPONSE" de volta. Desta forma, a entidade que iniciou a operação recebe uma primitiva do tipo "CONFIRMATION". Ver Figura 2.1 e Figura 2.2.

2.2.1 Descrição das Camadas do Modelo de Referência OSI

- **Camada 1 ou Camada Física**: Trata da transmissão de bits em um canal de comunicação. Eletricamente codifica e fisicamente transfere mensagens entre os nodos da rede. As especificações da camada física descrevem as interfaces elétricas e mecânicas, assim como o meio físico de transmissão usado. Como exemplo encontramos: cabeamento com pares trançados, cabos de fibra óptica, transmissão em microondas e cabo coaxial de banda larga, entre outros.

- **Camada 2 ou Camada de Enlace**: A camada de enlace trata da transmissão de quadros ("frames") e da detecção de erros que podem ocorrer nos quadros que se movem ao longo da rede. Ela também faz o controle do fluxo de dados para controlar casos nos quais o transmissor é mais rápido que o receptor. Como exemplos de especificações a nível de enlace de dados, existem vários protocolos ponto-a-ponto: SDLC (Synchronous Data Link Control), HDLC (High Level Data Link Control), LAPB (Link Access Procedure B), redes X.25. Nas redes

locais que utilizam meios compartilhados, a camada de enlace de dados é dividida em duas sub-camadas:

1. *Sub-camada* **MAC** (Medium Access Control), que executa um algorítimo de acesso ao meio, também denominado "protocolo de acesso" ou "tecnologia de rede". Exemplos de métodos de acesso são o CSMA-CD (Carrier Multiple Access with Collision Detection) padrão IEEE 802.3, conhecido como Ethernet, o IEEE 802.5, conhecido como Token Ring, e o FDDI (Fiber Distributed Data Interface), entre outros.
2. *Sub-camada* **LLC** (Logical Link Control), que trata das funcionalidades adicionais da camada de enlace (sequenciamento, controle de fluxo, ...) para os diferentes métodos de acesso.

- **Camada 3 ou Camada de Rede**: A camada de rede faz o roteamento dos pacotes de dados da rede de origem até a rede de destino. Esta é uma tarefa simples quando se considera uma rede local isolada, mas se torna muito mais complexa quando se considera redes interconectadas. O protocolo da camada de rede pode ser orientado à conexão (quando todos os pacotes seguem o mesmo percurso entre a origem e o destino) ou sem conexão (quando o percurso de cada pacote é determinado independentemente).

- **Camada 4 ou Camada de Transporte**: A camada de transporte é responsável pela transferência de dados confiável e é transparente entre os sistemas finais da rede. A camada de transporte introduz o conceito de qualidade de serviço (QOS) através das diversas classes de serviço que possui. A camada de transporte forma uma interface entre as camadas orientadas à aplicação e as camadas dependentes da rede. A camada de transporte e todas as outras que existirem acima são implementadas apenas nos sistemas finais (computadores da rede), enquanto as camadas abaixo dela podem ser implementadas tanto nos sistemas finais como em nós intermediários (pontes e roteadores).

- **Camada 5 ou Camada de Sessão**: É responsável pelo controle do diálogo e pelo gerenciamento da sincronização. O controle de diálogo estabelece quem

vai "falar" em um determinado momento através do controle de "tokens". O gerenciamento da sincronização estabelece pontos para checagem e sincronização ao longo da transmissão de forma a permitir, na ocorrência de erros, que seja retransmitido somente o trecho de mensagem a partir do último ponto de sincronização.

- **Camada 6 ou Camada de Apresentação**: A camada de apresentação trata da conversão dos dados, garantindo a tradução do mesmos para um formato comum. Como exemplo, quando dois computadores usam diferentes conjuntos de caracteres (EBCDIC ou ASCII), se torna necessário a conversão para um formato comum a ambos. Além disto, a camada trata também de criptografia e compressão de dados.

- **Camada 7 ou de Camada de Aplicação**: A camada de aplicação fornece os serviços que são diretamente usados pela aplicação do usuário a fim de executar tarefas para os mesmos. A figura 2.3 mostra a estrutura e a relação entre as camadas do Modelo de Referência OSI.

A implementação descrita neste trabalho trata do protocolo para gerenciamento de redes, o CMIS/CMIP, que no modelo OSI ocorre à nível de aplicação.

A camada de aplicação é composta por diversas entidades de serviço de aplicação (Association Control Service Element - ACSE, Remote Operation Service Element - ROSE, etc). Um processo do usuário à nível de aplicação geralmente envolve a utilização de vários serviços comuns ou serviços específicos oferecidos por esta camada. Ver figura 2.4.

Um processo de aplicação pode utilizar os serviços de mais de um Elemento de Serviço de Aplicação, conhecidos por (ASEs). Um ASE é uma parte de um processo de aplicação que serve para um propósito específico (ex: Uma determinada aplicação pode utilizar os serviços de Terminal Virtual, FTAM, ROSE, ACSE, RTSE, CMIP, etc). Desta forma, a aplicação pode utilizar os serviços de diversos ASEs para implementar os serviços oferecidos ao usuário final.

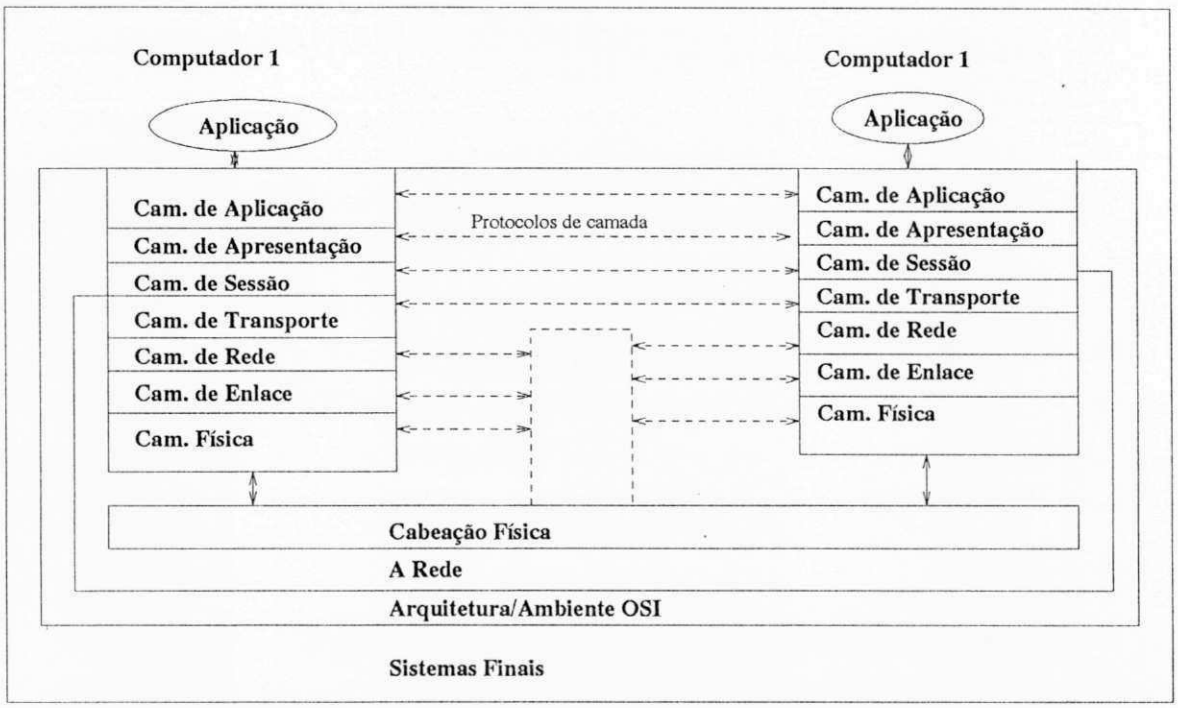

Figura 2.3: Estrutura do Modelo de Referência OSI.

2.3 Modelo de Gerenciamento OSI

A ISO desenvolveu uma solução para a gerência de redes de computadores. A solução padronizada proposta procura lidar com a complexidade do problema de gerência e assume o modelo descrito abaixo:

O modelo de gerenciamento ISO inclui gerentes, agentes, protocolo de gerência, objetos gerenciados e seu repositório (MIB-Management Information Base), além de toda uma estrutura para modelagem da informação ver Figura 2.5.

- **Agente** - É um processo executando num sistema sendo gerenciado. É uma parte do processo de gerenciamento de rede. O agente informa ao gerente o estado dos elementos gerenciados da rede e recebe diretivas do processo gerente sobre ações que podem envolver estes elementos através de um protocolo de gerência de redes (no caso: CMIP). A MIB fica localizada no agente de gerenciamento.

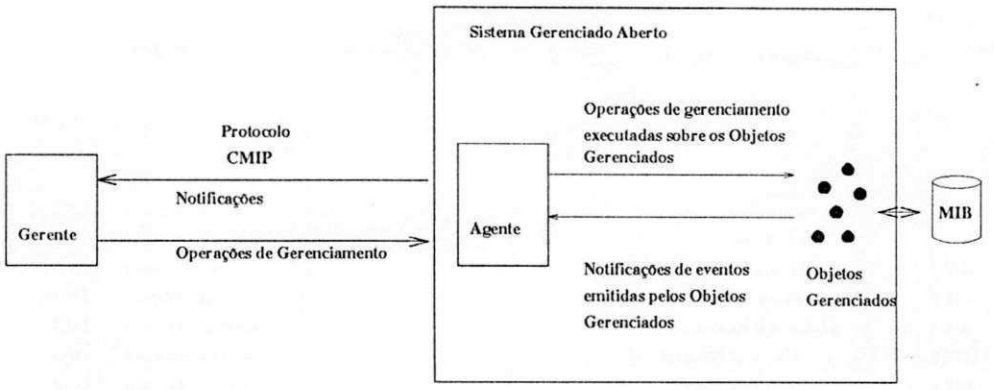

Figura 2.5: Relação entre Agentes, Gerentes e Objetos Gerenciados.

e, também, a comunicação gerente/gerente. É um protocolo padrão orientado a transação, que usa serviço de padrões também comuns como o ACSE e o ROSE. É descrito em detalhe no capítulo 4.

A responsabilidade da troca de informações é do CMIP que permite aos agentes enviarem notificações para os gerentes, e aos gerentes enviarem diretivas de operações para os agentes. Um agente executa as operações de gerenciamento sobre os objetos e envia notificações sobre estes objetos para o processo gerente (Figura 2.6). Cada camada possui uma entidade, ("LME-Layer Management Entity"), que é responsável pelas suas interações de gerenciamento.

Cada recurso da rede é supervisionado e controlado globalmente pelo centro de controle da rede (processo gerente) através do seu mapeamento na forma de objetos gerenciados. Este controle pode ser centralizado ou descentralizado. No contexto desta apresentação consideramos, por simplicidade, um gerente controlando todos os objetos da rede.

A localização física dos componentes de gerenciamento podem ocorrer como na Figura 2.7.

Dentro do Modelo de Gerenciamento ISO, está definida a Entidade de Aplicação de Gerenciamento de Sistema (SMAE), uma SMAE corresponde a uma entidade a nível de

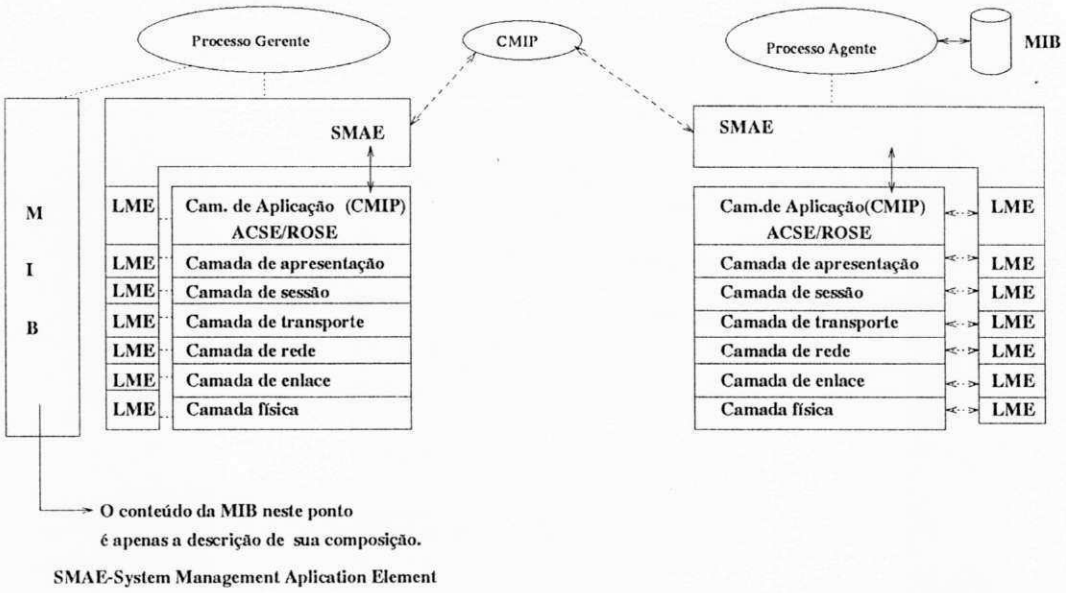

Figura 2.6: Localização do Gerente e do Agente dentro do OSI.

aplicação, cujas funções são utilizadas pelo gerente e pelo agente. Ver Figura 2.8. Uma SMAE fornece ao processo de aplicação usuário da mesma um determinado conjunto de funções especificadas das área funcionais do modelo de gerenciamento OSI. Uma SMAE faz uso dos serviços do CMISE, do ACSE e do ROSE para implementar as aréas funcionais do modelo de gerenciamento. A interação entre os SMAEs de sistemas remotos ocorre através dos serviçdo CMIS/CMIP.

Os Elementos de Serviço da camada de aplicação que são úteis para o gerenciamento de redes OSI são: ACSE, ROSE (que são utilizados pelo CMISE) e FTAM (que pode ser utilizado na implementação das funções de gerenciamento descritas adiante).

O **contexto de aplicação** utilizado pelo CMISE define os serviços à nível de aplicação que o mesmo utiliza, assim como o relacionamento entre estas entidades e a camada de apresentação.

O contexto de aplicação estabelece os Elementos de Serviço de Aplicação que podem ser usados durante a existência da associação. O contexto de aplicação relacionado ainda inclui uma descrição:

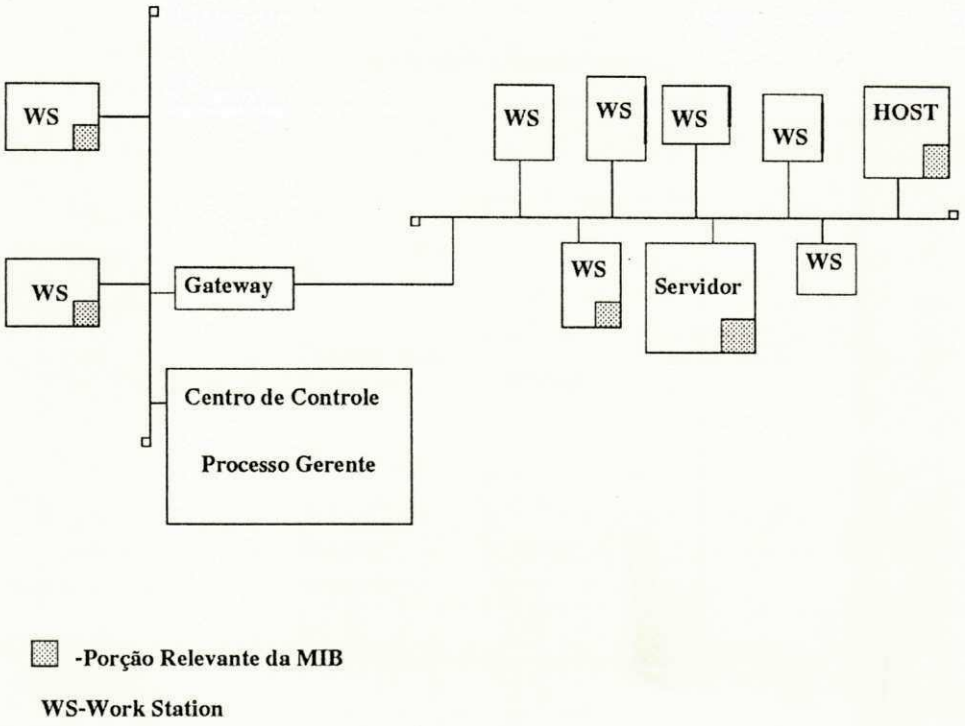

Figura 2.7: Localização dos Componentes de Gerenciamento de Rede.

- dos relacionamentos e a dependência do ACSE e do ROSE.
- das características opcionais que podem ser usadas de cada Elemento de Serviço.
- da estrutura lógica da informação que será trocada entre as entidades que implementam o protocolo nos sistemas pares.
- das regras para o mapeamento da informação de controle do protocolo CMIP na primitiva de apresentação.

A associação entre os dois processos CMIP, rodando em dois sistemas remotos é estabelecida e gerenciada através dos seguintes serviços do ACSE:

- A-ASSOCIATE
- A-RELEASE

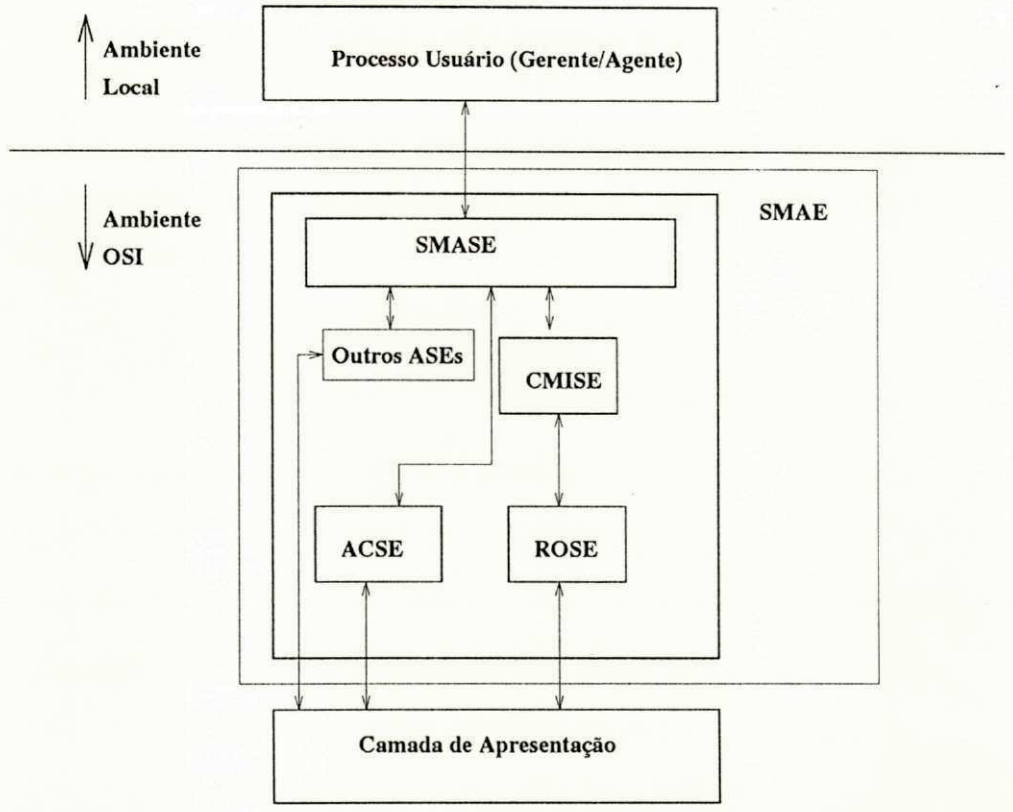

Figura 2.8: Estrutura do Gerenciamento ISO.

- A-ABORT
- A-P-ABORT

Cada associação de aplicação recebe um nome, correspondente a um identificador de objeto, que é a forma como a mesma será tratada. Na negociação da associação é estabelecida a sintaxe abstrata e a sintaxe de transferência das PDUs de aplicação que serão trocadas na associação. No caso do CMIP, a sintaxe abstrata é a definida em [17] e a sintaxe de transferência é baseada em [16].

O ROSE trata das transações relacionadas com a execução da operações no sistema remoto. As operações solicitadas podem ocorrer em dois modos distintos:

- Modo **síncrono**, que bloqueia o entidade que solicita a operação até o retorno de um resultado ou erro;
- Modo **assíncrono**, quando a entidade que executou solicitou o serviço ou executou a primitiva é logo liberada, ficando um identificado relacionado com a invocação para o reconhecimento do retorno da mesma. Neste caso, a entidade fica liberada para fazer novas operações.

Os serviços de gerenciamento podem divididos em três tipos:

- **Monitoração**: É um serviço que permite a obtenção de informação de gerenciamento sobre objetos.
- **Controle**: É um serviço que objetiva controlar os recursos (objetos) da rede..
- **Notificação**: É um serviço através do qual obtem-se informações sobre a ocorrência de eventos anormais em recursos (objetos) da rede.

Os serviços de gerenciamento são acionados utilizando-se o modelo de operações remotas, devido toda a especificação do CMIS/CMIP ser baseado no conceito de operações remotas. Este serviço pode ser melhor percebido na interação entre dois SMAEs monstrada na figura 2.9. Estes SMAEs podem ser usados pelo processo agente e pelo processo gerente.

A definição de cada objeto gerenciado inclui quatro aspectos do gerenciamento de redes:

- **Atributos**: São as características de cada objeto e podem ser consideradas como a parte visível de cada recurso (objeto) da rede.
- **Operações**: Elas formam o conjunto de operações que podem ser executadas sobre os recursos (objetos) da rede.
- **Notificações**: São informações geradas a respeito do estado dos recursos (objeto) da rede.

