UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Hans Manfred Schönberger

Sistema de Informação para Suporte à Gerência de Falhas

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Profª. Drª. Elizabeth Sueli Specialski

Florianópolis, Dezembro de 2000

Sistema de Informação para Suporte à Gerência de Falhas

Hans Manfred Schönberger

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração de Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Fernando Álvaro Ostuni Gauthier, Dr.
Coordenador do Curso

Banca Examinadora

Profa. Elizabeth Sueli Specialski, Dra.
Orientadora

Prof. João Bosco da Mota Alves, Dr.

Alexandre Moraes Ramos, Dr.

Alessandra Schweitzer, M.Sc.

Dedico este trabalho a meus pais,
Diedlin Janke e Manfred Georg

AGRADECIMENTOS

Agradeço a todos os companheiros e amigos que de alguma forma contribuíram para realização deste trabalho.

De maneira especial a professora Elizabeth Sueli Specialski pelo conhecimento, apoio, paciência, compreensão e estímulo dedicados em cada etapa.

À minha irmã Selena Schönberger pelo estímulo e apoio dado no decorrer deste trabalho.

Aos componentes da banca, João Bosco da Mota Alves, Alessandra Schweitzer e Alexandre Moraes Ramos pelo tempo dedicado na avaliação e as sugestões apresentadas para a melhoria desse trabalho.

Sumário

Lista de Figuras

Resumo

Com o objetivo de padronizar a área de gerência de redes de telecomunicações, o ITU-T desenvolveu uma estrutura organizada de rede denominada TMN, que possibilita a interligação dos diversos sistemas existentes, permitindo às empresas de telecomunicação migrarem para um sistema de gerência distribuído e padronizado. Esse modelo está sendo aceito gradativamente por empresas operadoras de serviços e por fabricantes de equipamentos de telecomunicação.

Este trabalho consiste no desenvolvimento de um sistema de informação para suporte à gerência de falhas com o objetivo de padronizar os relatórios de alarmes emitidos pelas centrais, atendendo ao padrão da recomendação [X.733] do ITU-T. Foi utilizada a tecnologia AXE10 da Ericsson para o desenvolvimento deste trabalho. São apresentados os conceitos da TMN, os conceitos de supervisão de alarmes e a descrição da central AXE-10. Na descrição da central AXE-10 é apresentado os seus subsistemas, o formato do alarme emitido pela central e as informações contidas nos alarmes.

Abstract

Aiming the standardization of the telecommunication nets management area the ITU-T developed an organized net structure called TMN, which allows the linking of the several existent systems, permitting the telecommunication companies migrate to a distributed and standardized management system. This model has been gradually accepted by telecommunications service suppliers and equipment manufacturers.

This work consists in the development of an information system to support to the fault management with the objective of standardizing the alarms reports emitted by the switching, following the pattern [X.733] recommendation by ITU-T. The technology AXE10 of Ericsson was used for the development of this work. The TMN concepts, the alarms supervision concepts, and the AXE-10 switching description are presented. In the AXE-10 switching description its subsystems, the alarm format emitted by the switching and the information contained in the alarms are presented.

1 INTRODUÇÃO

O ambiente de telecomunicações está em constante mudança através do desenvolvimento tecnológico, do crescimento do número de clientes e da globalização, que tem levado a um dinamismo e a um ambiente imprevisível. Há pouco tempo, as operadoras se preocupavam em aproveitar a máxima capacidade de transmissão dos sistemas existentes e muito pouco com a qualidade do sistema e dos serviços. Esse perfil já vem mudando, há alguns anos, pois a habilidade de entender e antecipar as necessidades dos clientes e a rapidez em adaptar os serviços exigidos pelo mercado a preços competitivos se fazem necessárias para a sobrevivência nessa área.

O gerenciamento de redes é um fator importante no fornecimento de serviços na indústria de telecomunicações, já que a disponibilidade dos serviços é afetado pela eficiência do gerenciamento da rede.

Talvez a área onde a implicação dessas mudanças seja mais evidente é a Rede de Gerência de Telecomunicação (TMN), visto que, no início, os sistemas de gerência direcionavam para a habilidade de operar grandes e complexas redes e enfocavam aspectos básicos de Operação, Administração e Manutenção (OAM), que estavam mais ligados aos aspectos internos e quase invisíveis ao mundo externo e aos clientes. Esses aspectos ainda mantêm sua importância, porém muitos outros novos aspectos estão entrando em foco. O suporte para o serviço de provisionamento é um deles, permitindo um desenvolvimento rápido de novos serviços, a gerência da rede de clientes, o aumento da eficiência e economia de custos das operações, os quais podem ser alcançados por meio de sistemas de operação.

Há alguns anos, o maior avanço em relação aos sistemas de gerência de rede foi devido a adoção do Modelo Orientado-a-Objetos Gerente/Agente, definindo um Modelo de Informação de Objetos Gerenciáveis. Apesar das vantagens da utilização do mesmo, a implementação do padrão OSI de gerência baseada em TMN foi lenta, devido ao

valor, à complexidade dos sistemas já instalados e aos custos envolvidos na substituição desses sistemas, fazendo com que a implementação da TMN fosse feita de forma gradativa.

Mesmo que as perspectivas técnicas em termos de cobertura de padronização, que podem ser visíveis nos sistemas desenvolvidos e plataformas computacionais sejam muito animadoras, não se deve esquecer o desafio que a implementação da TMN ainda apresenta. Isso se deve ao crescimento da rede de telecomunicação, através da aquisição de equipamentos de diferentes fabricantes e geralmente com protocolos de comunicação incompatíveis entre si, criando um parque heterogêneo.

Esse parque heterogêneo se formou com o aumento da demanda nas telecomunicações, que fez com que as operadoras investissem em equipamentos para atender a crescente demanda. Sendo estatais, eram obrigadas a adquiri-los por meio de licitações e, com isso, foram instalando equipamentos de diferentes fabricantes com tecnologia proprietária. Isso dificulta a monitoração e integração dos dados em virtude da incompatibilidade dos protocolos de comunicação utilizados pelos fornecedores.

Desta forma, o setor de telecomunicação se viu diante dos problemas gerados na administração de suas redes com equipamentos não padronizados, além da necessidade de fornecer novos serviços com rapidez e qualidade.

A necessidade da gerência integrada da rede criou o anseio à padronização das informações. Devido ao parque heterogêneo e aos sistemas de gerência proprietários, as empresas do setor dispõem de duas opções para implantar a gerência integrada. A primeira, consiste na troca do parque instalado, substituindo os componentes não padronizados por padronizados. A outra opção é adaptar as informações através de sistemas, que coletam os dados e adaptam ao padrão especificado. Entre as duas opções, a segunda é a mais viável para as empresas uma vez que esta proporciona um custo reduzido em relação à primeira.