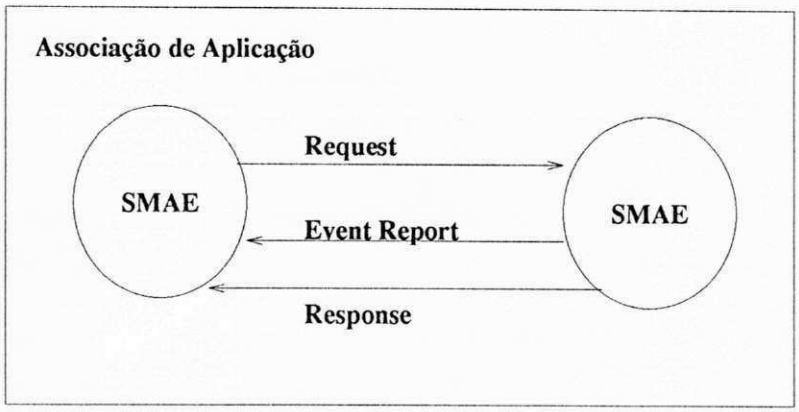

Figura 2.9: Interação entre SMAEs.

- **Comportamento**: Está relacionado com as modificações de estado que ocorrem em consequência ou como resposta às operações executadas sobre os mesmos.

Os atributos de cada objeto podem ser vistos como certas características que os distinguem de todos os outros. Os atributos identificam as características, o estado e as condições de operação do objeto gerenciado. Cada atributo consiste de um tipo com um ou mais valores. Ver Figura 2.10.

As possíveis operações sobre objetos gerenciados são as seguintes:

- *Create*: Cria um novo objeto gerenciado.
- *Delete*: Deleta o objeto da base de dados (MIB).
- *Action*: Executa uma operação sobre o objeto gerenciado.
- *Get Value*: Obtém o valor de um atributo de um objeto gerenciado.
- *Add Value*: Soma um determinado valor a um atributo de um objeto gerenciado.
- *Remove Value*: Remove (zera) o valor de um atributo.
- *Replace Value*: Substitui o valor de um atributo.

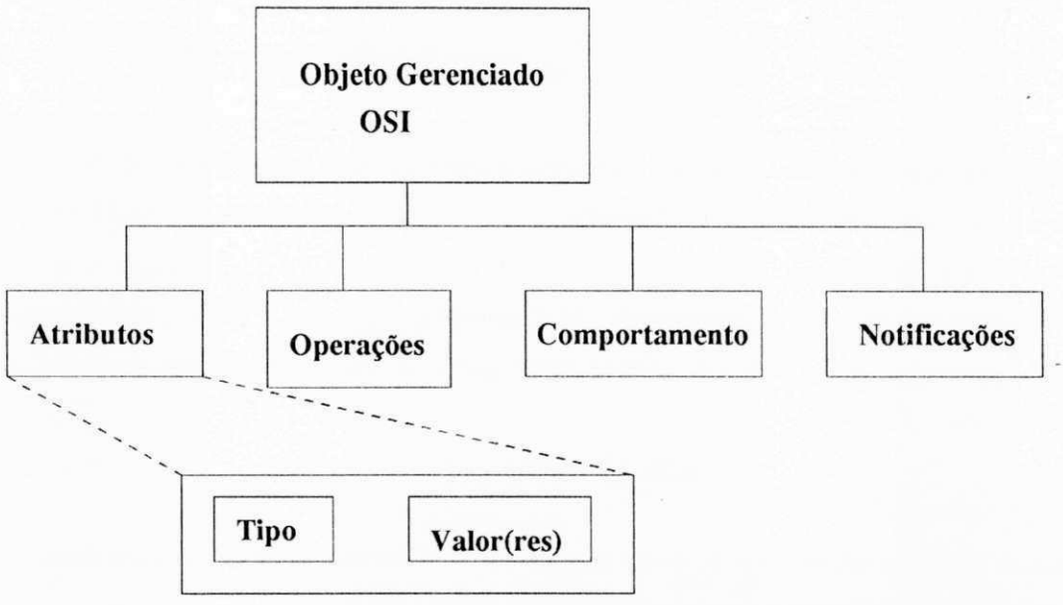

Figura 2.10: Definição de um Objeto Gerenciado.

- *Set Value*: estabelece valores pré-definidos ("default") para um atributo de um objeto.

Os objetos gerenciados podem enviar notificações sobre eventos ocorridos, e sua natureza depende de como o objeto é classificado dentro da rede.

O comportamento do objeto gerenciado inclui a forma como ele deve reagir quando for executada uma determinada operação, e estabele condições sobre as quais os valores de seus atributos devem ser alterados. Estas condições podem ser relacionadas com eventos externos: quando uma operação altera o valor de um atributo, ou internos: quando algum evento interno ao objeto altera o valor do mesmo.

O Modelo de Gerenciamento da ISO especifica mecanismos para comunicação da informação de gerenciamento, é divididos em:

- **Gerenciamento de Sistemas** - que trata de informações relacionadas com a monitoração, o controle e a coordenação dos objetos gerenciados e o sistema final.
- **Gerenciamento de Camada** - É usado em circunstâncias especiais nas quais é necessário a manipulação de informações relacionadas com camadas específicas;

- **Operação de Camada** - Geralmente fornece um conjunto de facilidades que controlam e gerenciam uma instância de comunicação.

A Gerência de Sistemas é o nível mais alto da estrutura de gerenciamento OSI. É através da Gerência de Sistemas que é feita a coordenação, a monitoração e o controle dos recursos dentro de um ambiente OSI. Os recursos de comunicação que normalmente são gerenciados podem ser vistos na Figura 2.11. Além destes recursos de comunicação outros recursos componentes da rede também podem ser gerenciados (multiplexadores, hubs, roteadores, etc). Estes são definidos como objetos gerenciados para fins de gerenciamento e sua representação está contida na MIB.

Camada OSI	Recurso Gerenciado Relacionado com a Camada
Camada de Aplicação	MHS, VT, Serviço de Diretório, FTAM, TP
Camada de Apresentação	Entidades da Camada de Apresentação
Camada de Sessão	Entidades da Camada de Sessão
Camada de Transporte	Entidades da Camada de Transporte
Camada de Rede	RDSI Rede de Comutação de pacotes, B-RDSI Rede de Comutação de Circuitos
Camada de Enlace	Redes Locais Redes de Longa Distância
Camada Física	Linhas Dedicadas, Conexões de Satélite, etc

LAN - Rede local
WAN - Rede de Longa Distsância
RDSI - Rede Digital de Serviços Integrados
B-RDSI- Rede RDSI Banda Larga
MHS - Message Handling System
VT - Virtual Terminal
FTAM - File Transfer Access and Manag.
TP - Transaction Processing

Figura 2.11: Recursos de Comunicação Gerenciados.

As informações relacionadas com o gerenciamento de redes e que trafegam na rede

têm como orígem:

- Outros sistemas abertos.
- Comunicação gerente/agente, através do protocolo de gerência;
- Informações sobre recursos gerenciados locais;
- Comunicação dos protocolos das camadas inferiores.

Dentro do Modelo de Gerência OSI a comunicação pode ocorrer também entre domínios de gerenciamento (Figura 2.12), que são criados devido a característica distribuída dos Sistemas de Gerência. Estes domínios são agrupamentos de recursos que participam do ambiente de gerência OSI baseados em critérios funcionais, geográficos, tecnológicos, etc, com o objetivo de estabelecer uma organização do controle dos recursos gerenciados [18].

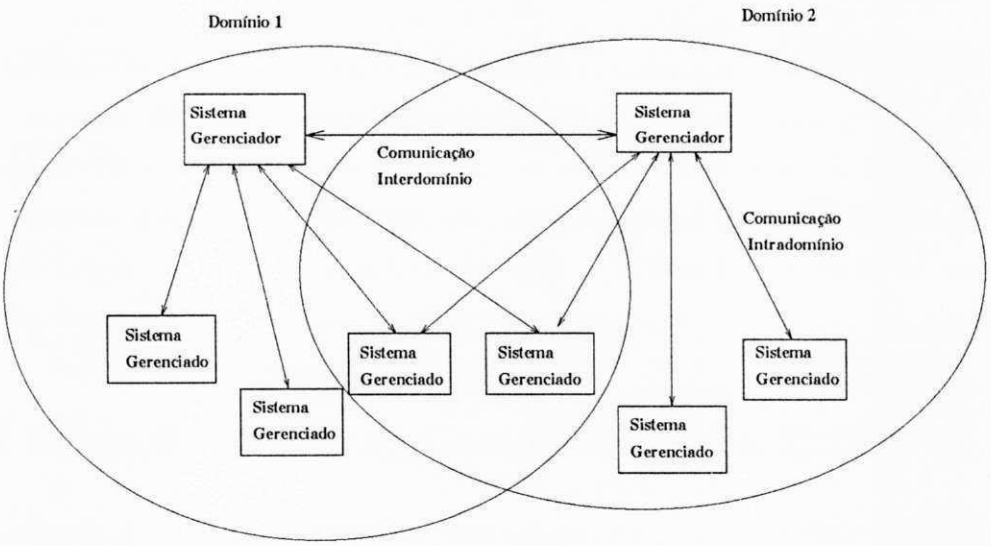

Figura 2.12: Organização entre os Domínios de Gerência.

2.4 Áreas Funcionais do Gerenciamento OSI

O Modelo de Gerência é dividido em cinco áreas funcionais:

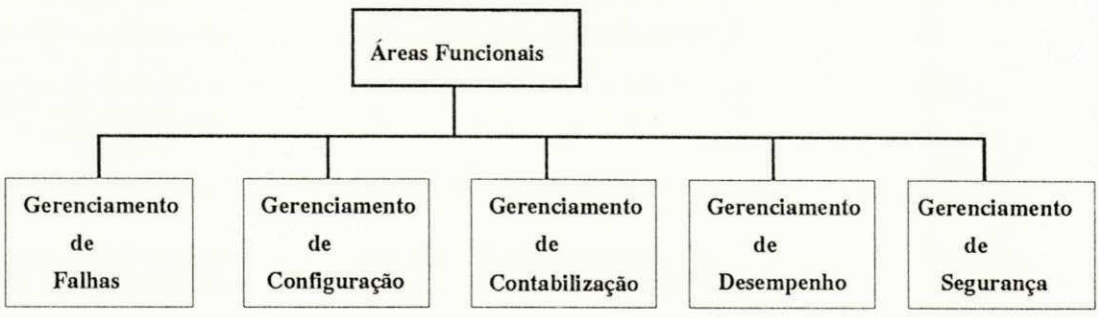

Figura 2.13: Áreas Funcionais do Gerenciamento de Rede OSI.

2.4.1 Gerenciamento de Falhas

O gerenciamento de falhas é usado para detectar, isolar e solucionar problemas envolvendo por exemplo os recursos de comunicação Figura 2.11 mapeados como objetos gerenciados. Executa atividades do tipo: encontrar defeitos na rede, ou transmitir informações de diagnóstico e corrigir, quando possível, as situações de erro. Esta área funcional também trata dos registros de erros, e utiliza basicamente as notificações recebidas dos agentes a respeito dos objetos gerenciados.

2.4.2 Gerenciamento de Contabilização ou Tarifação

Este conjunto de funções é necessário para qualquer tipo de ambiente de recursos compartilhados. É ele que define a utilização da rede, taxas de utilização e custos envolvidos, determinando desta forma parâmetros de valores para a utilização dos serviços. Permite também aos usuários e gerentes, estabelecerem limites sobre a utilização e negociarem o uso de recursos adicionais.

2.4.3 Gerenciamento de Configuração

Este conjunto de funções identifica o estado de operação dos equipamentos da rede possibilitando o controle dos mesmos. Estas funções possibilitam colocar um equipamento em serviço, fora de serviço, reservado, etc. Permite rotear o tráfego em consequência de falhas identificadas na rede, excluir o serviço ou equipamento em falha e fazer teste de inicialização de outros equipamentos.

2.4.4 Gerenciamento de Segurança

Trata da proteção dos objetos gerenciados através de controle de acesso, manutenção de facilidades de autorização, e criação de "logs" de segurança.

2.4.5 Gerenciamento de Desempenho

Coleta estatísticas sobre os dados da rede e aplica rotinas de análise que calculam o desempenho do sistema. Possibilita a detecção de tempos de respostas insuficientes, eficiência dos sistemas, etc. Está relacionado com conceitos residentes em outras camadas da rede como: taxas de erros nas camadas, atrasos no estabelecimento de conexões, etc.

2.5 Comparação entre CMIP e SNMP

O sucesso atual do **SNMP** ("Simple Network Management Protocol") versão 1 [19] [20] parte do fato dele ser relativamente fácil de implementar, requerendo poucos recursos de máquina (memória, processamento,...). Usar poucos recursos de máquina é importante, pois em alguns casos equipamentos pequenos já existentes na estrutura da rede não necessitam ser trocados, por motivo de sua baixa capacidade, para serem gerenciados. Embora o uso do SNMP esteja bastante difundido, ele possui algumas limitações:

- O paradigma de gerenciamento do SNMP usa amplamente a estratégia de "polling", que não funciona muito bem para monitorar redes de longa distância, pois eles causam um grande número de mensagens SNMP, o que degrada o tempo de resposta [21] e implica em custos operacionais que podem ser significativos.

- O SNMP é ineficiente para a recuparação de grandes quantidades de dados fornecidos pelos dispositivos gerenciados.

- Os "traps" SNMP (mensagem sobre ocorrência de eventos) não são confirmados, e como são enviados sobre o UDP/IP (baseado em datagramas), não há nenhuma garantia de que chegam ao destino.

- O mecanismo de autenticação das implementações é relativamente simples, de forma que o SNMP possui mais utilidade para monitoração do que para controle.

- O SNMP não possui nenhuma definição para execução de comando imperativos de produtos especificos, como é o caso de auto-testes.

- O modelo de informação de gerenciamento do SNMP é limitado, pois geralmente as implementações não oferecem aplicações que permitam ao administrador da rede solicitar operações baseado somente em tipos e valores de objetos.

- O SNMP não oferece nenhuma solução para o problema da comunicação gerente-gerente. Como exemplo, não há nenhum mecanismo que permita a um sistema de gerenciamento adquirir conhecimento sobre a rede gerenciada a partir de um outro sistema de gerenciamento.

Embora o CMIP não possua todos os problemas do SNMP, este também tem algumas limitações:

- As implementações do CMIP necessitam de um quantidade significativo de memória e de recursos de processamento, sendo necessário, em alguns casos, reestruturar ou adaptar a arquitetura, principalmente naqueles equipamento que não possuem toda a pilha de protocolos OSI.

- A Complexidade exigida para a implementação em relação à conformidade com a padronização e com os testes de interoperabilidade.

Em relação a este trabalho, optamos pelo CMIS/CMIP devido o mesmo ser um padrão internacional de gerenciamento ISO e pela quantidade de recursos oferecidos pelo mesmo para gerencia de rede que, no caso, superam o SNMP.

Obviamente, a opção CMIS/CMIP tem igualmente um carater de experimentação acadêmica.

Capítulo 3

ISODE - Ambiente para Desenvolvimento de Aplicações Distribuídas

3.1 Motivação

O ISODE é um pacote para desenvolvimento de aplicações distribuídas, com facilidades para a implementação de padrões definidos no Modelo OSI para as camadas superiores do RM-OSI em sub-rede OSI ou TCP/IP. O ISODE disponibiliza algumas aplicações OSI, como o seriviço de diretório, transferência de arquivos, uma implementação do SNMP vs2 (Simple Network Management Protocol), etc

Embora o ISODE tenha sido escrito inicialmente para rodar em cima do sistema operacional UNIX, é portável para uma vasta gama de variantes do UNIX : HP-UX, OsX, IBM AIX, etc [4]. O ISODE deverá fazer parte do "Berkeley UNIX" nas próximas versões, considerando-se sua utilidade e importância para o meio acadêmico.

O ISODE pode ser visto como uma plataforma de trabalho cuja implementação não é proprietária e fornece os serviços dos níveis superiores do RM-OSI. Marshall T. Rose foi o seu principal idealizador. O objetivo da criação do ISODE é incentivar

o desenvolvimento de aplicações no ambiente de protocolos OSI, embora o mesmo também possa ser utilizado para o desenvolvimento de aplicações distribuídas genéricas.

O ISODE é uma plataforma de desenvolvimento amplamente difundida em outras universidades através do mundo.

O desenvolvimento de protocolos de comunicação a partir da utilização do ISODE é extremamente simplificado, desde que o ambiente de desenvolvimento seja disponibilizado com suporte técnico adequado, o que possibilita uma rápida prototipação de aplicações de gerenciamento de redes.

As facilidades, como rotinas de codificação e decodificação de PDUs, torna o ISODE uma opção extremamente eficiente e avançada comparada a outras formas de implementação, ie. utilizando "SOCKETS".

O interesse na utilização do ISODE para o desenvolvimento da implementação do protocolo CMIS/CMIP, se concentrava na experiência pioneira representada pela utilização do ISODE nos laboratórios da UFPB, permitindo com a mesma, uma possível utilização do ambiente em outros trabalhos da área de redes.

Com a versão do SNMP disponível no pacote de aplicativos, foi possível uma comparação da complexidade dos dois protocolos, sendo o CMIS/CMIP muito mais complexo do que o SNMP. Não foi possível fazer testes de desempenho envolvendo os dois protocolos, devido a falta de suporte técnico.

No caso do CMIS/CMIP, a interação entre o gerente e o agente utilizando os serviços do ISODE pode ser visto na Figura 3.1.

Todas as bibliotecas do ISODE são implementações dos padrões internacionais para as camadas superiores. Estas bibliotecas são disponibilizadas em forma de funções implementadas em linguagem C. Além das bibliotecas de funções, o ISODE oferece uma série de ferramentas que facilitam o desenvolvimento de aplicações. Como partes integrantes do ISODE, encontramos:

- Bibliotecas de funções que implementam os protocolos OSI das camadas de transporte, sessão e apresentação.

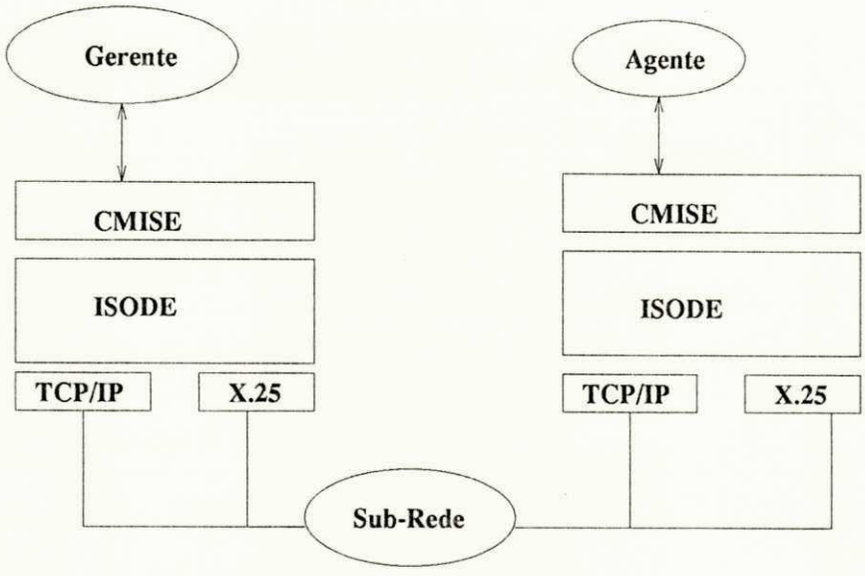

Figura 3.1: Interação Gerente/Agente utilizando o ISODE.

- Implementação dos elementos de serviço da camada de aplicação: ACSE, ROSE, RTSE.
- Implementação dos protocolos: VT (Virtual Terminal), FTAM (File Transfer, Access and Mangement), serviço de diretório X.500, serviço de correio eletrônico MHS e gateway FTAM/FTP.
- Compilador ASN.1, Gerador de chamadas de procedimentos para especificações de operações remotas, gerador de estruturas C para as especificações em ASN.a.

Os protocolos da camada de aplicação podem ser mapeados sobre TCP/IP através das camadas de transporte (TP0 e TP4), sessão e apresentação ou diretamente através do protocolo LPP (lightweight Presentation Protocol). A estrutura dos módulos do ISODE é vista na Figura 3.2.

Da experiência obtida com o ISODE, se constatou que ele pode ser usado para o desenvolvimento de aplicações distibuídas ou como suporte de migração para o modelo OSI.

3.2 Aplicações Distribuídas

A parte do ISODE mais utilizada na implementação do CMIS/CMIP descrita neste documento é a fornecida pelo "application cookbook", que é um ambiente de programação fornecido pelo ISODE para implementações baseadas em operações remotas, que são na realidade tarefas executadas em outras sistemas de forma transparente.

As operações remotas são consideradas parte da metodologia para a construção de sistemas distribuídos, facilitando o desenvolvimento da implementação. Isto também foi constatado com a experiência de implementação desenvolvida.

Para se compreender melhor a noção de operação remota utilizada pelo ISODE é necessário a compreensão dos componentes envolvidos nas operações remotas, que são os seguintes:

1. Tipos abstratos de dados;
2. Operações.

3.2.1 Tipos abstratos de dados

Um tipo abstrato de dados é um conceito utilizado para a descrição de estruturas de dados que são acessadas de forma bem definida, segundo as seguintes considerações:

- A estrutura de dados possui uma representação concreta no sistema local (ex: estrutura C), embora o seu tipo abstrato de dado correspondente seja definido de forma independente da implementação, sintaxe abstrata, que é formada pelas estruturas de dados que são trocadas na realidade pelo serviço;
- Existe um conjunto bem definido de regras fornecido pela notação de transferência abstrata utilizada para serializar a sistaxe abstrata de forma que as estruturas correspondentes aos tipos abstratos de dados possam ser transmitidos de forma não ambígua sobre a rede. Os tipos abstratos de dados são mapeados em sintaxe abstrata para então serem mapeados em sintaxe concreta que trata as propriedades de transmissão (geralmente "streams" de octetos);

- O acesso a um tipo abstrato de dados é feito por um conjunto unitário de ações chamadas operações, que definem o comportamento de uma instância de um tipo abstrato de dados.

3.2.2 Operações

Uma operação na sua forma mais primitiva é uma interação do tipo solicitação e resposta cujas interações envolvem os seguintes pontos:

- Chamada de uma operação, que consiste de:
  1. Um identificador de chamada, que possibilita identificar de forma única a solicitação no conjunto de operações sendo executadas.
  2. Um argumento relacionado com a operação.
  3. Um número relacionado com a operação, que identifica a operação em um conjunto de operações possíveis.
  4. Um identificador de "linked invocation", que é usado para identificar os resultados separados de uma mesma chamada.
- Resultado de uma operação em caso de sucesso é, formado por:
  1. O identificador de chamada, o resultado com uma solicitação feita previamente.
  2. Geralmente, algum tipo de argumento de retorno.
- Falha em uma operação resultando em erro, que consiste de:
  1. O identificador de chamada relacionado com a operação que falhou.
  2. Um número do erro que ocorreu.
  3. Geralmente algum tipo de argumento contido no retorno de erro.
- Se a operação não for executada por algum outro motivo (operação desconhecida, argumento de operação incorreto, etc), a solicitação da operação é rejeitada, contendo:

1. Um identificador de chamada correspondente à operação que deveria ter sido executada.
2. A razão pela qual a operação foi rejeitada.