Uma das principais dificuldades é a organização e correlação dos alarmes emitidos pelos equipamentos, uma vez que estes possuem formato e conteúdo diferentes. Com isso há a necessidade de coletar e adaptar o seu conteúdo para um formato padronizado. Este trabalho apresenta um sistema para adaptação dos formatos dos alarmes emitidos pela central AXE-10 da Ericsson, para o formato definido na

recomendação [X.733] do ITU-T. A tecnologia AXE-10 foi escolhida para o desenvolvimento deste trabalho devido à sua abrangência de uso nas operadoras.

Este trabalho está organizado em 7 capítulos :

O capítulo 2 faz um breve relato da redes de telecomunicações quanto à mudança da tecnologia analógica para digital, os componentes que compõem uma rede de telecomunicações, mostrando a hierarquia e a importância das centrais de comutação na rede.

O capítulo 3 é uma descrição dos conceitos da TMN, suas arquiteturas e áreas funcionais. A apresentação da TMN nesse capítulo se deve ao fato da supervisão de alarmes ser um item da gerência de falhas, uma das áreas funcionais da TMN.

O capítulo 4 faz uma descrição da supervisão de alarmes apresentando as funções e classes de objetos definidos pelas recomendações do ITU-T. Os relatórios de alarmes e seus parâmetros são apresentados na recomendação [X.733], que serão utilizados para a padronização do relatório de alarme da central.

O capítulo 5 apresenta a central AXE-10 da Ericsson, os seus subsistemas e a apresentação do formato dos relatórios de alarmes emitidos, os quais se propõem a converter ao padrão descrito no capítulo 4.

No capítulo 6 é apresentado o sistema desenvolvido, o qual é complementado pelos anexos. O anexo A contém a descrição das classes do sistema, o anexo B apresenta as tabelas que auxiliam o sistema e o anexo C contém o código fonte.

O Capítulo 7 trata das conclusões e sugestões para trabalhos futuros.

2 REDES DE TELECOMUNICAÇÕES

Neste mais de um século de existência, as redes de telecomunicações sofreram várias transformações tecnológicas, aumento da área geográfica e do número de assinantes, pois, conforme dados da ANATEL no ano 2000 o Brasil tinha 37 milhões de acessos fixos e 23 milhões de acessos móveis, abrangendo todas as regiões do Brasil.

Inicialmente, as redes de telecomunicações utilizavam a tecnologia analógica, tanto para o sistema de comutação como para o sistema de transmissão. Nessa fase, o serviço essencial é o telefônico, de comutação de voz, existindo, no entanto, em paralelo, comutação de texto através da rede de *telex* e posteriormente, também, transmissão de dados através da própria rede telefônica com o uso de modems.

Com o desenvolvimento tecnológico, é introduzida a transmissão digital na rede telefônica, mas a comutação continua sendo analógica. O sistema mais utilizado na transmissão digital é o PCM (*Pulse Code Modulation*), sendo um sistema simples de conversão de um sinal de voz analógico em uma cadeia de sinais digitais.

Há várias vantagens na transmissão digital. Podemos destacar as seguintes :

- eliminação quase total do ruído de transmissão, resultante da tolerância ao ruído da codificação digital e da introdução de regeneradores ao longo da linha;
- recuperação do sinal analógico sem atenuação, devido à digitalização;
- melhor adequação para sinalização por canal comum, devido ao aumento de capacidade do canal, maior versatilidade da sua utilização e possibilidade de detecção e controle de erros.

Numa fase seguinte, a comutação e a transmissão são totalmente digitais, as vantagens da integração da comutação e da transmissão digitais são consideráveis. Na parte técnica, devido à uniformização da tecnologia digital utilizada; e na área

econômica, devido à diminuição da complexidade das interfaces entre os dois sistemas, utilizando uma tecnologia de custo mais baixo.

Devido a isso, as redes de telecomunicações utilizam a transmissão digital PCM, a comutação digital, o controle das centrais de comutação por programa armazenado, o uso da sinalização por canal comum e a linha do assinante, que se mantém com tecnologia analógica.

O próximo passo das redes de telecomunicação seria a digitalização da linha do assinante, possibilitando digitalizar toda a rede de telefonia, permitindo, que os assinantes tenham acesso a canais de transmissão com capacidade maior e atender aos serviços que demandam uma carga de transmissão alta, como vídeo interativo.

Uma rede de telecomunicação é composta por uma diversidade de componentes e esses podem ser divididos em três categorias [Minoli96]: os equipamentos terminais, os equipamentos de transmissão, e os equipamentos de comutação (*switches*).

Os equipamentos terminais são os aparelhos telefônicos, centrais de pabx, fax, modems, ou gateway, que pertencem ao próprio assinante. Esses equipamentos estão conectados a uma central de comutação e tais conexões são denominadas linha de assinante, *loop* ou simplesmente linha.

Os equipamentos de comutação ou centrais de comutação (*switches*) são os equipamentos nos quais estão ligados os equipamentos terminais, ou as centrais de trânsito, que fazem a conexão entre duas centrais locais. As linhas que fazem a conexões entre duas centrais são chamadas de *trunks* nos Estados Unidos, de *junctions* na Europa e de "troncos" no Brasil. Um tronco é capaz de transportar uma ligação telefônica no padrão PCM, que necessita de uma banda larga , 64 kbits/s, para representar um sinal de voz. O PCM foi especificado pelo ITU-T na recomendação G.711.

O equipamento de transmissão precisa adequar o sinal a ser transmitido ao meio de transmissão a ser utilizado, que pode ser par trançado, cabos coaxiais, microondas de rádio, satélites, fibras óticas e infravermelho. Cada um desses meios de transporte tem sua aplicação conforme as circunstâncias do meio e distâncias.

A multiplexação é utilizada na transmissão para permitir que vários canais de voz possam ser transmitidos sobre um único meio. A multiplexação consiste em fracionar um meio de transmissão, alocando cada fração do meio a um assinante quando se fizer necessário. Uma das formas de multiplexação consiste em dividir-se a banda de freqüência do canal de maior velocidade em várias bandas mais estreitas. Esta forma de multiplexação é conhecida como FDM (*Frequency Division Multiplexing*). Uma outra forma mais sofisticada consiste em amostrar cada linha oriunda de um terminal seqüencialmente, enviando o sinal recebido por um canal de alta velocidade. Essa forma é conhecida como *TDM - Time Division Multiplexing*.

Dentre os elementos citados, o de maior peso para as empresas operadoras são as centrais de comutação, pois elas são responsáveis por toda a operação e controle dos serviços de telecomunicação. Elas fazem a conexão de um assinante de origem a um assinante destino através de um endereço específico.

Daniel Minoli [Minoli91] apresenta as centrais divididas em uma hierarquia de 5 níveis, mais as centrais tandem local, ilustradas pela Figura 1. Esta hierarquia é adotada pela Bell Systems, sendo que não é necessariamente empregada por outras companhias.