Constata-se portanto que o CMIS/CMIP possui operações confirmadas (que enviam algum tipo de resultado) e operações não confirmadas.

3.3 Associação

Neste ponto é necessário algum tipo de ligação entre as entidade (processos) de aplicação que se comunicam. Esta ligação é chamada associação.

Uma associação, quando estabelecida, ocorre entre uma entidade iniciadora (aquela que solicita o estabelecimento da associação) e uma entidade respondedora (aquela que confirma ou rejeita o estabelecimento da associação). Existem associações nas quais somente o iniciador pode enviar solicitação de operações remotas e outras nas quais tanto o iniciador quanto o respondedor podem enviar solicitações. A associação utilizada pelo CMIS/CMIP é de classe 3, permitindo tanto ao iniciador da associação quanto ao respondedor fazer chamada de operações. Uma experiência bastante válida na implementação do CMIS/CMIP foi encontrar uma forma de estabelecer associações dentro do ambiente ROS que possibilitasse a ocorrência de associações de classe 3, pois toda a estrutura de implementação utilizando o ambiente ROS, contida nos exemplos do manual do usuário ISODE citava apenas o estabelecimento de associações de classe 1. Isto foi conseguido através da utilização da rotina RoSetService.

No estabelecimento da associação é definido o serviço que será utilizado na associação, que no caso do CMIS/CMIP corresponde às operações e notificação selecionadas.

3.4 Descrição do Ambiente ISODE

A versão do ISODE usada é a 7.0. A versão 8.0 do ISODE foi lançada recentemente, contudo sua utilização como plataforma de implementação não acarreta modificações no trabalho de implementação feito para o CMIS/CMIP na versão 7.0.

A versão 7.0 é totalmente desenvolvida em linguagem C e foi instalada numa rede de estações de trabalho SUN, disponível no laboratório do DSC na UFPB. O sistema operacional utilizado foi o SUNOS 4 compatível com UNIX V. Os serviços do ISODE estão disponíveis a nível de processo do usuário. O programa do usuário é ligado com as bibliotecas específicas que contêm todos os serviços providos pelo ISODE.

Para dar suporte aos níveis superiores do modelo OSI utilizando os serviços do TCP-IP, o ISODE utiliza uma interface que fornece serviços semelhante àqueles disponibilizados pelo protocolo de transporte OSI de classe 0. Esta interface, por sua vez, utiliza o TCP-IP como um serviço de rede orientado a conexão. Esta interface está especificada no RFC [24]. O ISODE, além de simular um serviço de transporte do tipo TP0, utiliza um protocolo para empacotamento destinado a separar as TSDUs (Transport Service Data Units) enviadas através do TCP, já que o TCP é confiável e orientado a conexão, enquanto o TP 0 é baseado num serviço de datagrama.

A chamada das primitivas dos níveis ocorre através de chamadas a procedimentos.

Os serviços disponíveis para a aplicação estão contidos nas seguintes bibliotecas:

- " **libacsap** ", implementa o serviço de controle de associação OSI - ACSE ("Association Control Service Element").
- " **librosap** ", que implementa as diferentes formas de utilização dos serviços de operações remotas OSI - ROSE ("Remote Operation Service Element").
- " **librtsap** ", implementa o serviço de transfeência confiável OSI - RTS ("Reliable Transfer Service Element").
- " **libpsap** ", implementa a sintaxe abstrata OSI e os mecanismos de transferência.

- " **libpsap2** ", implementa os serviços de apresentação OSI.
- " **libssap** ", implementa os serviços de sessão.
- " **libtsap** ", implementa um ponto de acesso de serviço de transporte OSI.
- " **libpsap2-lpp** ", implementa o protocolo de apresentação lpp para TCP-IP conforme RFC-1085.
- " **librosy** ", implementa uma biblioteca de funções utilizados no ambiente ROS para aplicações distribuídas.

3.5 Esquema de Endereçamento

Para que o ISODE seja capaz de usar os diversos protocolos de camadas de níveis inferiores, como é o caso do TCP/IP e o X.25, se torna necessária uma definição de endereçamento que forneça uma abstração de uma rede OSI única. A informação de endereçamento do ISODE contém uma indicação da pilha apropriada de transporte. Na hora do estabelecimento de uma conexão de transporte, a pilha disponível é selecionada. Uma aplicação pode rodar utilizando diversas pilhas simultaneamente

Esta estratégia permite que aplicações desenvolvidas possam ser executadas utilizando o ISODE, rodando em LANs com TCP/IP e se comuniquem com outras aplicações rodando em redes de longa distância com X.25 ou com pontes TP0. Isto facilita o desenvolvimento de aplicações para rodar em rede onde a pilha OSI não está totalmente disponível.

O ISODE utiliza arquivos de configuração (isoentities e isoservices) para simular o Serviço de Diretório, a solução encontrada foi a alocação de sub-domínio do espaço de endereçamento de rede OSI.

O endereço de rede do computador no qual a aplicação roda, identifica de forma única o computador na rede. Este endereço fica disponível no arquivo de configuração **isoentities** e é composto pelo designador do " host ", o nome do serviço oferecido,

- **Isotailor.**

O arquivo *isoentities* possui uma relação dos serviços disponíveis e seus respectivos endereços dentro da rede, fornecendo um mapeamento entre o endereço de apresentação e e informações sobre a entidade de aplicação. Estas informações formam um OID (Identificador de Objeto), que possibilita identificar univocamente um objeto dentro do mundo OSI, considerando, neste caso, objeto como endereço. Esta é uma forma mais fácil para o tratamento de endereços, pois ocorre o mapeamento entre um nome e o endereço propriamente dito.

3.6 Definição de Um Serviço no ISODE

Para cada serviço implementado no ISODE é necessário especificar:

- A sintaxe abstrata, que representa os tipos de dados trocados no escopo da aplicação distribuída. A sintaxe abstrata é identificada através de um OID.
- O contexto de aplicação, que contém informações que permitem a identificação dos elementos de aplicação (serviços comuns tipo ACSE, ROSE, etc) usados pelo serviço implementado. O contexto de aplicação é identificado por um OID.
- AEInfo, que é uma estrutura contendo informações para identificar univocamente uma serviço na rede. A AEInfo é, atualmente constituída pelo título do processo de aplicação (nome do processo de aplicação, que é mapeado com o endereço do objeto correspondente). Seu valor varia conforme o mecanismo utilizado para pesquisa do endereço, podendo conter um OID ou um nome.
- Endereço de apresentação, que possibilita a localização da entidade na rede.
- O programa local, que é a entidade provedora do serviço.

É necessário incluir informações sobre o processo que fornece o serviço no arquivo de configuração ISOBJECTS. Esta entrada facilita a referência à sintaxe abstrata e

ao contexto de aplicação utilizado. Estes dados geralmente estão relacionados a um OID, que é mapeado com um determinado nome com um significado mais claro para o programador.

O arquivo de configuração ISOSERVICE possui entradas contendo o nome do programa e a entidade associada ao mesmo, seletores e o nome do provedor do serviço. Ex: **psap/cmip 10 iso.cmip**. Neste caso, o psap é o serviço que suporta a entidade de aplicação cmip. Figura 3.4.

```
##############################################################################
#
# isoservices - ISODE Services Database
#
#Mappings between services, selectors, and programs
#
##############################################################################
#<provider>/<entity> <selector> <arg0> <arg1> ... <argn>
"tsap/mib"                #261                ros.cmip
```

Figura 3.4: Entradas no Arquivo Isoservices.

Para a identificação de serviços dentro do escopo de endereçamento OSI, foi reservada uma sub-árvore referente ao número 1.17., que é alocada para serviços que utilizam o ISODE, segundo M.T.Rose et al [5]. Desta forma os OIDs 1.17.2.n.1 e 1.17.2.n.2 são utilizados para a definição de contextos de aplicação e sintaxe abstrata, onde " n "é um número qualquer que esteja disponível. O CMIP já possui uma entrada pré-definida no arquivo isobjects. Figura 3.5.

O arquivo de configuração ISOENTITIES simula um serviço de diretório e possui entradas onde estão registrados o título do processo de aplicação e o endereço de apresentação do computador que o suporta. Figura 3.6.

```
##############################################################################
# ISO CMIP
##############################################################################

# iso standard 9596 abstractSyntax(0) cmip-pci(0)
"cmip pci"              1.0.9596.0.0

# iso standard 9596 cmip(2) version(1) acse(0) functional-units(0)
"cmip initialize pci"   1.0.9596.2.1.0.0

# iso standard 9596 cmip(2) version(1) acse(0) m-abort-source(1)
"cmip abort pci"        1.0.9596.2.1.0.1

# TEMPORARY application-context until 9596-2 gets its act together
"iso cmip"              1.17.1.11.2
```

Figura 3.5: Entradas no Arquivo Isobjects.

O processo tsapd é responsável pela inicialização interativa de conexão em todos os níveis usados (transporte, sessão,...) e também dispara o programa que implementa a entidade solicitada no pedido de conexão quando o serviço de diretório é usado. O processo tsapd fica escutando a porta 102 do TCP/IP e acusa um novo pedido de estabelecimento de associação ou eventos que ocorrem em uma associação previamente estabelecida. No caso da implementação do CMIS/CMIP, este processo não foi utilizado, pois na estrutura de implementação utilizada não foi possível utilizar somente uma estrutura baseada em cliente-servidor, onde um servidor fica somente aguardando solicitações de operações. No caso do CMIS/CMIP, os dois processos executando em máquinas separadas necessitam ser simétricos, pois o processo tanto tem que tratar notificações quanto operações.

Quando o serviço de diretório do ISODE está instalado, não são utilizados os arquivos de configuração para a pesquisa de endereço, que é obtido diretamente deste serviço.

```
##############################################################################
#
# isoentities - ISODE Application Entity Title Database
#
#Application Entity Titles as per ACSE
#
#This file takes the place of "real" directory services; OIDs are used
#for AETs, rather than Distinguished Names.
# Syntax:
#
#<host> <service> <aet> <paddr>

default                 mib                     1.17.4.0.20#261/
 isodeinf cmip 1.0.9596.0.0 #10/#1/#1/Internet=150.165.2.1+102
 isodeinf dir 1.17.4.1.2 #10/#1/#1/Internet=150.165.2.1+5102
 isodeinf dir1 1.17.4.1.3 #10/#1/#1/Internet=150.165.2.1+5103
```

Figura 3.6: Entradas no Arquivo Isoentities.

3.7 Ferramentas de Desenvolvimento

O ISODE fornece uma série de ferramentas para facilitar o desenvolvimento de novas aplicações. O conjunto destas fomam o "**Aplications Cookbook**", mencionado antes. [3].

Estas ferramentas possibilitam a tradução automática de especificações feitas em ASN.1 para uma série de rotinas em linguagem C, relacionadas à codificação e à decodificação das PDUs do protocolo, e criam as estruturas C relacionadas com os campos das primitivas de serviço especificadas no documento (iso 9595 e 9596). Estas rotinas geradas, e algumas tabelas, são incluídas no programa principal. A tradução e a geração de rotinas é feita através de uma biblioteca de programas e ferramentas ASN.1 (Figura 3.7). No capítulo 4, sobre a implementação do protótipo, é feito uma abordagem mais detalhada desta Figura considerando a especificação do CMIS/CMIP submetida.

A única limitação para a utilização do "Aplications Cookbook" é que a especificação

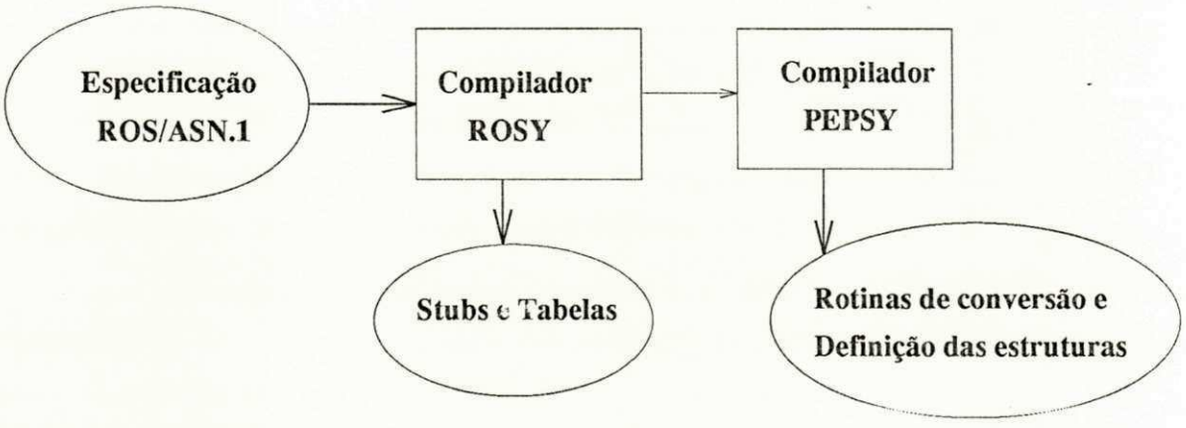

Figura 3.7: Interface do "Aplications Cookbook".

esteja em notação de operações remotas, descrevendo operações através do seu argumento, resultado e erro. As operações especificadas para o CMIS/CMIP são baseadas em operações remotas.

3.7.1 Compilador ROSY

O ROSY é um compilador que recebe como entrada as especificações baseadas em operações remotase fornecidas em sintaxe abstrata.

O programa tem como entrada um arquivo contendo a descrição do módulo de operações remotas em ASN.1, seguindo [16] e [17], e produz as estruturas básicas em linguagem C dos argumentos usados nas operações remotas e rotinas para codificação e a decodificação das PDUs.

O ROSY trata da descrição de cada operação descrita pelas cláusulas OPERATION, ERRORS e TYPES. A definição das operações é feita através de um nome que as identifica. Este nome deve começar com letras minúsculas. No caso do CMIP, podemos citar a operação m-get, e ser seguido da cláusula OPERATION. Em seguida, ocorre a definição dos argumentos, dos resultados e dos erros associados com a operação. A cláusula ARGUMENT refere-se à definição do tipo em ASN.1 que a operação espera como argumento. A cláusula RESULTS refere-se à definição de resultados que a operação deverá aguardar em caso de sucesso. ERRORS define os erros que a operação

pode retornar em caso de insucesso na chamada da rotina que implementa a operação. Cada operação terá um número associado que a identificará de forma unívoca dentro do conjunto de operações possíveis. O ROSY gera os arquivos Modulo.py; Modulo-ops.h; Modulo-ops.c e Modulo-stubs.c. Estes arquivos serão descritos de forma mais completa no capítulo 5, relacionado com a implementação do protótipo.

Dentro do ambiente fornecido pelos compiladores ROSY e PEPSY, muito do trabalho relacionado com a manipulação dos dados que formarão a PDU fica transparente para o programador.

Depois de estabelecer a associação, podem ocorrer solicitações de operações remotas através de duas interfaces:

- RyOperation (interface síncrona, que fica blocada até que a operação seja completada.
- RyStub (interface assíncrona, que empilha a chamada e libera. Quando ocorre um retorno da operação é chamada a rotina relacionada com o tratamento do resultado da mesma. Figura 3.8.

3.7.2 Compilador PEPSY

O PEPSY lê a descrição de um módulo em sintaxe abstrata e produz as tabelas que definem o mapeamento entre as estruturas C e os objetos em ASN.1 do módulo. O arquivo utilizado pelo PEPSY é gerado pelo ROSY. O compilador produz dois arquivos, um contendo tabelas de mapeamento e outro contendo estruturas em C. O PEPSY gera os arquivos CMIP-types.h e CMIP-tables.c.

O resultado/saída destes compiladores possibilita uma implementação bem mais simplificada. No caso do CMIS/CMIP, as rotinas para tratamento de codificação e decodificação foram fornecidas pelas rotinas de código geradas. O programador se preocupa basicamente com o tratamento da máquina de estados do protocolo, tratando

INTERFACE SÍNCRONA

```
int RyOperation (sd,ryo,op,in,out,response,roi)
    int sd;
    struct RyOperation *ryo;
    int op,
        *response;
    caddr_t in;
            *out;
    struct RoSAPindication *roi;
```

sd-identifica a associação
ryo- apontador para tabela RyOperation
op- o codigo da operacao a ser invocada
response- um indicador de sucesso da operacao
in- endereco da estrutura C relacionada com o argumento
out- estrutura C com a resposta
roi- estrutura que é atualizada em caso de falha na chamada.

INTERFACE ASSÍNCRONA

```
int RyStub (sd,ryo,op,id,linked,in,rfx,efx
            class,roi)
int sd;
struct RyOperation *ryo;
int op,
    id,
    *linked,
     class;
caddr_t in;
IFP rfx,
      efx;
struct RoSAPindication *roi;
```

id- identificador de invocação
class- classe de operação (sinc, assinc ou c/interrupção)
rfx- endereço da rotina a ser chamada em caso de sucesso
efx- end da rotina a ser chamada em caso de erro
in- argumento da operação.

Figura 3.8: Interface para Operações Remotas.

diretamente estruturas em C com os dados que formaram as PDUs. A máquina do protocolo CMIP lê uma fila de mensagens, analisa os dados da mesma, identifica a operação solicitada e entrega os dados para uma rotina de codificação, relacionada com a operação específica, que codifica e executa uma primitiva do ROSE para enviar a PDU formada, quando esta rotina libera o fluxo do programa, é ativado um bit de uma posição de uma arranjo de apontadores que indica, no caso de serviço confirmado que a operação esta aguardando algum resultado. Para o tratamento da chegada de primitivas do ROSE ou do ACSE, é registrada uma rotina que fica aguardando a ocorrência de eventos. Este processo é descrito com mais detalhes no capítulo 5.

Capítulo 4

CMISE e CMIP

4.1 Introdução

O CMIS/CMIP é o protocolo padrão especificado pela ISO para transferência de informações de gerenciamento entre os processos aplicação relacionados com gerenciamento de redes. Na comunicação para a troca de informação de gerenciamento entre dois processos pares, um funciona como agente enquanto o outro funciona como gerente (visto no capítulo 2). O CMIS/CMIP utiliza os serviços do ACSE e do ROSE. Ver Figura 4.1.

O CMIS/CMIP possui dois tipos de troca de informações de gerenciamento [1]:

- **Serviço de notificação de eventos**: usado pelos processos agentes para notificar o processo gerente a ocorrência de determinado evento. Como a estrutura do gerenciamento OSI é uma estrutura hierárquica, é possível utilizar o mecanismo de filtros para passar para o nível hierarquicamente superior apenas as notificações relacionadas com a competência deste nível.
- **Serviço de operação de gerenciamento**: relacionado com a execução de operações sobre os objetos gerenciados contidos na MIB. Ver Figura 4.2.

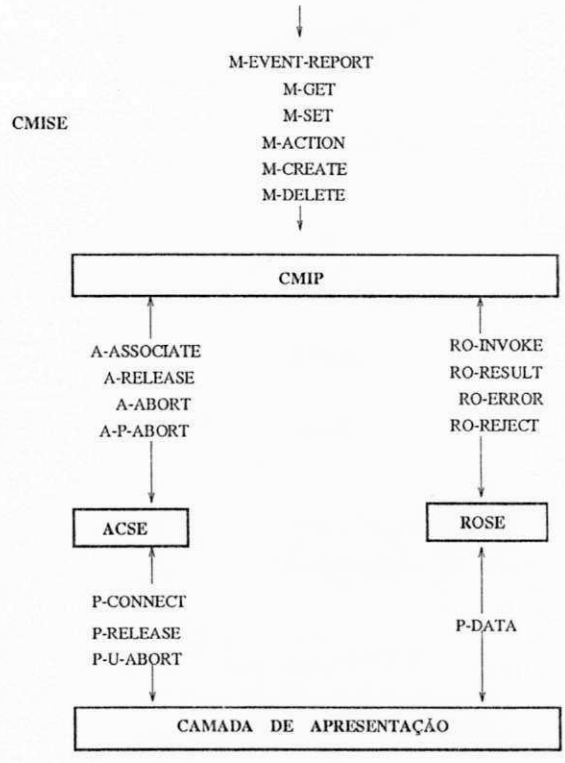

Figura 4.1: Primitivas de Serviço CMIS-CMIP.

4.2 IS 9595: Common Management Information Service Element (CMISE)

O Serviço de Informação de Gerenciamento Comum esta definido na norma ISO 9595 e fornece uma estrutura para procedimentos de gerenciamento comum que podem ser chamados remotamente, resultando na troca de informações de gerenciamento.

Os serviços especificados por esta norma consistem num conjunto de primitivas que podem ser trocadas entre os agentes e o gerente.

A norma contém as regras para a criação e uso destas primitivas. Estas primitivas quando recebidas pelo CMIP, são mapeadas em APDUs para serem transmitidas para o sistema par, através de primitivas do ROSE. Ver Figurafig42.