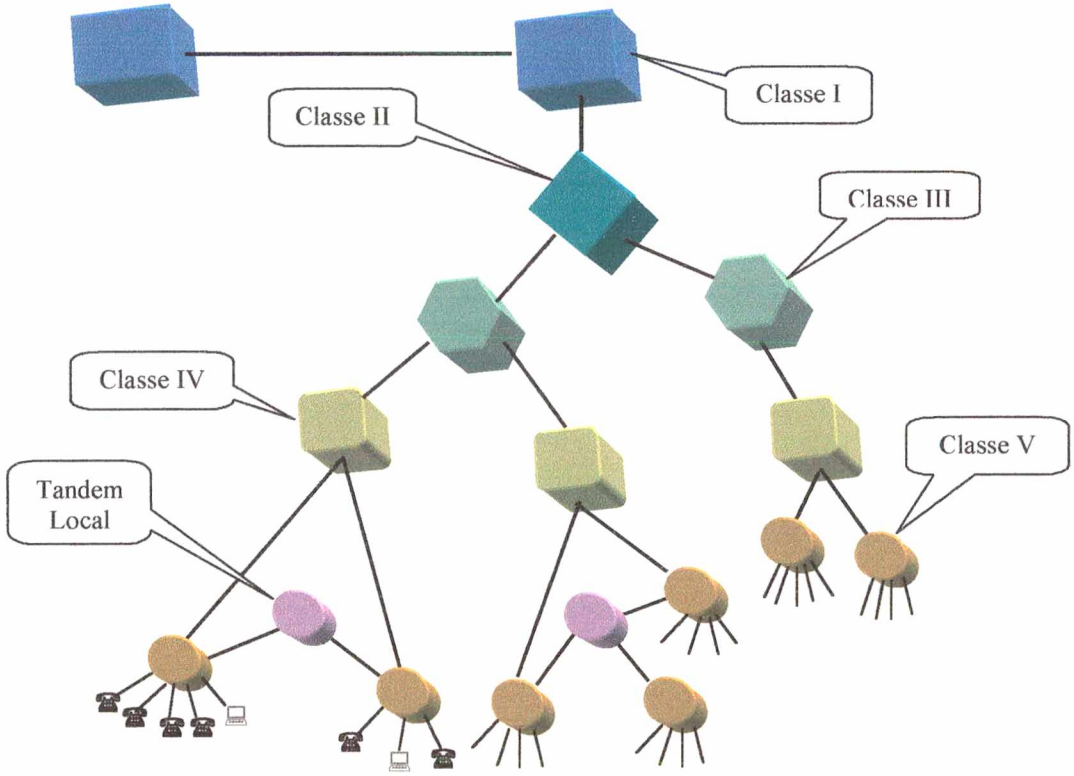


Figura 1 : Hierarquia das centrais de comutação

Classe I : Central de trânsito interurbana pertencente ao nível mais elevado de uma rede que empregue um plano de encaminhamento hierárquico. Estas centrais interligam pelo menos uma central internacional através de grupos de circuitos diretos, dimensionados com baixa probabilidade de perda.

Classe II : Usada para o tráfego entre as operadoras, é uma central de trânsito interurbana diretamente subordinada a uma central de trânsito classe I.

Classe III : Usada para tráfego interurbano regional, fica diretamente subordinada a uma central de classe II.

Classe IV : Usada para o tráfego interurbano regional e subordinada diretamente a uma central de classe III.

Classe V : Também chamadas de centrais locais. É nesta que as linhas de assinante estão conectadas.

Central Tandem Local : Central de trânsito que interliga centrais locais de uma área, servem como provedoras de caminhos alternativos entre as centrais locais, não possuem conexão direta com os assinantes.

Os custos das linhas telefônicas são muito elevados, principalmente se considerarmos entre duas cidades, dois estados ou dois países. Uma conexão entre duas centrais de duas cidades não comporta todos os assinantes por ser muito dispendioso e a mesma não é totalmente utilizada, já que nem todos os assinantes de um município irão ligar para o outro município ao mesmo tempo.

O canal de transmissão disponibilizado entre duas cidades é capaz de atender a um número máximo de chamadas simultâneas. Quando um assinante liga para uma outra cidade, ele utiliza uma fração do canal de transmissão. Ao finalizar a ligação, essa fração é liberada para que um outro assinante possa utilizá-la, permitindo a alocação dinâmica de cada fração do canal de transmissão.

Devido às mudanças e aperfeiçoamento da tecnologia dos equipamentos que compõem uma rede de telecomunicações, as operadoras, dentro do possível, mantiveram-se atualizadas adquirindo equipamentos com tecnologia proprietária. A dificuldade da gerência desta rede veio em decorrência dos protocolos e interfaces não compatíveis. No capítulo seguinte é apresentada a TMN que tem como objetivo introduzir um padrão para a gerência das redes de telecomunicações.

3 REDE DE GERÊNCIA DE TELECOMUNICAÇÕES – TMN

Inicialmente, o objetivo das companhias de telecomunicação era o de ampliar e modernizar as redes de telecomunicação para atender à crescente demanda do setor, mas a complexidade dessas redes e a necessidade de gerenciar esses recursos, que compõem a rede de telecomunicações, trouxeram a necessidade da introdução de uma rede de gerenciamento de telecomunicações para poder planejar, instalar, operar, manter e administrar os recursos das redes de telecomunicações instaladas, oferecendo serviços de alta qualidade.

Com esses objetivos, o CCITT, atual ITU-T, iniciou, em 1985, a padronização da Gerência de Redes de Telecomunicações (TMN) através do Grupo de Estudo IV. Em 1988, é editada a primeira recomendação chamada de M.30, como parte do "blue books" [Pras99], e em 1992, é editada uma recomendação completamente revisada, a série M.3000, e essa foi novamente alterada em 1996.