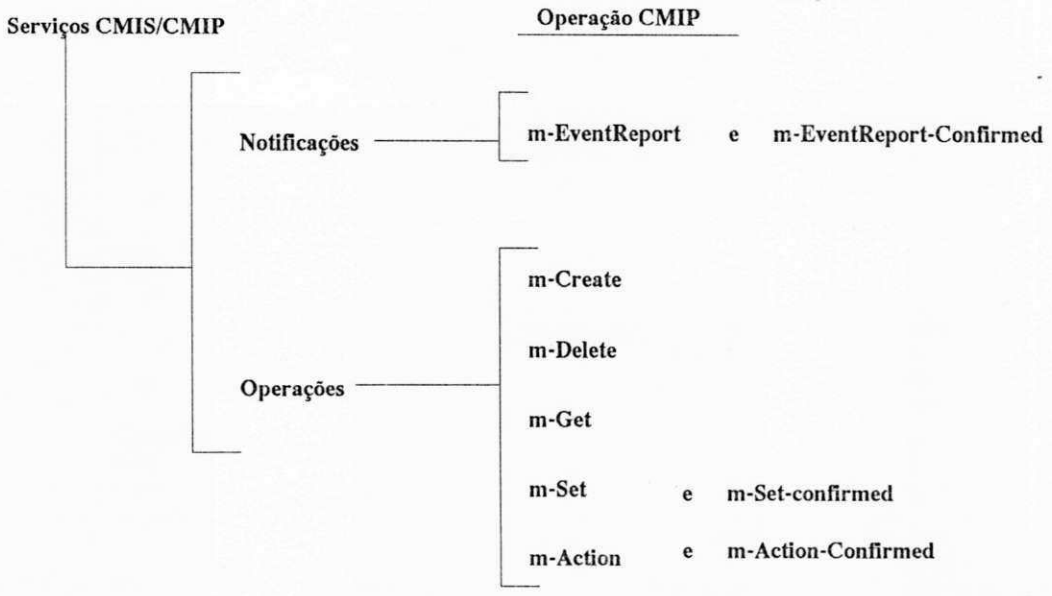

Figura 4.2: Serviços CMIS/CMIP.

4.2.1 Unidades funcionais

As unidades funcionais são usadas para descrever e definir as operações entre dois processos que se comunicam.

A organização dos serviços disponíveis através do CMISE é baseadasã no conceito de unidades funcionais, que definem a capacidade geral do protocolo e estão associadas com cada uma das primitivas de serviço do CMISE. As unidades funcionais estam relacionadas com os parâmetros das primitivas de serviço. Unidades funcionais é um meio prático para definição das operações que podem ocorrer entre dois processos que se comunicam [25].

No momento em que um usuário do CMISE deseja se comunicar com outro usuário par, ocorre a solicitação do estabelecimento de uma associação, onde as unidades funcionais são negociadas. O iniciador propõe as unidade funcionais que devem ser utilizadas na associação que está sendo estabelecida. Nos parâmetros contidos na resposta do A-ASSOCIATE podem vir parâmetros adicionais, que são na realidade as unidades funcionais que o respondedor propõe. Os detalhes dos parâmetros usados são descritos

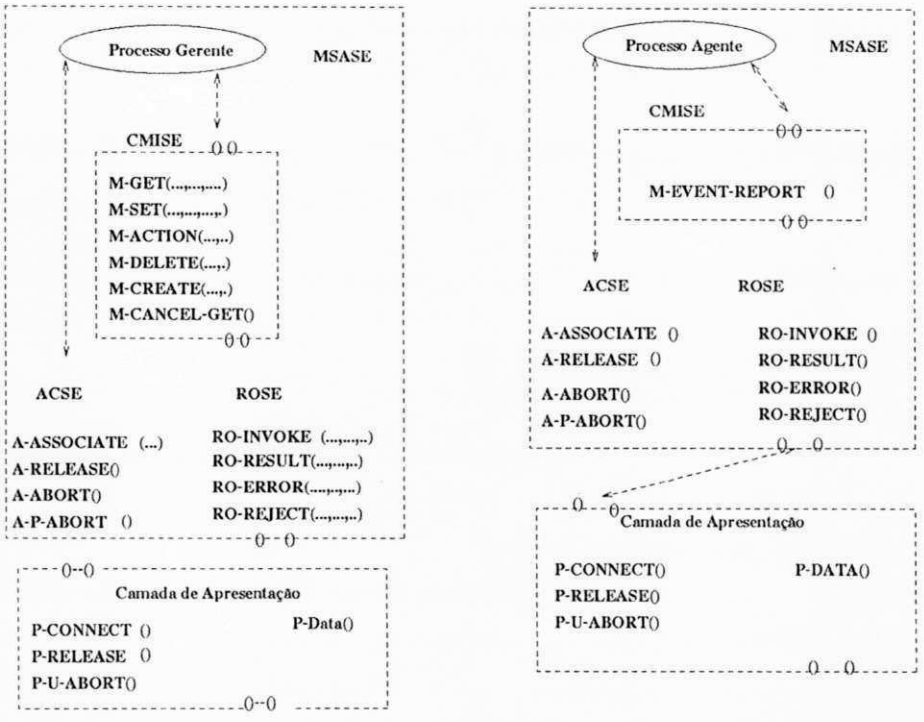

Figura 4.3: Primitivas do CMIS utilizadas normalmente pelos Agentes e Gerentes.

no contexto de aplicação a ser usado na associação.

No caso do CMIS/CMIP existem dois tipos de unidades funcionais: as unidades funcionais básicas ou do núcleo e as unidades funcionais adicionais.

- Unidade funcional do núcleo: estabelece parâmetro para respostas encadeadas quando selecionado e possui parâmetros que estabelecem o uso de escopo e sincronização.
- Unidade funcional adicional: Permite o uso de parâmetros não estabelecidos normalmente pela unidade funcional do núcleo.

A unidade funcional adicional estebelece o uso dos seguintes parâmetros:

1. Seleção de objetos múltiplos - o uso desta unidade funcional torna disponível os parâmetros de escopo e de sincronização nas operações M-EVENT-REPORT e M-CREATE.

2. Filtro - O uso desta unidade funcional abilita o parâmetro de filtro a ser usado nas operações M-EVENT-REPORT e M-CREATE.

3. Respostas múltiplas - Torna disponível o parâmetro de respostas encadeadas nas operações M-EVENT-REPORT e M-CREATE.

4. Extensão de Serviços - Torna outros serviços de apresentação disponíveis, além do P-DATA.

4.2.2 Estabelecimento de Associação

O serviço A-ASSOCIATE definido em [26] é chamado pelo usuário do serviço do CMISE para estebelecer uma associação com o processo com o qual deseja se conectar. Neste ponto os contextos de aplicação, apresentação e os requisitos de sessão e de transporte são estabelecidos. O estabelecimento de uma associação é o primeiro passo para a atividade de troca de informação de gerenciamento. O campo de dados do usuário do A-ASSOCIATE contém as seguintes informações:

- Unidades funcionais: São os parâmetros propostos para serem usados durante a fase de operações.

- Controle de acesso: não é definido, mas esta disponível caso o usuário do CMISE deseje estabelecer regras de acesso durante as operações e notificações.

- Informações do usuário: O valor deste parâmetro depende do contexto de aplicação e trata-se de qualquer informação adicional que o usuário deseja enviar para o usuário remoto através do CMIS.

Na fase de estabelecimento da associação vários ASEs podem trocar informações através do ACSE. Quando a entidade do protocolo iniciante recebe uma primitiva do tipo confirmação relacionada com o A-ASSOCIATE, indicando sucesso, a máquina do protocolo deve estabelecer a associação ou, em caso contrário deve ser rejeitada e a instância da entidade do protocolo deixa de existir. A figura 4.4 ilustra este princípio.

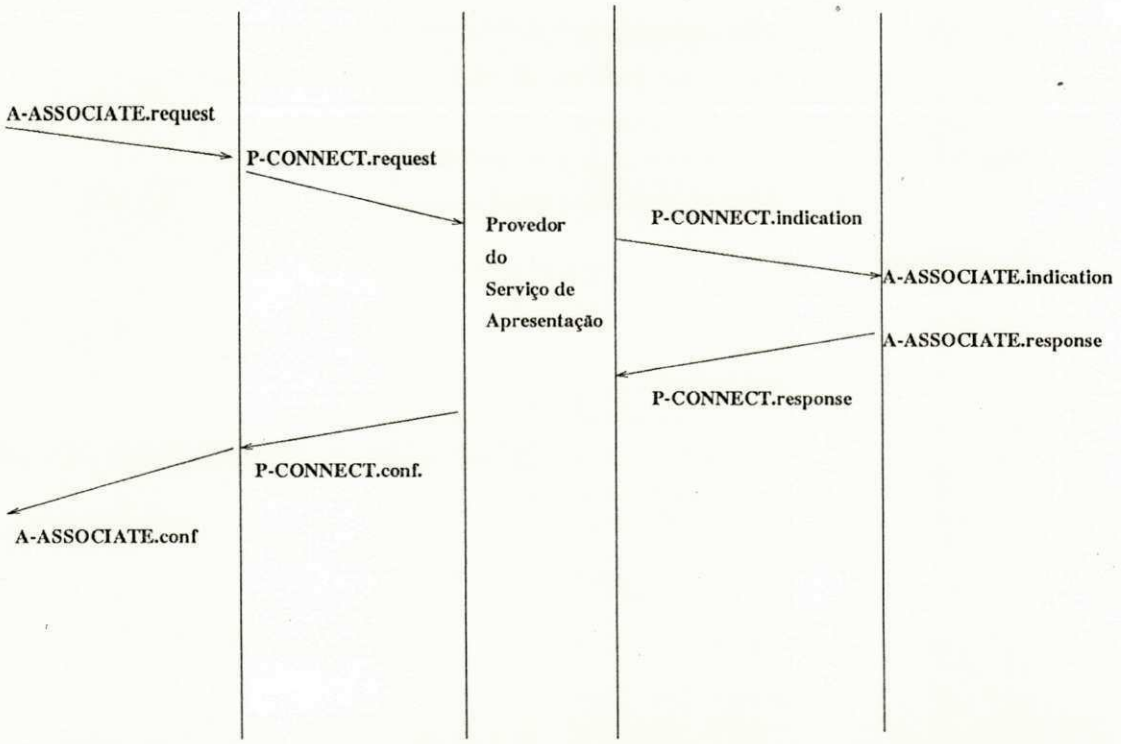

Figura 4.4: Estabelecimento de uma Associação de Aplicação.

Depois de estebelecida a associação tem-se a fase de troca de operação e notificação.

Uma associação após ser estabelecida pode ser liberada de forma abrupta com a utilização da primitiva A-ABORT.request ou pode ser liberada de forma negociada através da A-RELEASE.request, estes dois serviços pertencem ao ACSE.

4.2.3 Serviços do CMISE

Os serviços CMISE são implementados através de chamadas a primitivas que fornecem os serviços baseados em operações remotas, podem ser tanto operações simples, como operações de notificação. Um alarme da ocorrência de um determinado evento utiliza os serviços do CMISE para passar uma primitiva para o CMIP, e esta contém, em forma de parâmetro os argumentos utilizados para a formação da PDU do CMIP. Ver Figura 4.5. O CMIP forma a PDU, e toma as ações apropriadas, dependendo da primitiva e então executa uma primitiva do ROSE para passar os dados. O ROSE

por sua vez executa uma primitiva de apresentação, passando a PDU como dado do usuário para a camada de apresentação, e assim por diante.

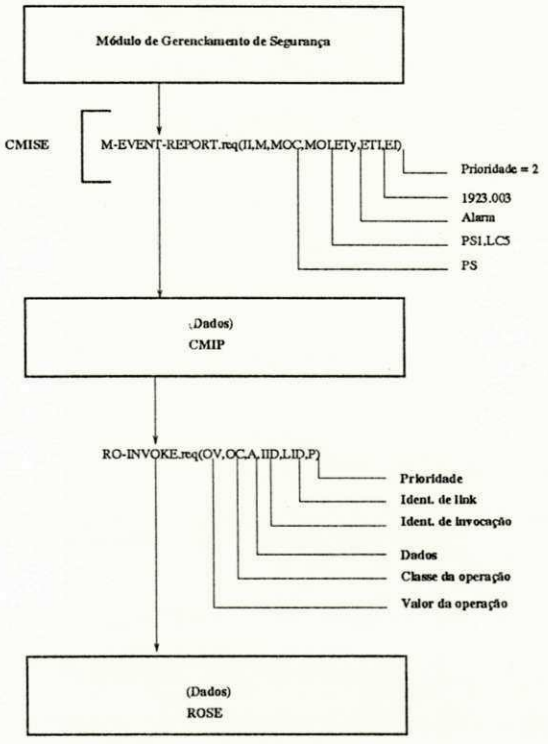

Figura 4.5: Utilização do CMISE pelo Gerenciamento de Alarme.

O CMISE fornece os seguintes serviços para um programa usuário da camada:

1. **Serviço M-GET** - Leva os seguintes parâmetros : (Figura 4.6)

   - *Invoke Identifier*: Identifica o número da invocação durante a operação M-GET. Identifica de forma única a invocação num determinado período de tempo. É um parâmetro necessário em todas as primitivas.
   - *Linked Identifier*: É utilizado no caso da ocorrência de respostas múltiplas e o seu valor é o mesmo do "invoke identifier" a fim de relacionar a invocação com as respostas.

[illegible] / BIBLIOTECA / [illegible]

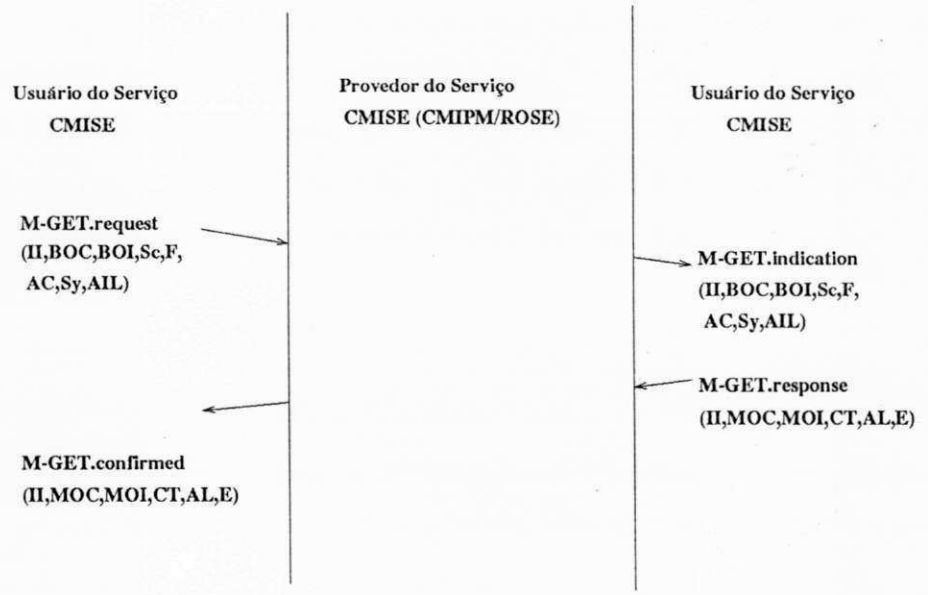

Figura 4.6: Serviço M-GET.

- *Base Object Class*: Especifica a classe do objeto gerenciado cujos atributos devem servir como parâmetros de resposta para a solicitação da operação M-GET. Na primitiva de indicação é utilizado como ponto de início para fins da aplicação do filtro.
- *Base Object Instance*: Especifica a instância do objeto gerenciado, também é utilizado como ponto de início para a aplicação do filtro.
- *Scope*: É utilizado para identificação da sub-árvore para a pesquisa. São permitidos três níveis de pesquisa: apenas o objeto base, o n-ésimo nível subordinado ao objeto base, o objeto base e todos os seus subordinados.
- *Filter*: Especifica teste de condição para a pesquisa do objeto. É aplicado ao objeto selecionado pelo escopo. Faz uso de operadores booleanos (AND,OR,NOT).
- *Access Control*: Este parâmetro não é especificado no padrão, mas deve existir para fins de controle. Não ocorre nas primitivas de resposta.
- *Synchronization*: Este parâmetro possibilita que o gerente informe ao agente como sincronizar as respostas enviadas.

- *Attribute Identifier List*: Indica quais os valores de atributos que devem compor a resposta.
- *Managed Object Class*: Especifica a classe do objeto no qual o evento deve ocorrer. Não é utilizado nas primitivas do tipo "request" e "indication", e é condicional nas de resposta e confirmação.
- *Managed Object Instance*: Especifica a instância do objeto na ocorrência do evento.
- *Current Time*: Contém a hora da geração da resposta. É opcional.
- *Attibute List*: Contém os identificadores de atributos e seus valores. Compõem a resposta e confirmação, dependendo do sucesso ou falha da operação.
- *Errors*: Contém possíveis condições de erro.

2. **Serviço M-CANCEL-GET** - É o único serviço que possui opção de cancelamento. Só existe para a operação GET porque poderia alterar a consistência dos dados da MIB se ocorresse em outras operações. Ver Figura 4.7. Possui os seguintes parâmetros:

   - *Invoke Identifier*: Identifica a invocação de uma operação M-CANCEL-GET.
   - *GET Invoke Identifier*: Identifica a operação M-GET que deve ser cancelada.
   - *Errors): Identifica alguma possível condição de erro ocorrida na operação.*

3. **Serviço EVENT-REPORT** - É utilizado para informar a ocorrência de eventos sobre os objetos gerenciados. Pode ocorrer de forma confirmada ou não confirmada. Ver Figura 4.8. Possui os seguintes parâmetros:

   - *Invoke Identifier*: Serve como identificador usado para a ocorrência de uma notificação.
   - *Mode*: Especifica se a operação é ou não confirmada.

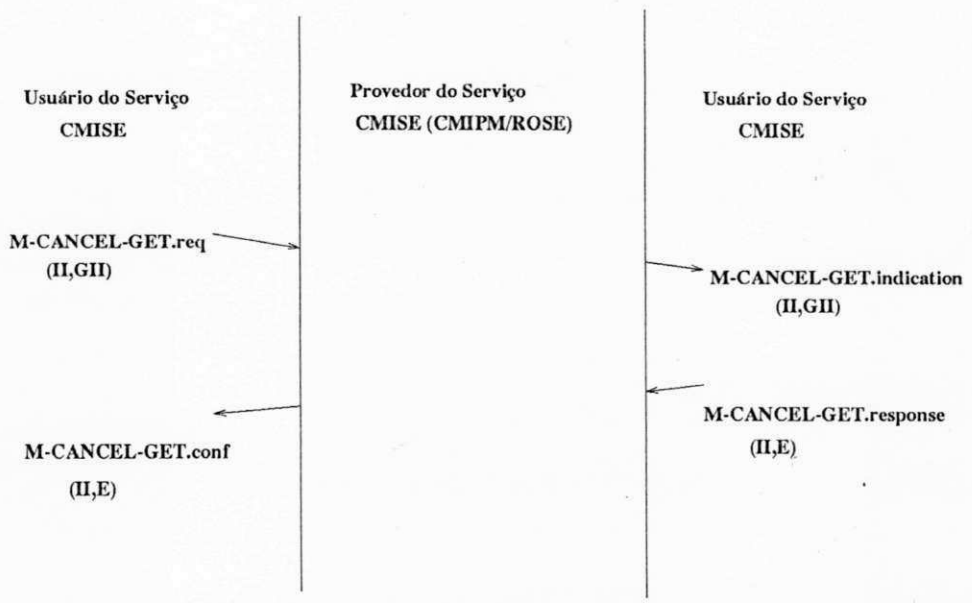

Figura 4.7: Serviço M-CANCEL-GET.

- *Managed Object Class*: Especifica a classe do objeto sobre o que está sendo informada a ocorrência do evento.
- *Managed Object Instance*: Informa a instância do objeto gerenciado para o qual está ocorrendo o evento.
- *Event Time*: Informa o tempo no qual o evento ocorreu.
- *Event Information*: Contém informações que o usuário acrescenta sobre o evento ocorrido. Este parâmetro não é definido. no padrão e é especificamente dependente da aplicação. É um parâmetro opcional.
- *Current Time*: Contém a hora na qual a resposta foi enviada.
- *Event Reply*: Contém a resposta da operação Event-Report em caso de sucesso ou não. É um parâmetro condicional.
- *Errors*: Contém informação sobre erros possíveis de ocorrer nesta operação.

4. **Serviço M-SET** - É usado pelo gerente para solicitar a um agente mudanças nos valores de atributos de um determinado objeto. Pode ser usado em serviço confirmado ou não confirmado. Os parâmetros para este serviço são:

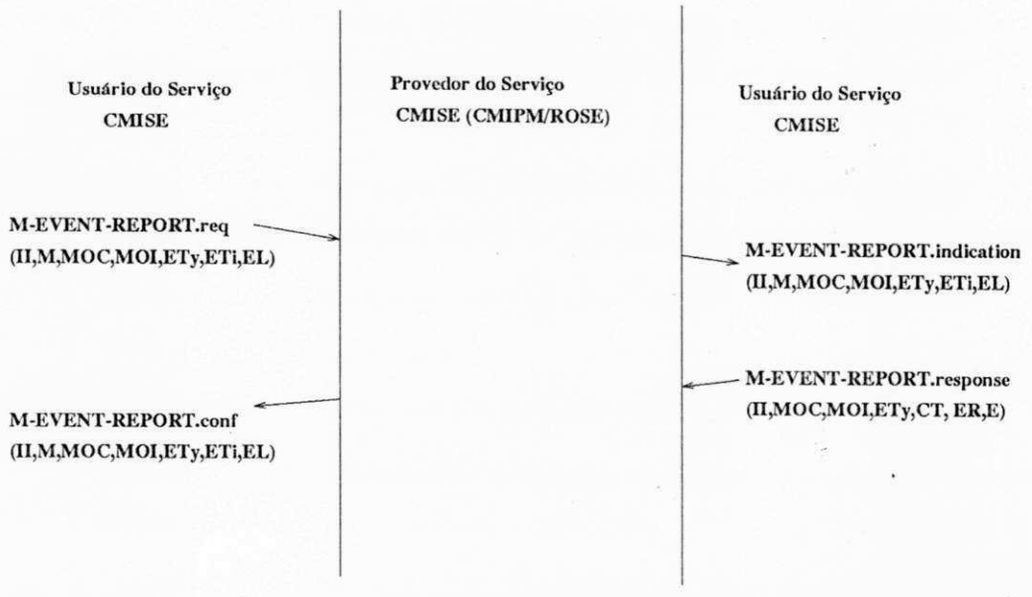

Figura 4.8: Serviço M-EVENT-REPORT.

- *Invoke Identifier*: Identifica a invocação do serviço.
- *Linked Identifier*: Identificador utilizado no caso da ocorrência de respostas múltiplas.
- *Mode*: Indica se o serviço solicitado é confirmado ou não.
- *Base Object Class*: Especifica a classe do objeto gerenciado no qual o evento deve recair, este parâmetro só é utilizado nas primitivas do tipo "Request" e "Indication".
- *Base Object Instance*: Especifica a instancia do objeto durante o evento.
- *Scope*: Identifica a sub-árvore de pesquisa. O nível 1 só inclui o objeto base; o nível 2 inclui até o n-ésimo objeto subordinado ao objeto base; e o nível 3 que inclui o objeto base e todos os seus subordinados.
- *Filter*: Especifica condições de teste para a pesquisa é aplicado após o escopo.
- *Access Control*: Faz o controle de acesso a objetos.
- *Synchronization*: Permite a sincronização das respostas.