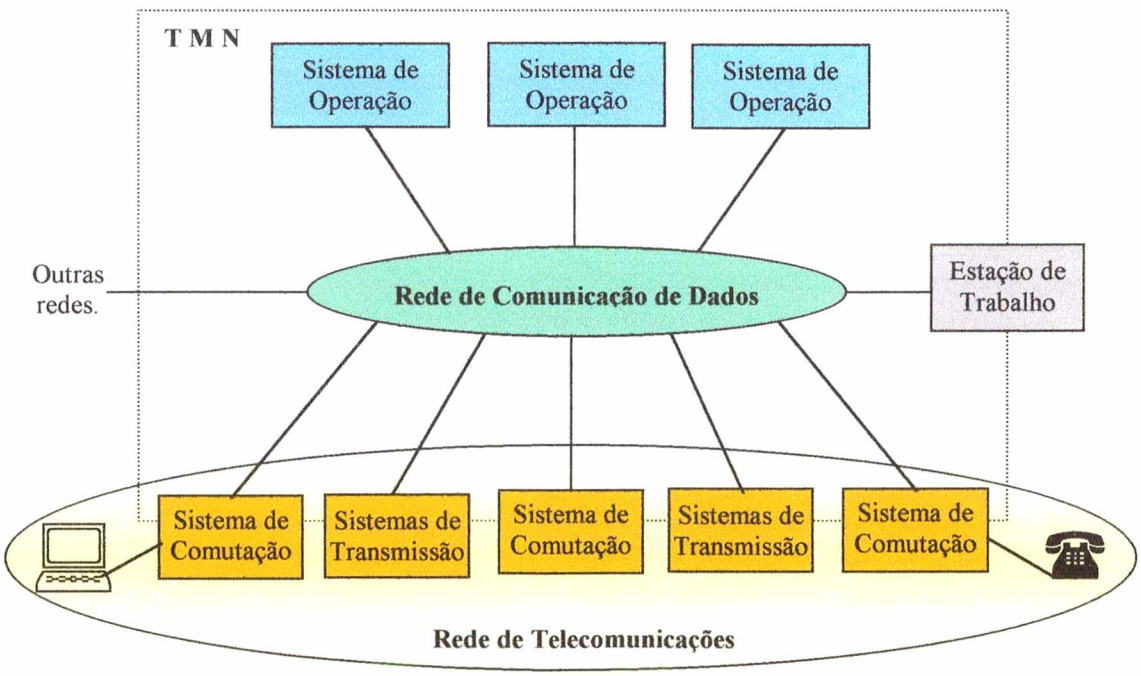


Figura 2 : Relacionamento geral entre a TMN com a Rede de Telecomunicações.

A TMN foi desenvolvida como uma infra-estrutura para suportar gerência e desenvolvimento de serviços de telecomunicações dinâmicos, provendo uma estrutura para alcançar a interconectividade e comunicação através de sistemas de operação e de redes de telecomunicação heterogêneos. Ela possibilita uma gerência de rede padronizada, flexível, escalável, mais barata e mais fácil para a comunicação através das redes [Vertel].

Conceitualmente, a TMN é uma rede separada que se comunica com uma rede de telecomunicações através de diversos pontos. Esses pontos são os comutadores e sistemas de transmissão que, por sua vez, estão conectados a um ou mais Sistemas de Suporte à Operação (OS) através de uma Rede de Comunicação de Dados (DCN), como mostra a Figura 2. A DCN também permite a conexão de estações de trabalho (WS), possibilitando, assim, que os operadores possam interpretar as informações de gerência. A TMN pode fazer uso dos recursos da própria rede de telecomunicação, assim haverá a necessidade da TMN gerenciar a própria TMN.

Há três aspectos básicos da arquitetura que podem ser considerados separadamente quando planejamos e projetamos uma TMN. esses aspectos são [M3010]:

Arquitetura Funcional;

Arquitetura de Informação;

Arquitetura Física.

3.1 ARQUITETURA FUNCIONAL

A arquitetura funcional define a distribuição apropriada das funcionalidades dentro da TMN, tendo como base : os Blocos Funcionais, os Componentes Funcionais, os Pontos de referência e a Função de Comunicação de Dados (DCF), conforme apresentados na Figura 3.

A TMN provê os meios para o processamento e transporte das informações de gerência, assim os blocos fornecem a capacidade de executar as funções de gerenciamento da TMN e a DCF provê as necessidades de transporte.

Dois blocos que trocam informações são separados por um ponto de referência. A definição dos blocos funcionais e pontos de referência entre os blocos leva à especificação da interface nos padrões da TMN [BRISA 92].

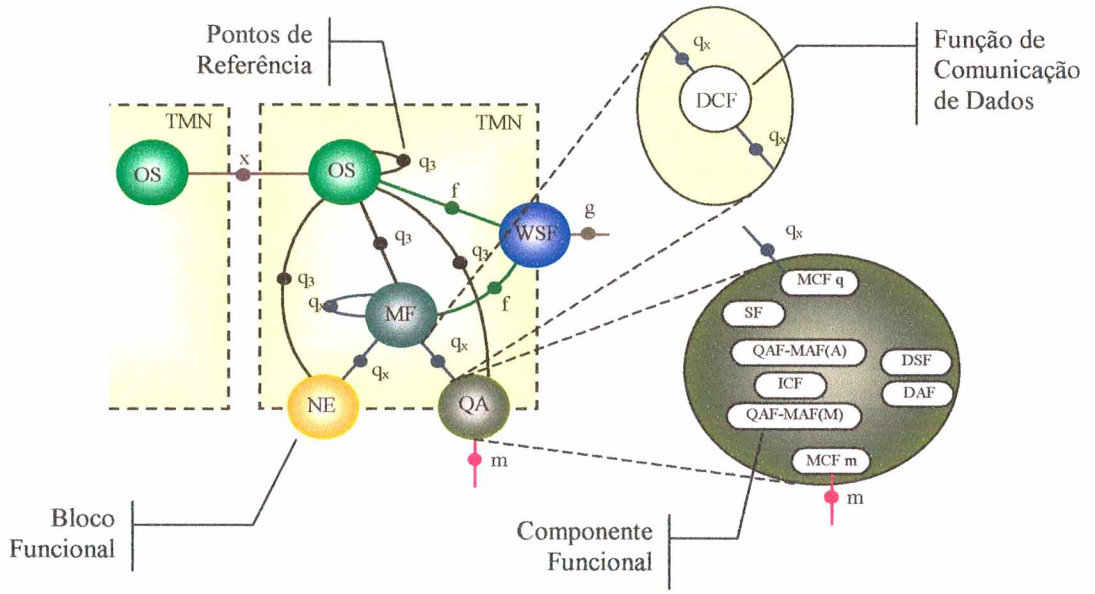


Figura 3 : Arquitetura Funcional TMN

3.1.1 Os Blocos Funcionais

A arquitetura funcional é baseada em blocos funcionais, que provêem funções que permitem a TMN executar as funções de gerenciamento. Esses blocos permitem implementar uma TMN de qualquer complexidade, não sendo necessário que todos os blocos estejam presentes em cada configuração TMN. Os blocos OSF e MF são especificados totalmente pelas recomendações TMN, enquanto que os blocos WSF, QAF e NEF são especificados parcialmente pela TMN, como é ilustrado pela Figura 3.

3.1.1.1 OSF – Sistema de Suporte à Operação (*Operations System Function*)

Processa as informações relativas à gerência de telecomunicações, com o propósito de monitorar, coordenar e/ou controlar as funções de telecomunicação e/ou de gerenciamento.

3.1.1.2 NEF – Função de Elemento de Rede (*Network Element Function*)

Se comunica com a TMN com a finalidade de ser monitorado e/ou controlado, provendo as funções de telecomunicações e de suporte que são requeridas pela rede de telecomunicações gerenciada. Estas funções não são parte da TMN, mas são representadas pela NEF para a TMN.

3.1.1.3 WS – Função de Estação de Trabalho (*Workstation Function*)

Provê os meios para interpretar informações TMN para o usuário humano e vice-versa.

3.1.1.4 MF – Função de Mediação (*Mediation Function*)

Traduz as informações trocadas entre OSF e NEF ou QAF, garantindo que a informação esteja de acordo com as expectativas dos blocos funcionais conectados ao MF, podendo armazenar, adaptar, filtrar, "*threshold*" e condensar informações.