- *Managed Object Class*: Especifica a classe do objeto gerenciado no qual o evento ocorreu. Só é utilizado nas primitivas do tipo "response"e "confirmation".
- *Managed Object Instance*: Especifica a instância do objeto durante a ocorrência do evento. Está presente nas primtivas do tipo "response" e "confirmation".
- *Attribute List*: São valores dos obtidos através dos identificadores de atributo.
- *Current Time*: Contém o tempo da geração da resposta.
- *Errors*: Contém quando for o caso, o erro ocorrido relacionado com uma tabela de erros definidos para a operação.

Como os parâmetros dos outros serviços são bastante semelhantes, só sera identificado o nome do parâmetro na descrição dos próximos serviços.

5. **M-CREATE**: Usado para criar uma nova representação de um objeto gerenciado. Só é usado no modo confirmado. Possui os seguintes parâmetros:

   - *Invoke Identifier*
   - *Managed Object Class*
   - *Managed Object Instance*
   - *Superior Object Instance*: Identifica a instância de um objeto já existente como superior a instância do novo objeto.
   - *Access Control*
   - *Reference Object Instance*: Especifica a instância do objeto gerenciado durante o evento. Só ocorre nas primitivas do tipo "Response" e "Confirmation".
   - *Attribute List*
   - *Current Time*
   - *Errors*

6. **Serviço M-DELETE**: É usado para deletar uma representação de uma instância de um objeto gerenciado. Só é usado no modo confirmado. Os parâmetros são os seguintes:

   - *Invoke Identifier*
   - *Linked Identifier*
   - *Base Object Class*
   - *Base Object Instance*
   - *Scope*
   - *Filter*
   - *Access Control*
   - *Synchronization*
   - *Managed Object Class*
   - *Managed Object Instance*
   - *Current Time*
   - *Errors*

7. **Serviço M-ACTION**: é usado para solicitar a execução de uma ação sobre um objeto gerenciado, pode ser usado no modo confirmado ou não confirmado. seus parâmetros são os seguintes:

   - *Invoke Identifier*
   - *Linked Identifier*
   - *Mode*
   - *Base Object Class*
   - *Base Object Instance*
   - *Scope*
   - *Filter*

- *Managed Object Class*
- *Managed Object Instance*
- *Access Control*
- *Synchronization*
- *Action Type*: Descreve o tipo de ação a ser executada no objeto gerenciado.
- *Action Argument*
- *Current Time*
- *Action Result*
- *Errors*

4.3 IS 9596: Common Management Information Protocol (CMIP)

A norma ISO 9596 estebelece a especificação do protocolo CMIP. O CMIP suporta os serviços mostrados na Figura 4.9.

Serviço	Modo
M-EVENT-REPORT	**Confirmado ou Não Confirmado**
M-GET	**Confirmado**
M-CANCEL-GET	**Confirmado**
M-SET	**Confirmado ou Não Confirmado**
M-ACTION	**Confirmado ou Não Confirmado**
M-CREATE	**Confirmado**
M-DELETE	**Confirmado**

Figura 4.9: Serviços Suportados pelo CMIP.

4.4 Exemplo de Uma Operação CMIP

A operação de notificação é uma das mais importantes do CMIP. A operação ocorre entre duas entidades CMIP conhecidas por CMIPM (Common Management Information Protocol Machine), que implementam a própria máquina do protocolo. Ver Figura 4.10.

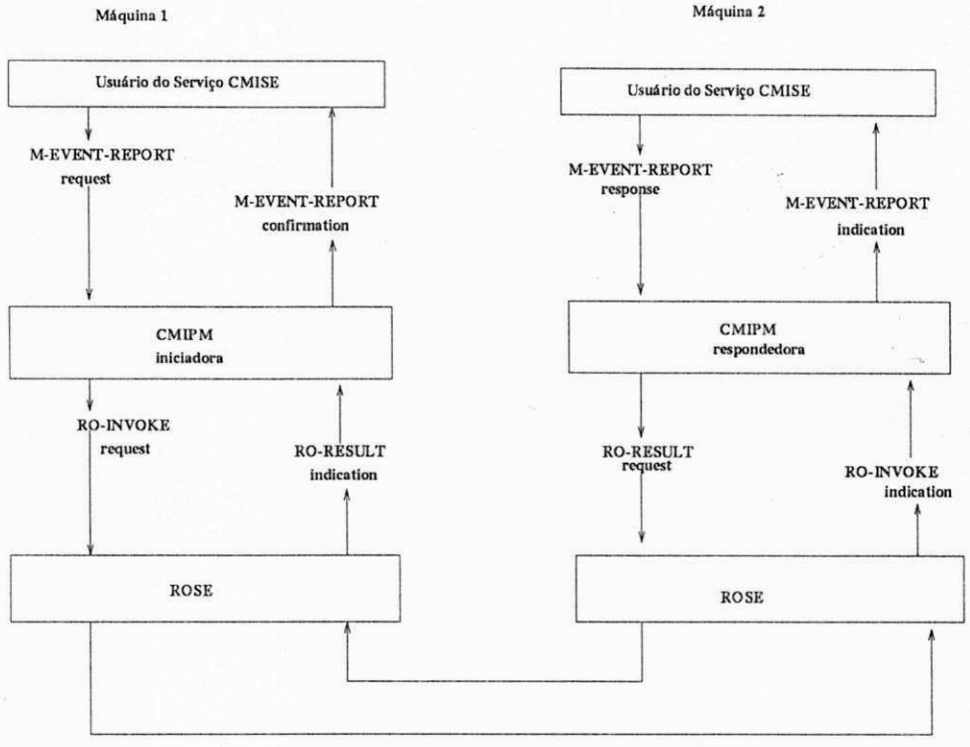

Figura 4.10: Exemplo de operação CMIP.

Uma operação de notificação confirmada (m-EventReport-Confirmed) segue os seguintes passos:

1. A CMIPM recebe uma primitiva M-EVENT-REPORT de um usuário do serviço CMISE. Os parâmetros da primitiva contêm valores que permitem a identificação desta operação em particular, informações sobre o objeto gerenciado, o tipo de evento que está sendo informado, a hora de sua ocorrência e outras informações sobre o evento.

2. AS CMIPM examina a primitiva e constroi uma PDU de aplicação m-EventReport (APDU).

3. Então envia a APDU para o ROSE através da primitiva de serviço RO-INVOKE request.

4. A CMIPM receptora recebe a APDU através de uma primitiva ROSE RO-INVOKE indication.

5. Se os dados estiverem bem formados, a CMIPM envia uma primitiva M-EVENT-REPORT indication para o usuário do serviço CMISE receptor.

6. O usuário do serviço CMISE envia uma primitiva do tipo M-EVENT-REPORT response para a sua CMIPM. A primitiva contêm valores para identificação da operação específica, informação sobre o objeto gerenciado, o tipo de evento sendo informado, a hora da geração da resposta e algum parâmetro relacionado com o resultado.

7. A CMIPM usa estes parâmetros para construir uma nova APDU.

8. A CMIPM envia a APDU para o ROSE através de uma primitiva do tipo RO-RESULT request.

9. A CMIPM iniciadora da operação recebe a APDU através de uma primitiva RO-RESULT indication do ROSE.

10. Se a aPDU estiver bem formada, a CMIPM executa uma primitiva M-EVENT-REPORT confirmation para o usuário CMISE. Neste ponto a operação de notificação confirmada é encerrada.

Capítulo 5

Implementação do Protótipo

5.1 Introdução

O objetivo deste capítulo é apresentar a estrutura da implementação dos serviços do protocolo OSI - CMIS/CMIP assim como as estruturas de dados envolvidas.

O uso de operações remotas é uma técnica bem conhecida para a implementação de aplicações distribuídas. O modelo OSI oferece uma notação poderosa para a especificação das interações externas desses sistemas através de operações remotas especificadas em ASN.1. Escolhemos o ISODE como plataforma de trabalho pela experiência pioneira de utilização na UFPB e também pelas facilidades de implementação que o mesmo oferece em comparação com modelos de implementação utilizando "SOCKETS".

A estrutura do código segue a organização fornecida pelo ambiente ROS do ISODE, onde rotinas geradas pelos compiladores ASN.1 são introduzidas no programa principal. Este modelo é composto pela organização dos arquivos de código fornecidos pelo programador e dos arquivos de código fornecidos pelos compiladores do ISODE para a implementação, as estruturas de dados em C utilizadas para a formação das PDUs e a interface com as camadas adjacentes.

Todas as operaçẽs do serviço são completamente dependentes do ROSE devido a especificação do protocolo ser baseada no conceito de operações remotas [ISO 9072-2]

Figura 5.1. A especificação do CMIP é completamente dependente dos serviços do ROSE (notação ROS-operação remota), e devido a isto, a implementação do protocolo não necessita de tabelas de estado, listas de eventos, predicados ou tabelas de ação [25].

```
-- CMISE operations

-- In the following operations, the argument type is mandatory in the
-- corresponding ROSE APDU.
-- Action operations (M-ACTION)

m-Action OPERATION
 ARGUMENT ActionArgument
 ::= 6

m-Action-Confirmed OPERATION
 ARGUMENT ActionArgument
 RESULT   ActionResult
 -- This result is conditional; for conditions see [Recommendation X.CMIS.0 |
 -- ISO/IEC 9595] clause 8.3.3.2.9.
 ERRORS { accessDenied, classInstanceConflict, complexityLimitation,
  invalidScope, invalidArgumentValue, invalidFilter, noSuchAction,
  noSuchArgument, noSuchObjectClass, noSuchObjectInstance,
  processingFailure, syncNotSupported }
 LINKED { m-Linked-Reply }
 ::= 7

...
```

Figura 5.1: Exemplo de Especificação de Operação do CMIP

5.2 Estrutura da Implementação

A estrutura da implementação do CMIP se baseia no ambiente ROS oferecido pelo

ISODE, que fornece uma estrutura pré-definida para a utilização das rotinas geradas pelos compiladores ASN.1 - Figura ??.

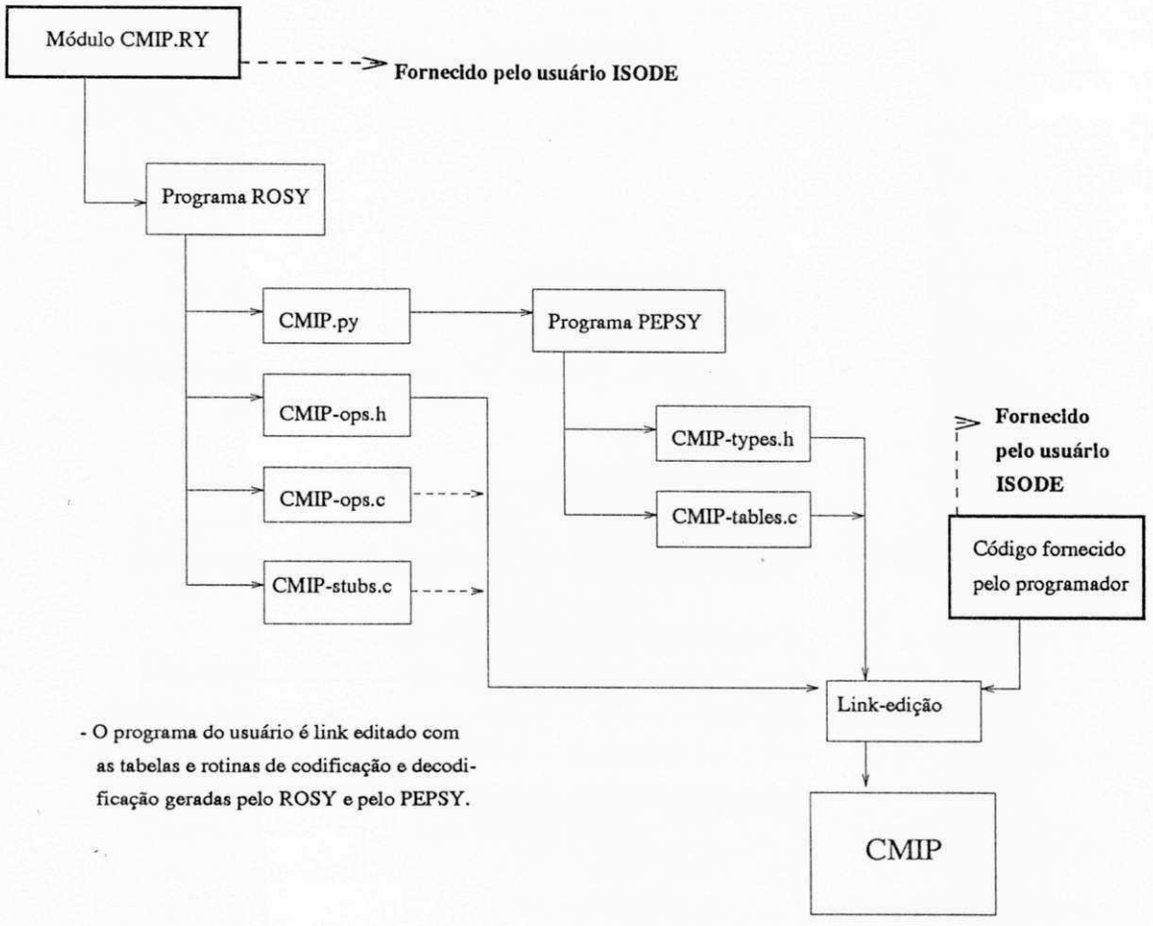

Figura 5.2: Utilização dos compiladores ROSY e PEPSY.

Os compiladores (ROSY e PEPSY) fornecem as rotinas para codificação e decodificação das APDUs para a implementação do protocolo e também geram interfaces estruturadas que possibilitam a chamada dos serviços do ROSE de forma simples e eficiente. Estas facilidades são conhecidas no ISODE como ambiente ROS(baseado nos serviços do ACSE, ROSE e nos compiladores ROSY e PEPSY). O ponto de partida para o processo de codificação de um protocolo qualquer no ISODE esta na criação de um arquivo contendo a especificação do protocolo em ASN.1 a ser submetido aos

compiladores, e que no caso do CMIP foi obtido com o padrão ISO 9596 (Common Management Information Protocol Specification).

A implementação esta organizada em blocos de código que são chamados após a inicialização de uma associação gerenciada através dos serviços A-ASSOCIATE, A-RELEASE, A-ABORT e A-P-ABORT disponíveis no ACSE. Para a implementação das operações remotas do CMIP, são utilizados RO-INVOKE, RO-RESULT, RO-ERROR e RO-REJECT-U, fornecidos pelo ROSE [ISO 9072-2]. O ROSE utiliza o serviço P-DATA da camada de apresentação definido em [ISO 8822] para a transferência das PDUs relacionadas com as primitivas de serviço. O CMIP utiliza associações de classe 3 e operações remotas de classe 1 (sincronas) podendo ser também de classe 2 (assincronas).

5.3 Interfaces Com Camadas Adjacentes

5.3.1 Interface com o ACSE

Antes de qualquer operação do protocolo ser solicitada, é necessário o estabelecimento de uma associação entre as duas entidades que fornecem os serviços.

Para o estabelecimento e o encerramento de uma associação são usados os serviços do ACSE através do ambiente ROS. A Figura 5.3 mostra um trecho do programa principal que trata do estabelecimento de associação de forma simplificada. A função AcAssocRequest(), quando executada equivale a uma primitiva A-ASSOCIATE do tipo request.

É neste nível que são determinados o endereço e as informações necessárias da entidade de aplicação desejada para o estabelecimento da associação. Estas informações são obtidas através de funções que pesquisam o endereço de apresentação a partir do designador e do qualificador do serviço nos arquivos de configuração (isoservices, isobjects isoentities) que são usados, pois o serviço de diretório não foi disponibilizado na instalação da versão da plataforma utilizada.

```
if((aei=_str2aei(argv[1],myservice,mycontext,isatty(fileno(stdin)),NULLCP,NULLCP))==NULLAEI)
  printf("\nImpossivel fornecer servico\n");
if((pa=aei2addr(aei))==NULLPA)
  printf("\nfalha na traducao de endereco\n");
if((ctx = ode2oid (mycontext)) == NULLOID)
  printf("\ndescritor de objeto desconhecido: %s\n",mycontext);
if ((ctx=oid_cpy(ctx))==NULLOID)
  printf("\nFora de memoria\n");
if((pci=ode2oid(mypci))==NULLOID)
  printf("\nDescritor de object desconhecido: %s",mypci);
if((pci=oid_cpy(pci))==NULLOID)
  printf("\nFora de memoria\n");
pc->pc_nctx=1;
pc->pc_ctx[0].pc_id=1;
pc->pc_ctx[0].pc_asn=pci;
pc->pc_ctx[0].pc_atn=NULLOID;

if((sf=addr2ref(PLocalHostName()))==NULL){
 sf=&sfs;
 (void)bzero((char *)sf,sizeof *sf);
}
if (AcAssocRequest(ctx,NULLAEI,aei,NULLPA,pa,pc,NULLOID,0,ROS_MYREQUIRE,      ...)==NOTOK)
printf("\nErro no A-ASSOCIATE.REQUEST\n");
if(acc->acc_result != ACS_ACCEPT)
printf("\nAssociacao Rejeitada: [%s]\n",AcErrString(acc->acc_result));
sd=acc->acc_sd;
ACCFREE(acc);
if(RoSetService(sd,RoPService,roi)==NOTOK)
 printf("\nFalha no acesso de APRESENTACAO\n");
```

Figura 5.3: Estabelecimento de Associação no Ambiente ISODE.

No estabelecimento da associação foram usadas as funções AcAssocRequest e AcRel-Request, que equivalem às primitivas A-ASSOCIATE.REQUEST e A-RELEASE.REQUEST. A função AcAssocRequest possui dezesete parâmetros, que são:

1. Identificador da associação.
2. Contexto de aplicação.
3. AEInfo do iniciador e do respondente.
4. Dados inicias representador em um EP.
5. Qualidade de serviço.

6. Lista do contexto de apresentação proposto, que é representado por uma estrutura contendo o identificador do contexto, o OID da sintaxe abstrata, o IOD da sintaxe de transferência (neste caso sintaxe gerada pelas BER) e um campo para resultado pelo serviço de apresentação para aceitar ou rejeitar o contexto proposto.

7. Contexto " Default " de apresentação.

8. Identificador da unidades funcionais desejadas nos níveis de apresentação e de sessão.

9. E outros.

O estabelecimento de associação ocorre no momento em que uma das partes necessita utilizar os serviços do CMIP para transmitir alguma informação para a outra. É neste ponto que ocorre a negociação de certos parâmetros contidos nas unidades funcionais do protocolo, infomações que permitem a identificação das unidades funcionais adicionais, versão do protocolo, mecanismos de controle de acesso, e informações específicas do usuário. A Figura 5.4 mostra uma parte da especificação em ASN.1, de forma simplificada, para estas informações.

```
-- Informações do usuário no serviço A-ASSOCIATE

CMIPUserInfo ::= SEQUENCE
                 { protocolVersion   [0] IMPLICIT ProtocolVersion
                   functionalUnits   [1] IMPLICIT FunctionalUnits
                   accessControl     [2] EXTERNAL OPTIONAL
                 userInfo            [3] EXTERNAL OPTIONAL
                 }

FunctionalUnits ::= BIT STRING
                    { multipleObjectSelection   (0),
                      filter                    (1),
                      multipleReply             (2),
                      extendedService           (3),
                    }
```

Figura 5.4: Informações do CMIP contidas no A-ASSOCIATE.

As funções contidas na *libacsap* para acesso ao ACSE, permitem a negociação do

contexto a ser utilizadp na associação entre as duas entidades de aplicação. Esta chamada ao AcAssocRequest equivaleco a enviar uma primitiva A-ASSOCIATE.REQUEST, cujo retorno esta contido na estrutura aci ou acc, que são parâmetros da rotina acima e dependem da função obter ou não sucesso. Quando a chamada desta função não obtém sucesso, o parâmetro aci contém apontador para a estrutura AcSAPindication que é atualizada com o motivo do erro. Em caso de erro, a estrutura AcSAPabort, que é um sub-campo da AcSAPindication, trás a fonte geradora e o número do erro, que é pesquisado na tabela de códigos de erros do ACSE, para então ser passado para a entidade que solicitou a associação. No caso da chamada obter sucesso, o parâmetro da estrutura acc, que é um apontador para a estutura AcSAPconnect, é atualizado e representa a chegada de uma primitiva A-ASSOCIATE.CONFIRMATION, confirmando o estabelecimento da associação, Figura 5.5. Antes de se chamar a rotina AcAssocRequest, é necessário a recuperação de algumas informações contidas nas bases de dados do ISODE sobre a entidade de aplicação chamada, como o contexto de aplicação a ser negociado, o endereço de apresentação, o pci, etc. Estas informações são dados que estam disponíveis a partir de funções de bibliotecas contidas no ISODE.

A entidade que recebe a solicitação para o estabelecimento de associação, chama a função AcInit, que recebe os parâmetros enviados pelo iniciador da associação através de um estrutura acSAPstart e recupera parâmetro contendo o estado do ACSE. A chamada desta função equivale a ocorrência de uma primitiva A-ASSOCIATE.INDICATION, que é analisada conjuntamente com o estado do ACSE. Em caso de sucesso, é feita uma chamada a função AcAssocResponse que possui desesseis parâmetros de corresponde ao envio de uma primitiva A-ASSOCIATE.RESPONSE, cujos parâmetros são listados abaixo:

1. Descritor da associação.
2. Indicador de que a associação foi aceita ou rejeitada.
3. Um código para diagnóstico.
4. A confirmação do contexto de aplicação a ser usado.

```
struct AcSAPindication {
  int aci_type;
#define ACI_FINISH 0x00
#define ACI_ABORT 0x01
 union {
          struct AcSAPfinish aci_un_finish;
          struct AcSAPabort aci_un_abort;
        } aci_un;
#define aci_finish      aci_un.aci_un_finish
#define aci_abort       aci_un.aci_un.abort
};

. . .
struct AcSAPabort {
 int aca_source;
 int aca_reason;
 int aca_ninfo;
 PE aca_info[NACDATA];
#define ACA_SIZE 512
    int aca_cc;
    char aca_data[ACA_SIZE]
};
. . .
```

```
struct AcSAPconnect {
  int acc_sd;              -ident. da associação
  int acc_result;          -resposta da associação
  int acc_diagnostic;      -diagnóstico
  OID acc_context;         -nome do contexto de aplicação
  AEInfo acc_respondtitle; -info. sobre a entidade respondedora
  struct PSAPconnect acc_connect;
  int acc_ninfo;           -algum dado de inicialização
  PE acc_info[NACDATA]
};
```

- Estas estruturas são manipuladas em caso de confirmação ou erro na solicitação de estabelecimento de associação.