3.1.1.5 QAF – Função Adaptador Q (*Q Adaptor Function*)

Conecta à TMN aquelas entidades que não suportam pontos de referência padrão TMN. A responsabilidade do QAF é a tradução entre um ponto de referência TMN e um não-TMN, por isso que uma parte deste bloco funcional é apresentada fora dos limites da TMN.

3.1.2 Componentes Funcionais

Os componentes funcionais representam os elementos básicos dos blocos funcionais TMN descritos anteriormente.

3.1.2.1 MAF – Função de Aplicação de Gerência (*Management Application Function*)

Implementa os serviços de gerenciamento, podendo assumir o papel de Gerente ou Agente. Dependendo do bloco funcional na qual está contida, ela pode ser denominada: OSF-MAF, MF-MAF, NEF-MAF, QAF-MAF.

3.1.2.2 ICF – Função de Conversão de Informação (*Information Conversion Function*)

Usado como sistema intermediário para prover mecanismos de tradução entre os modelos de informação e outra interface, sendo que a tradução pode ser feita a nível de sintaxe e/ou de semântica. Estes modelos de informação podem ou não ser orientados a objetos.

3.1.2.3 WSSF – Função de Suporte à Estação de Trabalho (*Workstation Suport Function*)

Provê suporte para o WSF, incluindo acesso e manipulação a dados, invocação e confirmação de ações, transmissão de notificações, e também prover suporte administrativo para o WSF e acesso para administrar o OSF.

3.1.2.4 UISF – Função de Suporte a Interface de Usuário (*User Interface Support Function*)

Transforma as informações contidas no modelo de informação TMN para um formato apresentável à interface homem-máquina e vice-versa. Integra as informações de uma ou mais seções com um ou mais OSFs ou MFs, assim que a informação esta presente em uma forma correta e consistente na interface do usuário. Funções similares ao MAF e ICF podem ser providas pelo UISF.

3.1.2.5 MCF – Função de Comunicação de Mensagens (*Message Communication Function*)

Está associada a todos os blocos funcionais que tenham um interface física. Seu uso limita-se à troca de informações contidas nas mensagens de gerenciamento entre esses blocos. A MCF é composta de uma pilha de protocolos que permitem a conexão de blocos funcionais às funções de comunicação de dados (DCF). O MCF pode prover funções de convergência de protocolo para interfaces onde nem todas as camadas do modelo OSI são suportadas. Diferentes tipos de MCF podem existir, dependendo do ponto de referência ao qual estão associados.

3.1.2.6 DSF – Função de Sistema de Diretórios (*Directory System Function*)

Representa um sistema de diretório, armazenando as informações de diretório como um conjunto de objetos de diretórios (DOs – *Directory Objects*) que são hierarquicamente ordenados. Pode ser implementado, construindo um ou mais agentes de sistema de diretório (DSAs – *Directory System Agents*).

3.1.2.7 DAF – Função de Acesso à Diretório (*Directory Access Function*)

Faz-se necessário aos blocos funcionais que necessitam acessar o diretório, pois permite o acesso e manutenção das informações relacionadas à TMN que são representadas na Base de Informações de Diretório (DIB *Directory Information Base*).

3.1.2.8 SF – Função de Segurança (*Security Function*)

Provê serviços de segurança que são necessários para os blocos funcionais a fim de satisfazer a política de segurança e requisições dos usuários.

3.1.3 Pontos de Referência.

Os Pontos de referência definem a fronteira entre dois blocos funcionais que trocam informações entre si, identificando as informações trocadas entre os blocos funcionais.

A definição dos blocos funcionais e os respectivos pontos de referência definem as interfaces padrões da TMN.

Pontos de referência TMN : $\mathbf{q_x}$, $\mathbf{q_3}$, **f** e **x**

Pontos de referência não TMN : **g** e **m**.

A Figura 3 ilustra todos os pares possíveis entre os blocos funcionais, e o respectivo ponto de referência.

3.1.4 Função de Comunicação de Dados – DCF (*Data Communication Function*)

Fornece os meios necessários para o transporte de informações entre os blocos funcionais, podendo também fornecer roteamento, retransmissão e funções de interfuncionamento. O DCF fornece as funções das camadas 1, 2 e 3 do modelo OSI [**M.3010**].

3.2 ARQUITETURA FÍSICA

A arquitetura física da TNM, ilustrada na Figura 4, define os blocos construtivos e as interfaces que permitem interligá-los. Esses blocos representam implementações físicas de funcionalidade da TMN.

A arquitetura TMN deve ser projetada de modo a evitar que falhas impossibilitem a transferência de mensagens críticas de gerenciamento e evitar que congestionamentos na DCN não causem bloqueios ou retardos excessivos de mensagens de gerenciamento que visam corrigir a situação.

3.2.1 Os Blocos Constitutivos

A TMN tem os seguintes blocos constitutivos.

3.2.1.1 OS – Sistema de Suporte à Operação (*Operations System*)

O OS é o sistema que executa OSFs. O OS pode opcionalmente prover MFs, QAFs e WSFs.