Figura 5.5: Estrutura AcSAPindication e AcSAPabort.

5. Informações sobre a entidade respondente.

6. Apontador para uma estrutura AcSAPindication, que é atualizada em caso de falha na chamada da função.

Após o estabelecimento da associação, é informado ao ROSE que a associação estabelecida é de tipo 3, possibilitando tanto ao iniciador da associação quanto à entidade respondedora enviar e receber solicitações de operações. Na implementação, tanto o iniciador da associação quanto a entidade respondedora podem enviar solicitações das operações remotas e notificações, assim como receber confirmações. É feita então uma chamada a função RoSetService que estabelece os serviços do ROSE que devem ser usados na associação.

5.3.2 Interface com o ROSE

Todas as PDUs do CMIP são definidas em ASN.1, com as operações definidas através das macros externas OPERATION e ERROR [ISO 9072-1]. A definição do CMIP é completamente dependente do ROSE, e devido a isto, a implementacao do protocolo não necessita de tabela de estado, lista de eventos, predicados ou tabelas de ação.

O núcleo do CMIP que trata das operações remotas pode ser dividido em duas partes: uma que aguarda a ocorrência de eventos na associação estabelecida, esperando primitivas do ROSE e outra que recebe primitivas da aplicação usuária e chama as rotinas que implementam as operações remotas através do ROSE.

A chegada de uma primitiva de solicitação de operação é passada para o CMIP através de uma área de memória compartilhada e controlada por semáforos. Esta área é gerenciada através de duas funções: recebe() e envia(). A aplicação de gerenciamento que usa esta implementação do CMIP, tem acesso a duas filas relacionadas com os semáforos definidos para controlar as áreas de memória usadas. Uma das áreas é usada para receber primitivas das aplicação e outra para entregar primitivas para a aplicação. A recepção de primitiva para solicitação dos serviços do CMIP é feito através de leituras periódicas da rea de memória compartilhada. Esta área passa um "buffer" preenchido com a primitiva que solicita um dos serviço do CMIP. O primeiro campo sempre contém o número da operação que esta sendo requisitada, seja uma operação remota ou uma notificação. Este identificador de operação se relaciona com o número da operação especificada em ASN.1 e é utilizado para chamar a função de tratamento da operação específica. Ver Figura 5.6. Dentro desta função é feita a leitura dos dados para a formação da PDU.

Além da sinalização da operação o CMIP também recebe através do " buffer " os parâmetros que formarão a PDU que será passada para o ROSE. O " buffer " é convertido em um vetor que é tratado por rotinas específicas a cada operação. A comunicação do CMIP com o ROSE é feita através de chamadas a rotinas fornecidas pelo (ROSY e PEPSY) que se encarregam da codificação dos parâmetros da PDU específica que após codificados são passados para o ROSE, através de funções implementadas na *librosy*.

```
if(recebe(1,buffer,sizeof(buffer))==NOTOK)
 printf("\n.....");              /* Faz a leitura da primitiva */
 else {
  if(str2vec(buffer,vec)<1)
     continue;
  switch (vec[0]) {
     case '6':
          do_mAction(sd);
     case '7':
          do_mActionConfirmed(sd);
     case '8':
          do_mCreate(sd);
     case '9':
          do_mDelete(sd);
     case '0':
          do_mEventReport(sd);
     case '1':
          do_mEventReportConfirmed(sd);
     case '3':
          do_mGet(sd);
     case '4':
          do_mSet(sd);
     case '5':
          do_mSetConfirmed(sd);
            }
/* A escolha dentro do switch se relaciona com a   */
/* numeracao das operacoes no codigo ASN.1         */
```

Figura 5.6: Rotinas de Leitura de Dados do CMIP.

Entre os parâmetros passados estão: um identificador de associação, um identificador de invocação, parâmetros específicos da primitiva, um apontador para a estrutura **roi**, que funciona de forma semelhante à estrutura apontada pelo parâmetro aci da interface com o ACSE Figura 5.7. Estas rotinas estão contidas no arquivo CMIP-stubs.c.

Após o recebimento da primitiva da aplicação e a analise da operação solicitada, é feita uma chamada a uma rotina determinada para o tratamento da operação específica. Quando a rotina da operação é chamada são passados alguns parâmetros, dentre os quais, um identificador de associação e um "buffer" contendo parâmetros específicos da primitiva recebida. Dentro da rotina específica da operação é feita uma conversão dos dados contidos no "buffer" para preencher a estrutura em C relacionada com a

```
stub_CMIP_mGet(sd,id,arg,do_mGetResultado,do_GetErro, ...)
stub_CMIP_mSet(sd,id,NULLID,NULLID,...)
stub_CMIP_mSetConfirmed(sd,id,do_mSetConfirmedResultado,do_mSetConfirmedErro,...)
stub_CMIP_mAction(sd,id,NULLID,NULLID,...)
stub_CMIP_mActionConfirmed(sd,id,do_mActionConfirmedResultado,do_mSetConfirmedErro,...)
stub_CMIP_mEventReport(sd,id,NULLID,NULLID,...)
stub_CMIP_mEventReportConfirmed(sd,id,do_mEvent...Resultado,do_mEvent...Erro,...)
stub_mCreate(sd,id,do_mCreateResultado,do_mCreateErro,...)
stub_CMIP_mDelete(sd,id,do_mDeleteResultado,do_mDeleteErro,...)
```

Figura 5.7: Interface com o ROSE.

PDU da operação solicitada (ver anexo 1). Após a alocação da estrutura de dados e o seu preenchimento com os dados lidos do "buffer", é feita uma chamada à função que implementa a solicitação das operações ao ROSE, que trata conjuntamente da codificação da PDU enviada e da decodificação dos dados que são entregues para as rotinas de tratamento de resultado ou de erro. Esta função também faz a chamada automatica às rotinas que tratam o resultado ou erro no caso de operações confirmadas Figura 5.8.

```
do_mGet(sd, buffer);
...
{ arg=(struct type_CMIP_GetArgument calloc(1,sizeof *arg)
  ...
  /* Tratamento do buffer e atribuição de arg */
  switch(resultado=stub_CMIP_mGet(sd,id=RyGenID(sd),arg,do_GetResultado,do_GetErro,ROS_RYNC,roi);
    case NOTOK:
            /*indica problemas na operação */
    case OK:
            break;
    ...
  }
  free_CMIP_GetArgument(arg);
  return OK;    }
```

Figura 5.8: Chamada das Rotinas de Operações Remotas.

São passados os seguintes parâmetros para a função:

- Identificador da associação.

- Identificador da chamada (evita duplicações).
- Apontador da estrutura C com os dados da PDU.
- Endereço da rotina para tratamento de resultado.
- Endereço da rotina para tratamento de erro.
- Parâmetro que especifica a classe da operação (sync ou async).
- Apontador para uma estrutura que é atualizada em caso de falha na chamada da rotina.

São definidas rotinas para o tratamento dos eventos de chegada de primitivas de indicação relacionadas com os serviços fornecidos. Estas rotinas estão agrupadas em uma **tabela DISPATCH** - Figura 5.9, e são registradas através da função **RyDispatch**. Estas rotinas são chamadas através da função iserver-wait descrita abaixo.

```
. . .
static struct dispatch disp[] = {

    "mSet", operation_CM_mSet, op_mSet,
    "mSetConfirmed", operation_CM_mSetConfirmed, op_mSetConfirmed,
    "mAction", operation_CM_mAction, op_mAction,
    "mEventReport", operation_CM_mEventReport, op_mEventReport,
    "mEventReportConfirmed", operation_CM_mEventReportConfirmed, op_mEventReportConfirmed,
    "mCreate", operation_CM_mCreate, op_mCreate,
    "mDelete", operation_CM_mDelete, op_mDelete,
    "mGet", operation_CM_mGet, op_mGet,

    NULL  };
. . .
```

Figura 5.9: Tabela Dispatch.

É disparado um "listener", que aguarda a ocorrência de eventos na associação. As funções utilizadas permitem registrar um servidor que fica sempre escutando a associação e enfileirando os eventos que ocorrerem. Isto foi obtido com as funções

iserver-init e **iserver-wait**. O uso desta rotina possibilita a verificação periodica da ocorrência de eventos e a chamada das rotinas registradas relacionadas com cada evento em particular. Quando a função faz a chamada a uma das rotinas de tratamento de evento, ela passa alguns parâmetros, entre os quais o identificador da associação e um " buffer " contendo os dados recebidos para serem analisados, tratados e enviados para a aplicação através de uma outra área de memória compartilhada. Dependendo do retorno recebido da aplicação a rotina retorna um resultado relacionado com a chamada ou um dos erros contidos na tabela de erros da operação. Estas tabelas estão no arquivo CMIP-ops.c.

No caso do retorno de resultado é utilizada a função **RyDsResult**, que recebe os seguintes parâmetros:

- Identificador da associação.
- Um apontador para a tabela **RyOperation** contida em CMIP-ops.c responsável pela codificação e envio de primitivas de retorno da operacao específica.
- Um apontador para a estrutura RoSAPinvoke contendo outras informações relacionadas com a operação (identificador de invocação e outros).
- Um apontador para uma estrutura C que serve como argumento de retorno.
- Um apontador para estrutura roi, que é atualizada no caso falha na chamada da função.

Em caso de retorno de erro é utilizada a função **RyDsError** com os seguintes parâmetros:

- Identificador da associação.
- Identificador da chamada que esta sendo respondida.
- O código do erro encontrado.
- Um apontador para ums estrutura C relacionada com parâmetros de erro.

- Prioridade.
- Um apontador para a estrutura RoSAPinvoke.

5.3.3 Interface com a aplicação

A interface com a aplicação que usa os serviços do protocolo, é feita através das áreas de memória compartilhadas, que já foram descritas acima.

5.4 Módulos gerados

Como foi dito anteriormente, os compiladores ROSY e PEPSY se encarregam de facilitar o trabalho do programador, gerando alguns arquivos com rotinas que são incluídas no programa principal. A entrada do ROSY é um arquivo contendo definições na notação ROS, que é o caso da especificação IS-9596 para o CMIP. A partir deste arquivo de entrada, são gerados quatro outros arquivos:

1. CMIP-ops.c: Contém as estruturas de dados utilizadas pelo ambiente ROS. Contém também uma tabela table_CMIP_Operations com informações sobre todas as operações analisadas, tais como: o nome da operac cão, o código associado, informações relacionadas com as tabelas das rotinas de codificação e decodificação geradas pelo PEPSY. Também gera uma tabela contendo os possíveis erros referentes a cada operação. A codificação refere-se à conversão da representação dependente de máquina (em linguagem C) para uma representação independente de máquina(sintaxe de transferência), onde a estrutura das rotinas se baseia em tabelas.

2. CMIP-ops.h: Possui funções amigáveis que facilitam a chamada das primitivas do ROSE, ex. RO-INVOKE.request. Os nomes das rotinas são compostos pelo prefixo "stub" ou "op" (para chamadas assíncronas e sincronas), mais o nome do

módulo e a palavra OPERATION (ex.mEventReport-stub-OPERATION(....,....,,)). Em caso de chamadas assíncronas, deve-se fornecer os endereços das funções que serão chamadas quando a chamada retornar um resultado ou um erro. Em caso de chamadas síncronas, são passados os seguintes parâmetros para a rotina:

- Identificador de associação.
- O tipo de dado C instanciado relacionado com o argumento da operação, assim como os tipos C que foram gerados pelo compilador PEPSY que por sua vez estão relacionados com os tipos em ASN.1 da especificação inicial.
- Uma variável que sinaliza o retorno de um resultado ou de um erro.
- Um endereço da estrutura C relacionada com o retorno de um erro ou de um resultado.
- O endereço apontando para uma estrutura RoSAPindication que será usada para sinalizar eventos que podem ocorrem durante a chamada à função.

3. CMIP-stubs.c: este arquivo contém funções que utilizam o lint do UNIX.
4. CMIP.py: contém a sintaxe abstrata já sem a notação ROS e é utilizado como entrada para o compilador PEPSY. O compilador PEPSY gera por sua vez mais dois arquivos:
   - CMIP-types.h: contém os tipos C relacionados com as definições em ASN.1. Esta estrutura C é utilizada para transmissão e recepção de dados.
   - CMIP-tables.c: contém tabelas que tratam da conversão dos dados a serem transmitidos e recebidos para/do serviço de apresentação.

Além das rotinas geradas, são utilizadas algumas funções do ISODE implementadas no ambiente ROS. A função RyDsResult funciona de forma equivalente a uma primitiva RO-RESULT.request do ROSE. Esta função relaciona-se com um retorno de resultado. Enquanto a função RyDsError equivale à primitiva RO-ERROR.request, para retorno

de erro. São utilizadas ainda a função RyDsUReject que equivale à primitiva RO-REJECT-U.request, e a RyWait, utilizada para aguardar a ocorrência de eventos, e a RyDispatch que é usada para registrar as funções que executam as operações, dentre outras.

5.5 Organização do Código

O codigo final do protocolo é uma junção do programa principal contendo as rotinas criadas pelo programador, que utilizam as estruturas de dados das PDUs com rotinas de codificação e decodificação geradas pelos compiladores, e as funções da biblioteca *librosap* do ISODE

A implementação do protocolo faz uso da tabela (Dispatch) referenciada anteriormente para agrupar as rotinas que tratam os eventos relacionados com a recepção de primitivas. Estas rotinas são registradas pela função RyDispatch. E são acessadas através desta tabela na hora da ocorrência de algum evento relacionado com uma das rotinas de operações registradas. O relacionamento das rotinas criadas pelo programador é mostrado na Figura 5.10.

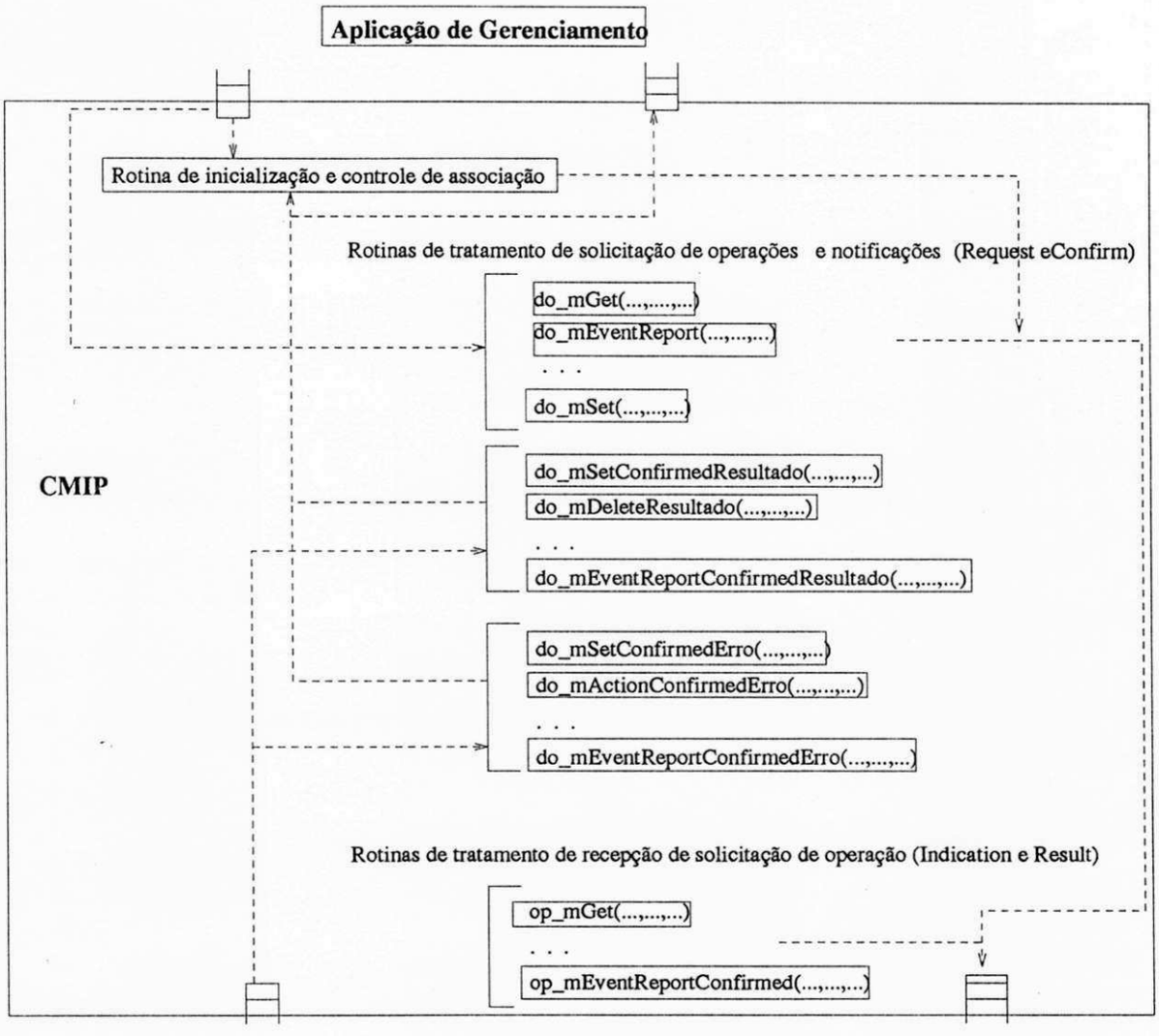

Figura 5.10: Funcionamento do CMIP.

Capítulo 6

Conclusão

A proposta básica do ISODE é possibilitar o desenvolvimento de aplicações distribuídas baseadas no Modelo de Referência OSI. Neste sentido, constatamos que o ISODE realmente facilita o desenvolvimento na medida em que:

- Modela a aplicação e os seus serviços (operações síncronas, assíncronas, remotas, ...);
- Fornece serviços básicos da arquitetura OSI (apresentação, sessão);
- Codifica e decodifica PDUs;
- Permite o uso parcial de métodos formais de codificação (ASN.1).

No caso particular desta implementação do CMIS/CMIP obteve-se também um ganho em termos de código com a utilização deste ambiente. No caso:

- Código gerado pelo programador foi minimizado (aproximadamente 3200 linhas de código C).
- O código gerado pelo ISODE foi de aproximadamente 4000 linhas para a parte da implementação executada.

Em relação aos demais trabalhos da área de gerência de redes, este trabalho se propõe a ser o protótipo básico de gerenciamento que deverá servir de base pāra a comunicação entre agentes, gerentes e MIBs a serem desenvolvidos em outras teses, relacionando-se também com o trabalho referenciado em [11], já desenvolvido.

Uma sugestão de trabalho futuro pode ser a verificação da eficiência do codigo gerado pelo ISODE. Do ponto de vista da sua estruturação, o código gerado faz chamadas recursivas em número considerável o que nos leva a um questionamento sobre a eficiência de sua execução. Um compromisso a ser verificado neste caso é a relação de ganho entre eficiência e simplificação da codificação dos programas. Neste contexto, uma outra sugestão de trabalho futuro, consiste em avaliar funcionalmente o CMIP em relação à implantações do protocolo SNMP largamente disponível no mercado, e que também é parte do pacote do ISODE.

Referências

[1] ISO. *ISO 9595 - Common Management Information Service Definition*, first edition, 1990.

[2] ISO/IEC. *ISO 9596 - Common Management Information Protocol Specification*, first edition, 1990.

[3] M. T. Rose, J. P. Onions, and J. C. Robbins. *The ISO Development Environmet: User's Manual. Vol:4: The Application Cookbook. Versão 7.0.*, julho 1991.

[4] M. T. Rose, J. P. Onions, and C. J. Robins. *The ISO Development Environmet: User's Manual Vol 1. Application Services. Versão 7.0.*, julho 1991.

[5] M. T. Rose, J. P. Onions, and J. C Robbins. *The ISO Development Environment: User's Manual, Vol2: Underlaying Services. Versão 7.0.*, julho 1991.

[6] Marshall T. Rose. *The Open Book - A Practical Perspective on OSI.* Prentice Hall, 1990.

[7] Douglas E. Commer. *Internetworking With TCP/IP*, volume 01. Prentice Hall, 1991.

[8] Douglas E. Commer. *Internetworking With TCP/IP*, volume 02. Prentice Hall, 1991.

[9] ISO. *ISO7498, Information Processig Systems - Open Systems Interconnection - Basic Reference Model.*, 1984.

[10] Clizenit Pinheiro Assis de Lima. Um modelo para implementagco de um servigo e protocolo de transferjncia de arquivos em rede. Master's thesis, DSC - UFPB -Campus II, Maio 1990.

[11] Francisco Araripe Costa. Aspectos de uma metodologia para modelagem de uma base de informagues de gerenciamento para o modelo osi. Master's thesis, DSC - UFPB - Campus II, Agosto 1993.

[12] ISO. *ISO9545, Information Processing Systems - Open Systems Interconnection - Application Layer Structure*, 1988.

[13] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall, 1988.

[14] Fred Halsall. *Data Communications, Computer Networks and OSI.* Addison-Wesley Publishing Company, second edition, 1989.

[15] Will Collins. *OSI Management Service Elements, Protocols and Application Layer Structure (ALS).* IFIP., 1989.

[16] ISO. *ISO 8825, Information Processing Systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).*, 1987.

[17] ISO. *ISO 8824, Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).*, 1987.

[18] Paulo R. L. Rebelles e José Pedro de Freitas. Modelagem de informação aplicada à gerência integrada de redes de telecomunicações. *Revista Telebrás.*, 17(59):134–145, dezembro 1993.