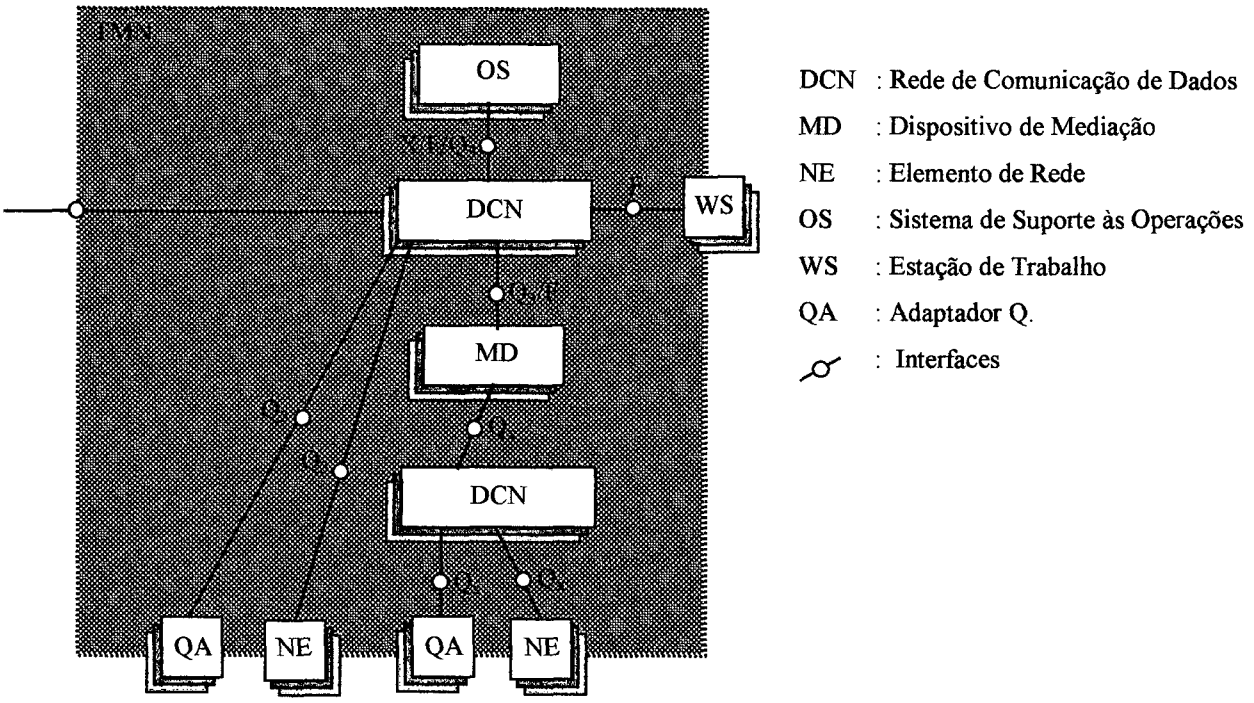


Figura 4 : Arquitetura Física TMN

A arquitetura física dos OSs deve possibilitar a centralização ou distribuição das funções de dados tratados pelos mesmos. Essas funções incluem:

a) programas de aplicação de suporte;

b) funções de banco de dados;

c) suporte aos terminais de usuários;

d) programas de análise;

e) formatação de dados e relatórios.

A arquitetura funcional dos OSs pode ser implementada em diferentes números de OSs, dependendo do tamanho da rede gerenciada, da funcionalidade requerida e da confiabilidade. Normalmente, as funções de OS são implementadas em um conjunto de OSs, comunicando-se via Q3, isso não impede que tais funções sejam implementadas num sistema com funções de NE ou MD.

3.2.1.2 MD – Dispositivo de Mediação (*Mediation Device*)

O MD é o dispositivo que implementa as MFs. Esse dispositivo atua sobre a troca de informações entre NEFs, QAFs e os OSFs, e provê funcionalidade de gerenciamento local para os NEs. Utiliza interfaces padronizadas mas pode ser implementado num sistema independente ou como parte do NE.

3.2.1.3 QA – Adaptador Q (*Q Adaptor*)

O QA é um dispositivo que conecta à TMN entidades similares a NEs e OSs, que não têm interfaces compatíveis com a TMN, através do ponto de referência m, às interfaces Q_x e Q_3.

3.2.1.4 DCN – Rede de Comunicação de Dados (*Data Communication Network*)

A DCN é uma rede de comunicação em uma TMN que deve, sempre que possível, seguir o modelo OSI. A DCN pode constituir-se de um número de sub-redes individuais de diferentes tipos interconectadas. A conexão física pode ser provida com enlaces constituídos a partir de diferentes tipos de componentes de rede, sendo que os enlaces podem ser de uso exclusivo da DCN ou compartilhados com outras redes de serviços.

3.2.1.5 NE – Elemento de Rede (*Network Element*)

O NE é composto por equipamentos de telecomunicações e equipamento de suporte ou qualquer item, ou grupo de itens considerados, que pertençam ao ambiente de telecomunicações que executem NEFs. O NE pode, opcionalmente, conter qualquer dos outros blocos funcionais TMN de acordo com sua implementação requerida. O NE tem uma ou mais interfaces-padrão tipo Q e pode opcionalmente ter interfaces F e X.

Existem equipamentos, como NE, que não processam uma interface-padrão TMN, obtendo acesso à TMN via uma Função Adaptador Q, que proverá a funcionalidade necessária para converter entre uma interface de gerência padrão e uma não-padrão.

3.2.1.6 WS – Estação de Trabalho (*Workstation*)

A WS é o sistema que executa WSFs. As funções da estação de trabalho traduzem as informações do ponto de referência f para um formato apresentável ao usuário, no ponto de referência g e vice-versa.

3.2.2 Conceito de Interface Interoperável

Para permitir que dois ou mais blocos constitutivos possam trocar informações de gerenciamento, eles precisam estar conectados ao mesmo meio de comunicação. Cada um desses elementos deve suportar a mesma interface desse meio usando, assim, o conceito de interface interoperável, evitando a proliferação de padrões de comunicação.

Uma interface interoperável define o conjunto de protocolos e de mensagens transportadas pelas respectivas unidades de dados de protocolo. As interfaces interoperáveis orientadas à transação estão baseadas na visão orientada a objeto da comunicação, na qual todas as mensagens transportadas dizem respeito à manipulação de objetos. Isso é formalmente definido pelo conjunto de protocolos, de procedimentos, do formato e da semântica das mensagens utilizadas para comunicação de gerenciamento.

3.2.3 Interfaces Padrão TMN

A interconexão dos vários blocos da TMN é feita através de um conjunto de interfaces interoperáveis padrão. Essas interfaces-padrão correspondem aos pontos de referência, e nesses é aplicado uma conexão física externa quando se fizer necessário.

3.2.3.1 Interface Q

A interface Q é aplicável no ponto de referência q para prover flexibilidade de implementação. A classe da interface Q é subdividida em duas subclasses, as quais se distinguem, basicamente, pelas informações que elas transportam.

A interface Q_x é aplicável ao ponto de referência q_x. Essa é caracterizada por aquela porção do modelo de informação, que é compartilhada entre o Dispositivo de Mediação (MD) e os Elementos de Rede (NEs).

A interface Q_3 é aplicável ao ponto de referência q_3. Essa é caracterizada por aquelas porções do modelo de informação compartilhada entre Sistemas de Suporte às Operações (OSs) e os elementos da TMN que realizam interface com eles.

Os modelos de informação para as interfaces Q_3 e Q_x podem ser, potencialmente, os mesmos, caso contrário, a Função de Mediação é necessária para prover a conversão entre eles.

3.2.3.2 Interface F

A interface F é aplicável ao ponto de referência f. Essa interface conecta as Estações de Trabalho (WSs) aos Sistemas de Suporte às Operações (OSs) ou aos Dispositivos de Mediação (MDs) através da Rede de Comunicação de Dados (DCN).

3.2.3.3 Interface X

A interface X é aplicável no ponto de referência x e é utilizada para interconectar duas TMNs ou para interconectar uma TMN com outros sistemas de gerenciamento de rede que possuem uma interface não-padronizada semelhante a da TMN. A interface X pode necessitar de maior segurança que a requerida para uma interface Q.

O modelo de informação na interface X estabelece os limites sobre o acesso disponível do outro lado da mesma. O conjunto de capacitação disponível na interface X para o acesso à TMN é referido como "acesso TMN".

3.3 ARQUITETURA DE INFORMAÇÃO

A arquitetura de informação é baseada no modelo orientado a objetos e, conforme o modelo OSI para gerência de sistemas, introduz o conceito de Gerente/Agente, domínios, conhecimento de gerenciamento compartilhado, necessários para a organização e o interfaceamento de sistemas de gerenciamento complexos.