[19] Martin L. Schoffstall Jeffrey D. Case, Mark S. Fedor and James R. Davin. *A Simple Network Management Protocol. Request for Comments 1157.* DDN Network Information Center, SRI International, may 1990.

[20] Mark S. Fedor Martin L. Schoffstall, James R. Davin and Jeffrey D. Case. *SNMP over Ethernet. Request for Comments 11 61.* DDN Network information Center. SRI International., fevereiro 1989.

[21] L. Steinberg. *Draft Memorandum for Managing Asynchronously Generated Alerts.* RFC Draft , IBM, fevereiro 1989.

[22] J. P. Onions. Isode: In support of migration. *Computer Networks and ISDN Systems*, 17(425):362–366, 1989.

[23] J. B. Postel. *Transmission Control Protocol, Request for Comments 793.* DDN Network Information Center, SRI International, setembro 1981.

[24] M. T. Rose and D. E. Cass. *ISO Transport Services on Top of the TCP. Request for Comments.* Network Information Center, SRI International, maio 1987.

[25] B. Uyless. *Network Management Standards.* McGrawHill, 1992.

[26] ISO. *ISO 8649 :Information Processing Systems-Open System Interconnection - Service Definition for the Association Control Service Element.*, 1988.

Anexo 1

Este anexo contém a especificação do CMIS/CMIP em ASN.1.

```
-- Common Management Information Protocol (CMIP)
-- IS version
-- $Header: /var/home/cmot/pd/cmip-common/RCS/cmip-is.ry,v 1.6
            90/07/05 06:00:02 solomon Distrib $

-- CMIP-1
Cmip {joint-iso-ccitt ms(9) cmip(1) version1(1) protocol(3)}
DEFINITIONS ::= BEGIN

    -- Remote Operations definitions

    -- IMPORTS OPERATION, ERROR FROM Remote-Operation-Notation
    --  {joint-iso-ccitt remoteOperations(4) notation(0)}

    -- Directory Service definitions

    -- DistinguishedName, RDNSequence FROM InformationFramework
    --  {joint-iso-ccitt ds(5) modules(1) informationFramework(1)};
    --****** The following is adapted from Information-Framework ****--
    RDNSequence ::= SEQUENCE OF RelativeDistinguishedName
```

```
    --****** The following is adapted from Information-Framework ****--
    RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

    DistinguishedName ::= RDNSequence

    RelativeDistinguishedName ::= SET OF AttributeValueAssertion

AttributeValueAssertion ::= SEQUENCE {
attr AttributeType,
val  ANY }

AttributeType ::= OBJECT IDENTIFIER
    --***** end of definitions from Information-Framework ****--
    --****** The following is adapted from the IS CMIP, sec 7.3.1 ****--
    --CMIP-A-ASSOCIATE-Information { joint-iso-ccitt ms(9) cmip(1)
            version1(1)
    --  aAssociateUserInfo(1) }
    -- DEFINITIONS ::= BEGIN

    FunctionalUnits ::= BIT STRING {
multipleObjectSelection (0),
filter                  (1),
multipleReply           (2),
extendedService         (3) }
    -- Function i is supported if and only if bit i is one.

    -- Information caarried in user-information parameter of A-ASSOCIATE

    CMIPUserInfo ::= SEQUENCE {
protocolVersion [0] IMPLICIT ProtocolVersion DEFAULT { version1 },
functionalUnits [1] IMPLICIT FunctionalUnits DEFAULT {},
```

```
accessControl [2] EXTERNAL OPTIONAL,
userInfo [3] EXTERNAL OPTIONAL }

    ProtocolVersion ::= BIT STRING { version1 (0) }

    --  END
    --***** end of definitions from section 7.3.1 ****--
    --****** The following is adapted from the IS CMIP, sec 7.3.2 ****--
    --CMIP-A-ABORT-Information { joint-iso-ccitt ms(9) cmip(1) version1(1)
    --  aAbortUserInfo(2) }
    -- DEFINITIONS ::= BEGIN

    -- Information carried in user-information parameter of A-ABORT

--    CMIPAbortInfo ::= SEQUENCE {
-- abortSource [0] IMPLICIT CMIPAbortSource,
-- userInfo [1] EXTERNAL OPTIONAL }
--    CMIPAbortSource ::= ENUMERATED {
-- cmiseServiceUser (0),
-- cmiseServiceProvider (2) }

    --END
    --***** end of definitions from section 7.3.2 ****--

-- CMISE operations

-- In the following operations, the argument type is mandatory in the
-- corresponding ROSE APDU.

-- Action operations (M-ACTION)
```

```
m-Action OPERATION
 ARGUMENT ActionArgument
 ::= 6

m-Action-Confirmed OPERATION
 ARGUMENT ActionArgument
 RESULT   ActionResult
 -- This result is conditional; for conditions see
                [Recommendation X.CMIS.0 |
 -- ISO/IEC 9595] clause 8.3.3.2.9.
 ERRORS { accessDenied, classInstanceConflict, complexityLimitation,
  invalidScope, invalidArgumentValue, invalidFilter, noSuchAction,
  noSuchArgument, noSuchObjectClass, noSuchObjectInstance,
  processingFailure, syncNotSupported }
 LINKED { m-Linked-Reply }
 ::= 7

-- Create operation (M-CREATE)

m-Create OPERATION
 ARGUMENT CreateArgument
 RESULT   CreateResult -- optional
 ERRORS { accessDenied, classInstanceConflict, duplicateManagedObjectInstance
  invalidAttributeValue, invalidObjectInstance, missingAttributeValue,
  noSuchAttribute, noSuchObjectClass, noSuchObjectInstance,
  noSuchReferenceObject, processingFailure }
 ::= 8

-- Delete operation (M-DELETE)

m-Delete OPERATION
```

```
 ARGUMENT DeleteArgument
 RESULT   DeleteResult
 -- This result is conditional; for conditions see [Recommendation X.CMIS.0 |
 -- ISO/IEC 9595] clause 8.3.5.2.8.
 ERRORS { accessDenied, classInstanceConflict, complexityLimitation,
  invalidFilter, invalidScope, noSuchObjectClass,
  noSuchObjectInstance, processingFailure, syncNotSupported }
 LINKED { m-Linked-Reply }
 ::= 9

-- Event Reporting operations (M-EVENT-REPORT)

m-EventReport OPERATION
 ARGUMENT EventReportArgument
 ::= 0

m-EventReport-Confirmed OPERATION
 ARGUMENT EventReportArgument
 RESULT   EventReportResult -- optional
 ERRORS { invalidArgumentValue, noSuchArgument, noSuchEventType,
  noSuchObjectClass, noSuchObjectInstance, processingFailure }
::= 1

-- Get operation (M-GET)

m-Get OPERATION
 ARGUMENT GetArgument
 RESULT   GetResult
 -- This result is conditional; for conditions see
          [Recommendation X.CMIS.0 |
 -- ISO/IEC 9595] clause 8.3.1.2.8.
```

```
 ERRORS { accessDenied, classInstanceConflict, complexityLimitation,
  getListError, invalidFilter, invalidScope, noSuchObjectClass,
  noSuchObjectInstance, processingFailure, syncNotSupported }
 LINKED { m-Linked-Reply }
 ::= 3

-- Linked operation to M-GET, M-SET (Confirmed), M-ACTION (Confirmed), and
-- M-DELETE

m-Linked-Reply OPERATION
 ARGUMENT LinkedReplyArgument
 ::= 2

-- Set operations (M-SET)

m-Set OPERATION
 ARGUMENT SetArgument
 ::= 4

m-Set-Confirmed OPERATION
 ARGUMENT SetArgument
 RESULT   SetResult
 -- This result is conditional; for conditions see
                [Recommendation X.CMIS.0 |
 -- ISO/IEC 9595] clause 8.3.2.2.9.
 ERRORS { accessDenied, classInstanceConflict, complexityLimitation,
  invalidFilter, invalidScope, noSuchObjectClass,
                  noSuchObjectInstance,processingFailure, setListError,
 LINKED { m-Linked-Reply }
 ::= 5
```

```
-- CMIS error definitions

-- In the following errors, unless otherwise indicated, the parameter type is
-- mandatory in the corresponding ROSE APDU.

accessDenied ERROR
 ::= 2

classInstanceConflict ERROR
 PARAMETER BaseManagedObjectId
 ::= 19

complexityLimitation ERROR
 PARAMETER ComplexityLimitation -- optional
 ::= 20

duplicateManagedObjectInstance ERROR
 PARAMETER ObjectInstance
 ::= 11

getListError ERROR
 PARAMETER GetListError
 ::= 7

invalidArgumentValue ERROR
 PARAMETER InvalidArgumentValue
 ::= 15

invalidAttributeValue ERROR
 PARAMETER Attribute
 ::= 6
```

```
invalidFilter ERROR
 PARAMETER CMISFilter
 ::= 4

invalidScope ERROR
 PARAMETER Scope
 ::= 16

invalidObjectInstance ERROR
 PARAMETER ObjectInstance
 ::= 17

missingAttributeValue ERROR
 PARAMETER SET OF AttributeId
 ::= 18

noSuchAction ERROR
 PARAMETER NoSuchAction
 ::= 9

noSuchArgument ERROR
 PARAMETER NoSuchArgument
 ::= 14

noSuchAttribute ERROR
 PARAMETER AttributeId
 ::= 5

noSuchEventType ERROR
 PARAMETER NoSuchEventType
```

```
 ::= 13

noSuchObjectClass ERROR
 PARAMETER ObjectClass
 ::= 0

noSuchObjectInstance ERROR
 PARAMETER ObjectInstance
 ::= 1

noSuchReferenceObject ERROR
 PARAMETER ObjectInstance
 ::= 12

processingFailure ERROR
 PARAMETER ProcessingFailure -- optional
 ::= 10

setListError ERROR
 PARAMETER SetListError
 ::= 8

syncNotSupported ERROR
 PARAMETER CMISSync
 ::= 3

-- Supporting type definitions.

AccessControl ::= EXTERNAL

ActionArgument ::= SEQUENCE {
```

```
-- values for localForm. Where this alternative is used, the
-- permissible values for the integers and their meanings shall be
-- defined as part of the application context in which they are used.

AttributeIdError ::= SEQUENCE {
errorStatus ENUMERATED {
accessDenied (2),
noSuchAttribute (5) },
attributeId AttributeId }

BaseManagedObjectId ::= SEQUENCE {
baseManagedObjectClass ObjectClass,
baseManagedObjectInstance ObjectInstance }

CMISFilter ::= CHOICE {
item [8] FilterItem,
and [9] IMPLICIT SET OF %[ filter_list_and %] CMISFilter,
or [10] IMPLICIT SET OF %[ filter_list_or %] CMISFilter,
not [11] CMISFilter }

CMISSync ::= ENUMERATED {
bestEffort (0),
atomic (1) }

ComplexityLimitation ::= SET {
scope [0] Scope OPTIONAL,
filter [1] CMISFilter OPTIONAL,
sync [2] CMISSync OPTIONAL }

CreateArgument ::= SEQUENCE {
managedObjectClass ObjectClass,
```

```
managedObjectInstance CHOICE %[ create_arg_choice %] {
managedObjectInstance ObjectInstance,
superiorObjectInstance [8] ObjectInstance } OPTIONAL,
accessControl [5] AccessControl OPTIONAL,
referenceObjectInstance [6] ObjectInstance OPTIONAL,
attributeList [7] IMPLICIT AttributeList OPTIONAL }

-- Added by solomon@cs.wisc.edu for clearer posy output
AttributeList ::= SET OF Attribute

CreateResult ::= SEQUENCE {
managedObjectClass ObjectClass OPTIONAL,
managedObjectInstance ObjectInstance OPTIONAL,
-- shall be returned if omitted from CreateArgument
currentTime [5] IMPLICIT GeneralizedTime OPTIONAL,
attributeList [6] IMPLICIT AttributeList OPTIONAL }

DeleteArgument ::= SEQUENCE {
COMPONENTS OF BaseManagedObjectId,
accessControl [5] AccessControl OPTIONAL,
synchronization [6] IMPLICIT CMISSync DEFAULT bestEffort,
scope [7] Scope DEFAULT baseObject,
filter CMISFilter DEFAULT and {} }

DeleteError ::= SEQUENCE {
managedObjectClass ObjectClass OPTIONAL,
managedObjectInstance ObjectInstance OPTIONAL,
currentTime [5] IMPLICIT GeneralizedTime OPTIONAL,
deleteErrorInfo [6] ENUMERATED { accessDenied (2) } }

DeleteResult ::= SEQUENCE {
```

```
managedObjectClass ObjectClass OPTIONAL,
managedObjectInstance ObjectInstance OPTIONAL,
currentTime [5] IMPLICIT GeneralizedTime OPTIONAL }

EventReply ::= SEQUENCE {
eventType EventTypeId,
eventReplyInfo [8] ANY DEFINED BY eventType OPTIONAL }

EventReportArgument ::= SEQUENCE {
managedObjectClass ObjectClass,
managedObjectInstance ObjectInstance,
eventTime [5] IMPLICIT GeneralizedTime OPTIONAL,
eventType EventTypeId,
eventInfo [8] ANY DEFINED BY eventType OPTIONAL }

EventReportResult ::= SEQUENCE {
managedObjectClass ObjectClass OPTIONAL,
managedObjectInstance ObjectInstance OPTIONAL,
currentTime [5] IMPLICIT GeneralizedTime OPTIONAL,
eventReply EventReply OPTIONAL }

EventTypeId ::= CHOICE {
globalForm [6] IMPLICIT OBJECT IDENTIFIER,
localForm [7] IMPLICIT INTEGER }
-- [Recommendation X.CMIP.0 | ISO/IEC 9596] does not allocate any
-- values for localIForm.  Where this alternative is used, the
-- permissible values for the integers and their meanings shall be
-- defined as part of the application context in which they are used.

FilterItem ::= CHOICE {
equality [0] IMPLICIT Attribute,
```

```
substrings [1] IMPLICIT SEQUENCE OF %[ substrings_assertion %]
    CHOICE %[ substring_choice $ substring %] {
initialString [0] IMPLICIT SEQUENCE %[ init_choice %] {
attributeId AttributeId,
string ANY DEFINED BY attributeId},
anyString [1] IMPLICIT SEQUENCE %[ any_choice %] {
attributeId AttributeId,
string ANY DEFINED BY attributeId},
finalString [2] IMPLICIT SEQUENCE %[ final_choice %] {
attributeId AttributeId,
string ANY DEFINED BY attributeId} },
greaterOrEqual [2] IMPLICIT Attribute,
-- asserted value >= attribute value
lessOrEqual [3] IMPLICIT Attribute,
-- asserted value <= attribute value
present [4] AttributeId,
subsetOf [5] IMPLICIT Attribute,
-- asserted value is a subset of attribute value
supersetOf [6] IMPLICIT Attribute,
-- asserted value is a superset of attribute value
nonNullSetIntersection [7] IMPLICIT Attribute }

GetArgument ::= SEQUENCE {
COMPONENTS OF BaseManagedObjectId,
accessControl [5] AccessControl OPTIONAL,
synchronization [6] IMPLICIT CMISSync DEFAULT bestEffort,
scope [7] Scope DEFAULT baseObject,
filter CMISFilter DEFAULT and {},
attributeIdList [12] IMPLICIT SET OF %[ attribute_id_list %]
AttributeId OPTIONAL }
```

```
GetInfoStatus ::= CHOICE {
attributeIdError [0] IMPLICIT AttributeIdError,
attribute [1] IMPLICIT Attribute }

GetListError ::= SEQUENCE {
managedObjectClass ObjectClass OPTIONAL,
managedObjectInstance ObjectInstance OPTIONAL,
currentTime [5] IMPLICIT GeneralizedTime OPTIONAL,
getInfoList [6] IMPLICIT SET OF %[ gi_status_list %]
GetInfoStatus }

GetResult ::= SEQUENCE {
managedObjectClass ObjectClass OPTIONAL,
managedObjectInstance ObjectInstance OPTIONAL,
currentTime [5] IMPLICIT GeneralizedTime OPTIONAL,
attributeList [6] IMPLICIT AttributeList }

InvalidArgumentValue ::= CHOICE {
actionValue [0] IMPLICIT ActionInfo,
eventValue [1] IMPLICIT SEQUENCE %[ event_error %] {
eventType EventTypeId,
eventInfo [8] ANY DEFINED BY eventType OPTIONAL } }

LinkedReplyArgument ::= CHOICE {
getResult [0] IMPLICIT GetResult,
getListError [1] IMPLICIT GetListError,
setResult [2] IMPLICIT SetResult,
setListError [3] IMPLICIT SetListError,
actionResult [4] IMPLICIT ActionResult,
processingFailure [5] IMPLICIT ProcessingFailure,
deleteResult [6] IMPLICIT DeleteResult,
```

```
actionError [7] IMPLICIT ActionError,
deleteError [8] IMPLICIT DeleteError }

NoSuchAction ::= SEQUENCE {
managedObjectClass ObjectClass,
actionType ActionTypeId }

NoSuchArgument ::= CHOICE {
actionId [0] IMPLICIT SEQUENCE %[ bad_actionId %] {
managedObjectClass ObjectClass OPTIONAL,
actionType ActionTypeId },
eventId [1] IMPLICIT SEQUENCE %[ bad_eventId %] {
managedObjectClass ObjectClass OPTIONAL,
eventType EventTypeId } }

NoSuchEventType ::= SEQUENCE {
managedObjectClass ObjectClass,
eventType EventTypeId }

ObjectClass ::= CHOICE {
globalForm [0] IMPLICIT OBJECT IDENTIFIER,
localForm [1] IMPLICIT INTEGER }
-- [Recommendation X.CMIP.0 | ISO/IEC 9596] does not allocate any
-- values for localForm.  Where this alternative is used, the
-- permissible values for the integers and their meanings shall be
-- defined as part of the application context in which they are used.

ObjectInstance ::= CHOICE {
distinguishedName [2] IMPLICIT DistinguishedName,
nonSpecificForm [3] IMPLICIT OCTET STRING,
localDistinguishedName [4] IMPLICIT RDNSequence }
```

```
-- localDistinguishedName is that portion of the distinguished name
-- that is necessary to unambiguously identify the managed object
-- within the context of communication between the open systems.

ProcessingFailure ::= SEQUENCE {
managedObjectClass ObjectClass,
managedObjectInstance ObjectInstance OPTIONAL,
specificErrorInfo [5] ANY DEFINED BY managedObjectClass }

Scope ::= CHOICE {
simpleScope INTEGER {
baseObject (0),
firstLevelOnly (1),
wholeSubtree (2) },
individualLevels [1] IMPLICIT INTEGER,
-- POSITIVE integer indicates the level to be selected.
baseToNthLevel [2] IMPLICIT INTEGER }
-- POSITIVE integer N indicates that the range of levels (0 - N) is to
-- be selected.
    -- With individualLevels and baseToNthLevel, a value of 0 has the same
    -- semantics as baseObject.  With baseToNthLevel, a value of 1 has the
    -- same semantics as firstLevelOnly.

SetArgument ::= SEQUENCE {
COMPONENTS OF BaseManagedObjectId,
accessControl [5] AccessControl OPTIONAL,
synchronization [6] IMPLICIT CMISSync DEFAULT bestEffort,
scope [7] Scope DEFAULT baseObject,
filter CMISFilter DEFAULT and {},
attributeList [12] IMPLICIT AttributeList }
```

```
SetInfoStatus ::= CHOICE {
attributeError [0] IMPLICIT AttributeError,
attribute [1] IMPLICIT Attribute }

SetListError ::= SEQUENCE {
managedObjectClass ObjectClass OPTIONAL,
managedObjectInstance ObjectInstance OPTIONAL,
currentTime [5] IMPLICIT GeneralizedTime OPTIONAL,
setInfoList [6] IMPLICIT SET OF %[ si_status_list %]
SetInfoStatus }

SetResult ::= SEQUENCE {
managedObjectClass ObjectClass OPTIONAL,
managedObjectInstance ObjectInstance OPTIONAL,
currentTime [5] IMPLICIT GeneralizedTime OPTIONAL,
attributeList [6] IMPLICIT AttributeList OPTIONAL }

END -- End of CMIP syntax defintions
```

UPPA / BIBLIOTECA / PRAI

Anexo 2

Códigos de erro do CMIP: Cada operação possui um conjunto de possíveis erros. Cada erro é mapeado em uma tabela que contém um número inteiro relacionado. Os erros do CMIS/CMIP são:

- *Access Denied* : A operação não foi executada devido a não ter permissão de acesso.
- *Class Instance Conflict*: O MOI não é membro da classe especificada.
- *Complexity Limitation*: O parâmetro contido na solicitação da operação é muito complexo.
- *Duplicate Invocation*: O identificador de invocação já foi especificado antes.
- *Duplicate Managed Object Instance*: Instância de objeto especificada já sendo utilizada em outra classe.
- *Get List Error*: Um ou mais atributos não foram lidos devido a problemas de acesso.
- *Invalid Argument Value*: O argumento especificado não é um argumento aceitável, está fora de campo ou não é reconhecido.
- *Invalid Attribute Value*: O valor do atributo não é permitido, não é reconhecido ou está fora de campo.

- *Invalid Attribute Value*: O valor do atributo não é permitido, não é reconhecido ou está fora de campo.
- *Invalid Filter*: O parâmetro de filtro não foi especificado pelas unidades funcionais estabelecidas ou o valor é desconhecido.
- *Invalid Object Instance*: O nome da instância do objeto especificado viola as regras para nomes de objetos.
- *Missing Attribute Value*: O valor do atributo solicitado não foi especificado ou não está disponível.
- *Invalid Scope*: Erro no parâmetro de escopo.
- *Mistyped Argument*: Argumentos não permitido para a associação.
- *No Such Action*: Tipo de ação não permitido.
- *No Such Argument*: Argumento não reconhecido.
- *No Such Attribute*: Atributo não reconhecido.
- *No Such Event Type*: Evento não cadastrado nos possíveis tipos.
- *No Such Invoke Identifier*: O receptor não reconhece o identificador de invocação.
- *No Such Object Class*: O objeto gerenciado especificado não foi reconhecido.
- *No Such Object Instance*: O MOI especificado não foi reconhecido.
- *No Such Reference Object*: O parâmetro de referência da instância do objeto não foi reconhecido.
- *Processing Failure*: Qualquer tipo de erro relacionado com o processamento.
- *Resource Limitation*: A operação não foi executada devido a limitação nas capacidades dos recursos.