3.3.1 Objeto Gerenciado - MO

Os sistemas de gerência de trocas de informações são modelados nos termos de objetos gerenciados (MOs), os quais são uma visão conceitual dos recursos que estão sendo gerenciados ou podem existir para suportar funções de gerência. Um MO também pode representar o relacionamento entre dois recursos ou a combinação de recursos. O conjunto dos MOs formam a Base de Informação de Gerenciamento (MIB – *Management Information Base*).

Não é necessário que um MO represente um único recurso real ou lógico, assim, um recurso pode ser representado por mais de um MO. Nesse caso, cada MO provê uma visão diferente desse objeto. Um MO pode prover uma visão de um recurso que já é representado por outro MO. Se um recurso não está sendo representado por um MO, esse recurso passa a não existir. Dessa forma, tal recurso não poderá ser gerenciado.

Os MOs que possuem características semelhantes são agrupados dentro de uma classe de objetos, que é um pacote. Cada pacote, que pode ser obrigatório ou condicional, pode conter atributos (*ATTRIBUTES*), operações (*ACTIONS*), notificações (*NOTIFICATIONS*) e comportamentos (*BEHAVIOUR*). Quando o pacote for obrigatório, deve estar presente em todas as instâncias de uma classe de objetos. Quando o pacote for condicional, ele só deverá estar presente em um objeto quando a condição associada à sua existência é satisfeita.

Atributo (*ATTRIBUTES*) : são os dados encapsulados em MO. Cada atributo corresponde a uma das características do recurso que o objeto representa. Cada atributo é constituído de um nome e tipo, e pode conter um ou mais valores.

Operações (*ACTIONS*) : São as ações que um MO pode executar, sendo que há as operações sobre atributo e as operações sobre o MO como um todo.

Operações sobre atributo :

- *get attribute value* : lê o valor de um atributo, ou uma lista de atributos especificados;
- *replace attribute value* : altera os valores dos atributos especificados com novos valores;

- *replace-with-default value*: altera os valores dos atributos especificados com seus respectivos valores padrão (*default*);
- *add member* : Adiciona os novos valores a um atributo do tipo conjunto (*sets*);
- *remove member* : remove os valores especificados de um atributo do tipo conjunto (*sets*).

Operações sobre o objeto como um todo :

- *Create* : Cria um novo MO com características da classe informada, e os valores dos atributos obrigatórios devem ser informados ou atribuídos com o valor padrão (*default*);
- *Delete* : Elimina um ou mais MOs;
- *Action* : Solicita a um MO que execute uma determinada ação e retorne o resultado.

Notificações (*NOTIFICATIONS*) **:** são as informações que os MOs reportam em resposta a algum evento interno ou externo.

Comportamento (*BEHAVIOUR*) : É a forma como os MOs interagem com os recursos modelados.

Hierarquias : Nas hierarquias, temos a Hierarquia de Herança de Classe e de nomeação, também conhecida como *Containment.*

Hierarquia de Herança de Classe: consiste em uma estrutura de classes de MOs, onde a hierarquia é organizada na especialização de classes. Isso permite que uma classe seja uma especialização de outra classe ou classes, dividindo a hierarquia em superclasses, que são as classes mais genéricas, e as subclasses, que são as classes mais especializadas, conforme apresentadas na Figura 5. Uma subclasse, além de herdar todas as características de sua superclasse, ainda pode ser acrescida de propriedades específicas, caracterizando a especialização.

O topo dessa hierarquia é determinado através de uma superclasse denominada de TOP.

A herança é a propriedade onde uma classe mais especializada (subclasse) herda todas as propriedades da classe mais geral (superclasse).

Hierarquia de Nomeação (*Containment*): define uma estrutura de relacionamento entre os objetos, onde um objeto estará contido em um outro objeto, e esse, por sua vez, poderá conter outros objetos.

O objeto de nível mais alto dessa estrutura é denominado *root* (raiz), que é um objeto que sempre existe.

Um objeto é identificado de forma única no seu nível, através *do Relative Distinguished Name* (RDN), e identificado na árvore pela concatenação de seu RDN com os RDNs dos objetos no qual ele está contido até o *root* (raiz), formando assim o *Distinguished Name* (DN).

A árvore criada pela hierarquia de nomeação é denominada de Árvore de Informações de Gerência, (MIT - *Management Information Tree*).

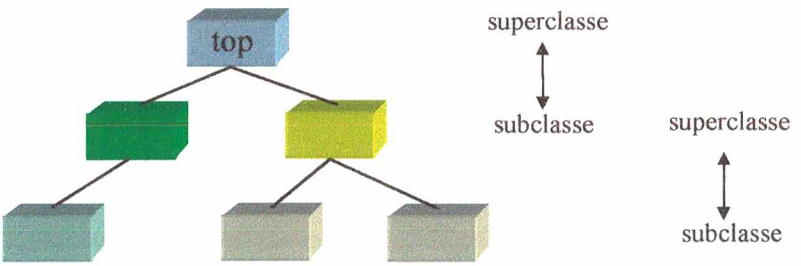


Figura 5 : Hierarquia de Herança de Classe

3.3.2 Gerente e Agente

A gerência de rede é uma aplicação distribuída, pois a rede de telecomunicação é um ambiente distribuído. A gerência envolve a troca de informações gerenciais entre os processos de gerência para a monitoração e controle dos vários recursos físicos e lógicos da rede. A Figura 6 mostra a interação entre o Gerente, Agente e os objetos gerenciados.

Gerente : coleta informações sobre os objetos gerenciados junto aos agentes, processa as informações, emite operações de gerência aos agentes e recebe notificações dos mesmos a fim de controlar o funcionamento do objeto gerenciado. Um Gerente pode atuar como um agente para um nível mais alto, respondendo às operações de gerenciamento recebidas do Gerente superior.

Agente : parte do processo da aplicação, que gerencia os objetos gerenciados associados, responde às operações de gerenciamento emitidas pelo gerente e reflete ao gerente uma visão dos objetos, emitindo notificações que espelham as ações destes objetos.

Um Agente pode rejeitar as operações de gerenciamento emitidas pelo Gerente, assim esse deve estar preparado para tratar as respostas negativas de um Agente.

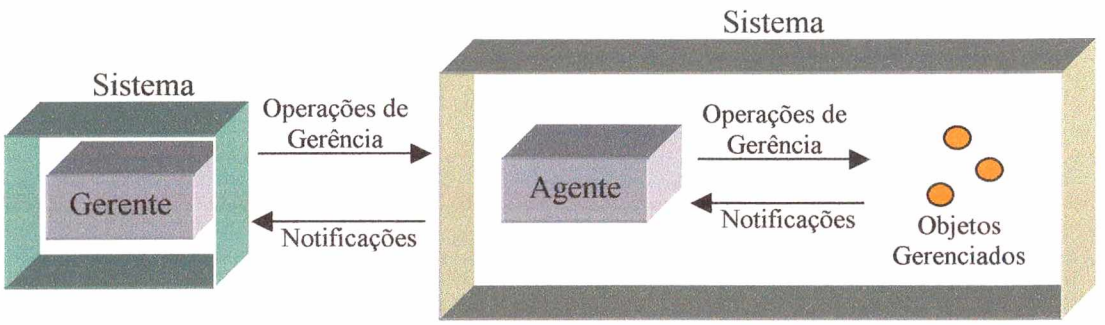


Figura 6 : Interação entre Gerente, Agente e Objetos Gerenciaveis

As informações são representadas através do paradigma de orientação a objetos. O conjunto de informações é organizado em uma base de dados denominada MIB – Management Information Base.