- *Set List Error*: Valores de atributos não puderam ser modificados, as razões são especificadas.
- *Synchronization Not Supported*: Sincronização não suportada.
- *Unrecognized Operation*: Operação não permitida ou não reconhecida.

Anexo 3

Este anexo contém trechos da implementação do CMIS/CMIP no ambiente ISODE.

```
#include <stdio.h>
#include <alloca.h>
#include <setjmp.h>
#include <varargs.h>
#include "shr_mem.h"
#include "ryres.h"
#include "CM-types.h"
#include "CM-ops.h"
#include "psap.h"
#include "ryinitiator.h"
#include "../include/isode/rosy.h"
#include "tsap.h"                          /* Para escutar a rede */
#include "tailor.h"


#define TRUE 1
```

```
#define TRUE 1
#define xalloc(p,type) ((p)=(type) calloc(1,sizeof*(p)))


int ros_init(), ros_work(), ros_indication(), ros_lose();
static IFP startfnx;
static IFP stopfnx;
static jmp_buf toplevel;
static LLog _pgm_log = {
    "responder.log", NULLCP, NULLCP,
    LLOG_FATAL | LLOG_EXCEPTIONS | LLOG_NOTICE, LLOG_FATAL, -1,
    LLOGCLS | LLOGCRT | LLOGZER, NOTOK
};
LLog *pgm_log = &_pgm_log;
IFP start,
stop;

/*      Para as operacoes confirmadas o quinto campo da table_CM_operations
        contida no CM-ops.c eh um inteiro diferente de zero
*/


int do_mEventReportConfirmed();
int op_mEventReportConfirmed();
int do_mEventReportConfirmedResultado();
int do_mEventReportConfirmedErro();
int do_mEventReport();
int op_mEventReport();
int do_mGet();
```

```
int op_mGet();
int do_mGetResultado();
int do_mGetErro();
int do_mDelete();
int do_mDeleteResultado();
int do_mDeleteErro();
int op_mDelete();
int do_mCreate();
int do_mCreateResult();
int do_mCreateErro();
int op_mCreate();
int do_mSet();
int op_mSet();
int do_mSetConfirmed();
int do_mSetConfirmedResultado();
int do_mSetConfirmedErro();
int op_mSetConfirmed();
int do_mActionConfirmed();
int do_mActionConfirmedResultado();
int do_mActionConfirmedErro();
int op_mActionConfirmed();
int do_mAction();
int op_mAction();


static struct dispatch disp[] = {
"mSet",operation_CM_mSet,op_mSet,

"mSetConfirmed",operation_CM_mSetConfirmed,op_mSetConfirmed,

"mAction",operation_CM_mAction,op_mAction,
```

```
"mActionConfirmed",operation_CM_mActionConfirmed,op_mActionConfirmed,

"mEventReportConfirmed",operation_CM_mEventReportConfirmed,
       op_mEventReportConfirmed,

"mEventReport",operation_CM_mEventReport,op_mEventReport,

"mCreate",operation_CM_mCreate,op_mCreate,

"mDelete",operation_CM_mDelete,op_mDelete,

"mGet",operation_CM_mGet,op_mGet,



           NULL
};


char *mycontext = "iso cmip";
char *mypci = "cmip pci";
char *myservice = "cmip";
static char *myname = "cmip";

main(argc,argv,envp)
int argc;
char **argv,
     **envp;
{
```

```
struct TSAPdisconnect *td;
fd_set rfds, wfds, nfds;
int resultado,resposta;
struct type_CM_EventReportArgument *arg;
struct RoSAPindication rois;
register struct RoSAPindication *roi=&rois;
register struct RoSAPpreject *rop=&roi->roi_preject;
int sd,id;
caddr_t *out;
struct SSAPref sfs;
register struct SSAPref *sf;
register struct PSAPaddr *pa;
struct AcSAPconnect accs;
register struct AcSAPconnect *acc=&accs;
AEI aei;
OID ctx,
    pci;
struct PSAPctxlist pcs;
register struct PSAPctxlist *pc=&pcs;
struct AcSAPrelease acrs;
register struct AcSAPrelease *acr=&acrs;
struct AcSAPindication acis;
register struct AcSAPindication *aci=&acis;
register struct AcSAPabort *aca=&aci->aci_abort;
char saida;
register struct dispatch *ds;
char *host;
struct RyOperation *ops;
char buffer[100];
```

```
if((aei=_str2aei(argv[1],myservice,mycontext,isatty(fileno(stdin)),
        NULLCP,NULLCP))==NULLAEI)
    printf("\nImpossivel fornecer servico\n");
if((pa=aei2addr(aei))==NULLPA)
    printf("\nfalha na traducao de endereco\n");
if((ctx = ode2oid (mycontext)) == NULLOID)
    printf("\ndescritor de objeto desconhecido: %s\n",mycontext);
if ((ctx=oid_cpy(ctx))==NULLOID)
    printf("\nFora de memoria\n");
if((pci=ode2oid(mypci))==NULLOID)
    printf("\nDescritor de object desconhecido: %s",mypci);
if((pci=oid_cpy(pci))==NULLOID)
    printf("\nFora de memoria\n");
pc->pc_nctx=1;
pc->pc_ctx[0].pc_id=1;
pc->pc_ctx[0].pc_asn=pci;
pc->pc_ctx[0].pc_atn=NULLOID;

if((sf=addr2ref(PLocalHostName()))==NULL){
 sf=&sfs;
 (void)bzero((char *)sf,sizeof *sf);
}
if (AcAssocRequest(ctx,NULLAEI,aei,NULLPA,pa,pc,NULLOID,0,ROS_MYREQUIRE,
          SERIAL_NONE,0,sf,NULLPEP,0,NULLQOS,acc,aci)==NOTOK)
printf("\nErro no A-ASSOCIATE.REQUEST\n");
if(acc->acc_result != ACS_ACCEPT)
printf("\nAssociacao Rejeitada: [%s]\n",AcErrString(acc->acc_result));
sd=acc->acc_sd;
ACCFREE(acc);
if(RoSetService(sd,RoPService,roi)==NOTOK)
 printf("\nFalha no acesso de APRESENTACAO\n");
```

```
/*
   if (myname = rindex (argv[0], '/'))
myname++;
    if (myname == NULL || *myname == NULL)
myname = argv[0];

    isodetailor (myname, 0);
    if (debug = isatty (fileno (stderr)))
ll_dbinit (pgm_log, myname);
    else {
static char  myfile[BUFSIZ];

(void) sprintf (myfile, "%s.log",
(strncmp (myname, "ros.", 4)
    && strncmp (myname, "lpp.", 4))
|| myname[4] == NULL
    ? myname : myname + 4);
pgm_log -> ll_file = myfile;
ll_hdinit (pgm_log, myname);
    }

    advise (LLOG_NOTICE, NULLCP, "starting");

    if ((aei = _str2aei (host, myservice, mycontext, 0, NULLCP, NULLCP))
    == NULLAEI)
adios (NULLCP, "unable to resolve service: %s", PY_pepy);

*/
   for (ds = disp; ds -> ds_name; ds++)
```

```
if (RyDispatch (NOTOK, ops, ds -> ds_operation, ds->ds_vector, roi)
== NOTOK)
    ros_adios (rop, ds -> ds_name);

    startfnx = start;
    stopfnx = stop;
ds=disp;
printf("\n------>%s\n",ds->ds_operation);



if (iserver_init(argc,argv,aei,ros_init,td)==NOTOK)
  printf("\nErro na recepcao\n");

      /* registra um "listener" e caso o "servidor" esteja rodando no */
      /* modo dinamico pode chamar ros_init para tratar outras solici-*/
      /* tacoes de associacao                                          */

for(;;)
  {
      /*verifica a ocorrencia de eventos na associacao                */

/*  switch(iserver_wait(ros_init,ros_work,ros_lose,sd,
        recebe(1,buffer,sizeof(buffer)),envia(1,buffer,sizeof(buffer)),
              20,td))
*/
  switch(iserver_wait(ros_init,ros_work,ros_lose,sd,&rfds,&wfds,NULLFP,
                      20,td))
       {
         case DONE:
              printf("\n ");
```

```
                exit(0);
                break;
          case NOTOK:
                printf("\n ");
                break;
          case OK:
                printf("\n Executa outras atividades neste ponto");
                break;
          default:
                printf("\n***\n");
                break;
         }

     /* verifica se existe dado na pilha do aplicativo          */
     /* caso afirmativo le e chama a funcao que implementa      */
     /* a operacao solicitada                                   */

if(recebe(1,buffer,sizeof(buffer))==NOTOK)
 printf("\n.........");
 else {
   if(str2vec(buffer,vec)<1)
       continue;
   switch (vec[0]) {
       case '6':
              do_mAction(sd);
       case '7':
              do_mActionConfirmed(sd);
       case '8':
              do_mCreate(sd);
       case '9':
              do_mDelete(sd);
```

```
        case '0':
                do_mEventReport(sd);
        case '1':
                do_mEventReportConfirmed(sd);
        case '3':
                do_mGet(sd);
        case '4':
                do_mSet(sd);
        case '5':
                do_mSetConfirmed(sd);
                   }
/* A escolha dentro do switch se relaciona com a   */
/* numeracao das operacoes no codigo ASN.1         */

printf("\nPassou 30 ************.............*****************\n");
}

scanf("%c",&saida);
 if(AcRelRequest(sd,ACF_NORMAL,NULLPEP,0,NOTOK,acr,aci)==NOTOK)
    printf("\nProblemas na liberacao da Associacao\n");


}

/*************************************************************************/

static int do_mGet(sd)
int sd;
                  /* trata Get.REQUEST e Get.CONFIRMATION */

{
```

```
                    /* Os dados sao lidos de uma area de memoria comparti-  */
                    /* lhada controladas pelas rotinas envia e recebe, para */
                    /* efeito de teste os dados sao atribuidos diretamente  */

if ( (arg =
 (struct type_CM_GetArgument*) calloc(1, sizeof *arg)) == NULL)
            printf("erro na alocacao");
if (( (arg) = (struct type_CM_GetArgument *)
      calloc(1,sizeof(struct type_CM_GetArgument))) == NULL)
            printf("erro na alocacao 1");
if (( (arg)->baseManagedObjectClass3 = (struct baseManagedObjectClass3 *)
      calloc(1,sizeof(struct baseManagedObjectClass3))) == NULL)
            printf("erro na alocacao 2");

(arg)->baseManagedObjectClass3->offset  = baseManagedObjectClass3_globalForm
(arg)->baseManagedObjectClass3->un.globalForm = str2oid("1.0.10162.1.1");


if (((arg)->accessControl3 = (struct type_UNIV_EXTERNAL *)
     calloc (1, sizeof (struct type_UNIV_EXTERNAL))) == NULL)
          {
            printf("erro na alocacao 2");
            exit(1);
          }
printf("xp11\n");
(arg)->accessControl3->direct__reference = str2oid("1.0.10162.1.1");
printf("xp12_X\n");
(arg)->accessControl3->indirect__reference = 10162;
printf("xp13\n");
(arg)->accessControl3->data__value__descriptor=str2qb("userfiled",
      strlen("userfiled"),1);
```

```
printf("xp14\n");

if (((arg)->accessControl3->encoding= (struct choice_UNIV_0 *)
       calloc (1, sizeof (struct choice_UNIV_0))) == NULL)
          {
            printf("erro na alocacao 2");
            exit(1);
          }
(arg)->accessControl3->encoding->offset = 2;
printf("xp15\n");
(arg)->accessControl3->encoding->un.octet__aligned =     str2qb("aquivemnarea
             ("aquivemnarealainfodousuario"),1);
printf("xp16\n");



(*arg)->synchronization3 = 1;

if (( (*arg)->scope3 = (struct scope3 *)
     calloc(1,sizeof(struct scope3))) == NULL)
          {
            printf("erro na alocacao 3");
            exit(1);
          }


(*arg)->scope3->offset = 1;
(*arg)->scope3->un.simpleScope = 2;

if (( (*arg)->filter3 = (struct type_CM_CMISFilter *)
      calloc(1,sizeof(struct type_CM_CMISFilter))) == NULL)
```

```
        {
          printf("erro na alocacao 3");
          exit(1);
        }

if (( (*arg)->filter3->un.item = (struct type_CM_FilterItem *)
      calloc(1,sizeof(struct  type_CM_FilterItem))) == NULL)
        {
          printf("erro na alocacao 3");
          exit(1);
        }


if (( (*arg)->filter3->un.item->un.equality= (struct equality *)
      calloc(1,sizeof(struct  equality))) == NULL)
        {
          printf("erro na alocacao 3");
          exit(1);
        }

if (( (*arg)->filter3->un.item->un.equality->attributeId= (struct type_CM_Att
      calloc(1,sizeof(struct  type_CM_AttributeId))) == NULL)
        {
          printf("erro na alocacao 3");
          exit(1);
        }

(*arg)->filter3->offset= 1;
(*arg)->filter3->un.item->offset= 1;
(*arg)->filter3->un.item->un.equality->attributeId->offset = 2;
(*arg)->filter3->un.item->un.equality->attributeId->un.localForm = 4;
```

```
if ((pe = pe_alloc(PE_CLASS_UNIV,PE_FORM_PRIM,PE_PRIM_BITS)) == NULLPE)
    printf("pE_alloc");

bit_off(pe,0);
bit_on(pe,1);
bit_off(pe,2);
bit_on(pe,3);

(*arg)->filter3->un.item->un.equality->attributeValue= 1;

printf("xp2.2\n");


switch(resultado=stub_CM_mGet(sd,id=RyGenID(sd),arg,do_GetResultado,
         do_GetErro,ROS_SYNC,roi)){
    case NOTOK:
          printf("\nProblemas na operacao EventReportConfirmed\n");
          break;
    case OK:
          break;
    case DONE:
          break;
    default:
          printf("\nRetorno do RyStub desconhecido\n");
          break;
    }
printf("\nid=%d",id);
free_CM_GetArgument(arg);
return OK;
}
```

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

static int op_mGet(sd,ryo,rox,in,roi)
int sd;
struct RyOperation *ryo;
struct RoSAPinvoke *rox;
caddr_t in;
struct RoSAPindication *roi;

{   PE pe;
    PS ps;
    PE rpe;
   int sei = TRUE;
   struct type_CM_GetResult *res = NULL;
   struct type_CM_GetArgument *arg = (struct type_CM_GetArgument*) in;

 printf("executando a operacao mGet ..........\n");
 printf("\n\nManagedObjectClass**********");
 printf("\n  Offset da Classe : %d\n",(arg)->baseManagedObjectClass2->
        offset);

if ((arg)->baseManagedObjectClass3->offset == 1)
   printf("\n  Forma Global : %s\n",sprintoid((arg)->baseManagedObjectClass4-
 if ((arg)->baseManagedObjectClass3->offset == 2)
   printf("\n  Forma Local  : %s",(arg)->baseManagedObjectClass2->
         un.localForm);

if ((arg)->baseManagedObjectClass3->offset ==1)
 {
   if (RyDsError(sd,rox->rox_id,error_CM_accessDenied,(caddr_t) NULL,ROS_NOPR
```

```
    printf("\nerror no RyDsError\n");
   return OK;
         }



res= (struct type_CM_GetResult*) calloc (1,sizeof *res);
res->managedObjectClass4 = (struct managedObjectClass4 *) calloc(1,sizeof(st
res->managedObjectClass4->offset = 1;
res->managedObjectClass4->un.globalForm = str2oid("1.0.10162.1.1");



if (RyDsResult(sd,rox->rox_id,(caddr_t) res,ROS_NOPRIO,roi) == NOTOK)
    {
      ros_adios(&roi->roi_preject,"RESULT");
     printf("erro no RyDsResult\n");
printf("%d,%d,%d,%d,%d\n",sd,rox->rox_id,(caddr_t) res,ROS_NOPRIO,roi);
     printf("erro: %d,%s,%s\n",roi->roi_type,roi->roi_ureject.rou_reason,roi-
       roi_un_ureject.rou_reason);
  printf("erro: %d,%s,%s\n",roi->roi_type,roi->roi_preject.rop_reason,roi->ro
       roi_un_preject.rop_reason);
     }



   return OK;
}
```

```
{

                        /* e feita uma pesquisa na tabela de erro CMIP   */
                        /* para entao ser passado para o aplicativo      */

  printf("Retorno de erro na operacao m-Get\n");
  printf("num associacao: %d num da invocacao  %d\n",sd, id);
  printf("erro: %d (tipo) %d (razao)\n",roi->roi_type,roi->roi_ureject.rou_re
  printf("\nRetorno de erro -> ");
  switch (error){
      case 15 :
           printf("Acesso nao permitido\n");
           break;
      case 0  :
           printf("Conflito de instancia de classe\n");
           break;
      case 1  :
           printf("Limitacao de complexidade\n");
           break;
      case 13 :
           printf("Erro na lista op Get\n");
           break;
      case 14 :
           printf("Filtro invalido\n");
           break;
      case 1  :
           printf("Scopo invalido\n");
           break;
      case 13 :
           printf("Classe de objeto inexistente\n");
           break;
```

```
      case 14 :
           printf("Instancia de objeto inexistente\n");
           break;
      case 10 :
           printf("Falha de processamento\n");
           break;
      case 3 :
           printf("Sincronizacao nao suportada\n");
           break;

      default :
           printf("Erro nao consta na tabela de eventos da operacao\n");
           break;
       }

printf("ERRO---> %d",error);

   return OK;
}



/*****************************************************************************/


...
...
```

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


static int  ros_init (vecp, vec)
int vecp;
char  **vec;
{
   int     reply,
    result,
    sd;
    struct AcSAPstart   acss;
    register struct AcSAPstart *acs = &acss;
    struct AcSAPindication  acis;
    register struct AcSAPindication *aci = &acis;
    register struct AcSAPabort   *aca = &aci -> aci_abort;
    register struct PSAPstart *ps = &acs -> acs_start;
    struct RoSAPindication  rois;
    register struct RoSAPindication *roi = &rois;
    register struct RoSAPpreject   *rop = &roi -> roi_preject;

    if (AcInit (vecp, vec, acs, aci) == NOTOK) {
acs_advise (aca, "initialization fails");
return NOTOK;
    }
    advise (LLOG_NOTICE, NULLCP,
"A-ASSOCIATE.INDICATION: <%d, %s, %s, %s, %d>",
acs -> acs_sd, oid2ode (acs -> acs_context),
sprintaei (&acs -> acs_callingtitle),
sprintaei (&acs -> acs_calledtitle), acs -> acs_ninfo);
```

```
    sd = acs -> acs_sd;

    for (vec++; *vec; vec++)
advise (LLOG_EXCEPTIONS, NULLCP, "unknown argument \"%s\"", *vec);

    reply = startfnx ? (*startfnx) (sd, acs) : ACS_ACCEPT;

    result = AcAssocResponse (sd, reply,
reply != ACS_ACCEPT ? ACS_USER_NOREASON : ACS_USER_NULL,
NULLOID, NULLAEI, NULLPA, NULLPC, ps -> ps_defctxresult,
ps -> ps_prequirements, ps -> ps_srequirements, SERIAL_NONE,
ps -> ps_settings, &ps -> ps_connect, NULLPEP, 0, aci);

    ACSFREE (acs);

    if (result == NOTOK) {
acs_advise (aca, "A-ASSOCIATE.RESPONSE");
return NOTOK;
    }
    if (reply != ACS_ACCEPT)
return NOTOK;

    if (RoSetService (sd, RoPService, roi) == NOTOK)
ros_adios (rop, "set RO/PS fails");

 return sd;
}

/*****************************************************************************/

static int  ros_work (fd)
```

```
int fd;
{
    int     result;
    caddr_t out;
    struct AcSAPindication  acis;
    struct RoSAPindication  rois;
    register struct RoSAPindication *roi = &rois;
    register struct RoSAPpreject   *rop = &roi -> roi_preject;

    switch (setjmp (toplevel)) {
case OK:
    break;

default:
    if (stopfnx)
(*stopfnx) (fd, (struct AcSAPfinish *) 0);
case DONE:
    (void) AcUAbortRequest (fd, NULLPEP, 0, &acis);
    (void) RyLose (fd, roi);
    return NOTOK;
    }

    switch (result = RyWait (fd, NULLIP, &out, OK, roi)) {
case NOTOK:
    if (rop -> rop_reason == ROS_TIMER)
break;
case OK:
case DONE:
    ros_indication (fd, roi);
    break;
```

```
else
    advise (LLOG_EXCEPTIONS, NULLCP,
    "RO-REJECT-U.INDICATION/%d: %s (id=%d)",
    sd, RoErrString (rou -> rou_reason),
    rou -> rou_id);
    }
    break;

case ROI_PREJECT:
    {
register struct RoSAPpreject   *rop = &roi -> roi_preject;

if (ROS_FATAL (rop -> rop_reason))
    ros_adios (rop, "RO-REJECT-P.INDICATION");
ros_advise (rop, "RO-REJECT-P.INDICATION");
    }
    break;

case ROI_FINISH:
    {
register struct AcSAPfinish *acf = &roi -> roi_finish;
struct AcSAPindication  acis;
register struct AcSAPabort *aca = &acis.aci_abort;

advise (LLOG_NOTICE, NULLCP, "A-RELEASE.INDICATION/%d: %d",
sd, acf -> acf_reason);

reply = stopfnx ? (*stopfnx) (sd, acf) : ACS_ACCEPT;

result = AcRelResponse (sd, reply, ACR_NORMAL, NULLPEP, 0,
    &acis);
```

```
ACFFREE (acf);

if (result == NOTOK)
    acs_advise (aca, "A-RELEASE.RESPONSE");
else
    if (reply != ACS_ACCEPT)
break;
longjmp (toplevel, DONE);
    }
/* NOTREACHED */

default:
    adios (NULLCP, "unknown indication type=%d", roi -> roi_type);
    }
}

/*****************************************************************************/

static int  ros_lose (td)
struct TSAPdisconnect *td;
{
    if (td -> td_cc > 0)
adios (NULLCP, "TNetAccept: [%s] %*.*s",
TErrString (td -> td_reason), td -> td_cc, td -> td_cc,
td -> td_data);
    else
adios (NULLCP, "TNetAccept: [%s]", TErrString (td -> td_reason));
}
```