Todas as trocas de informações de gerência entre Gerente e Agente são expressas nos termos de um conjunto consistente de operações e notificações de gerência. Essas operações são todas realizadas através do uso do Serviço de Informação de Gerência Comum (CMIS – *Commom Management Information Service*) [X.710] e o Protocolo de Informação de Gerência Comum (CMIP – *Commom Management Information Protocol*) [X.711].

3.3.3 Conhecimento Compartilhado de Gerenciamento (SMK - *Shared Management Knowledge*)

Permite o interfuncionamento dos sistemas de gerenciamento através do compartilhamento das informações referentes aos protocolos, às funções de

gerenciamento, às classes de objetos suportadas, às instâncias disponíveis dos objetos gerenciados, às capacitações autorizadas e aos relacionamentos de *containment* entre os objetos. [M.3010]

3.4 ARQUITETURA FUNCIONAL EM CAMADAS

A complexidade da TMN cria a necessidade de subdividir funcionalmente a gerência em níveis. Com o propósito operacional, a funcionalidade da TMN pode ser considerada como particionada em camadas. Cada camada reflete aspectos particulares de gerência e o agrupamento das informações de gerenciamento suportadas por esses aspectos. O agrupamento dessas funcionalidades implica em gerentes específicos para cada camada. A arquitetira funcional é composta por 5 camadas hierárquicas como mostra a **Figura 7**.

A estratificação da TMN também permite aos usuários obterem diferentes níveis de informações, conforme as suas necessidades.

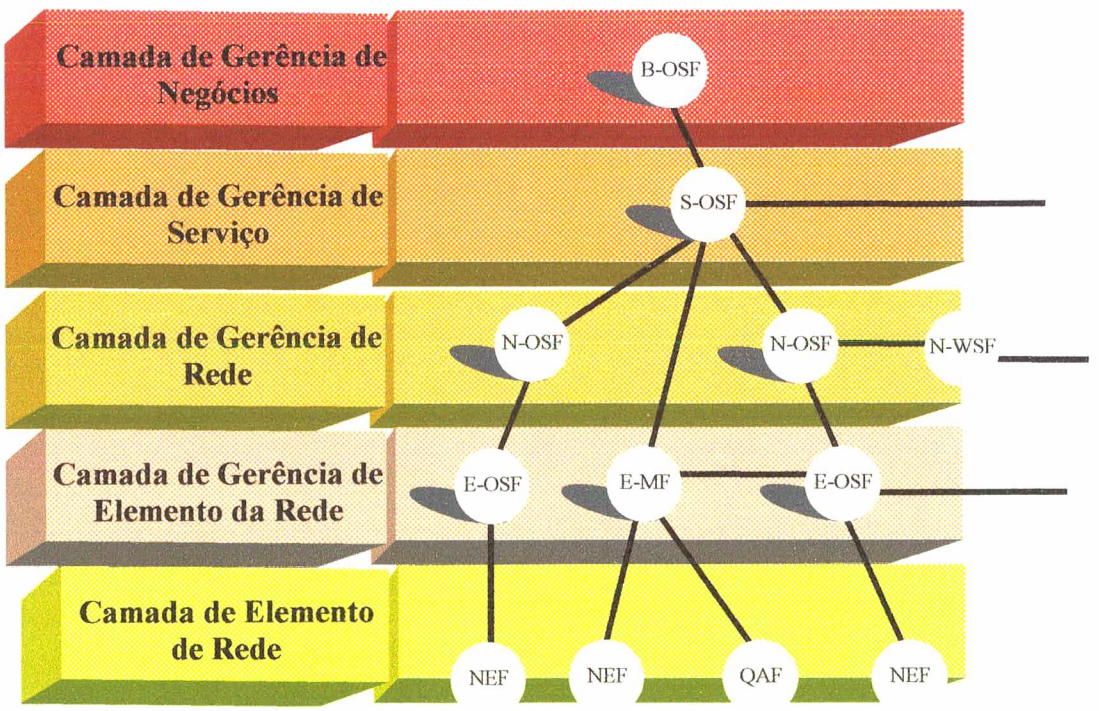


Figura 7 : Camadas Funcionais de Suporte de Gerenciamento

3.4.1 Camada Elemento de Rede

Corresponde aos recursos de telecomunicação que devem ser gerenciados. Esses recursos devem possuir agentes para que possam fornecer as informações necessárias ao sistema de gerência. É nessa camada que estão os blocos funcionais QAF e NEF.

3.4.2 Camada de Gerência de Elemento da Rede.

Gerencia cada elemento de rede individualmente e faz a abstração das funções fornecidas pela camada de elemento de rede (NE). Essa camada tem um conjunto de gerenciadores de elementos que são individualmente responsáveis por alguns subconjuntos de elemento de rede, em delegação da camada de Gerência de Rede. Cada gerenciador de elemento tem três funções principais:

a) controle e coordenação de um subconjunto de elementos de rede;

b) prover mediação para permitir que a camada de gerência de rede interaja com os elementos de rede;

c) manter dados estatísticos, registros e outros dados sobre elementos de rede.

3.4.3 Camada de Gerência de Rede

Tem a responsabilidade pela gerência de todos os elementos de rede e contém as funções e informações envolvidas na gerência global da rede de telecomunicações.

É composta dos sistemas destinados à operação, administração e manutenção de rede, tais como re-roteamento, detecção e isolamento de falhas e provisionamento de facilidades.

3.4.4 Camada de Gerência de Serviço

Essa camada está relacionada aos aspectos de interface com os clientes e realiza funções associadas com a prestação e gerenciamento dos serviços destinados à operação, administração e manutenção de serviços relacionados aos clientes.

3.4.5 Camada de Gerência de Negócio

É responsável pelo gerenciamento de todo o empreendimento. Pode ser vista onde são definidos os objetivos, antes que eles possam ser alcançados. Por essa razão, tal camada pode melhor ser relatada como gerência tática e estratégica, em vez de gerência operacional, como as outras camadas da TMN.

3.5 Áreas Funcionais

De forma a se englobar toda a funcionalidade necessária ao gerenciamento de uma rede de telecomunicações, tais como o planejamento, a instalação, a operação, a manutenção e o provisionamento, cinco áreas funcionais são identificadas:

3.5.1 Gerência de Desempenho

O gerenciamento de desempenho envolve as funções relacionadas com a avaliação e relato do comportamento dos equipamentos de telecomunicações e a eficiência da rede. Essas funções se dividem em :

- **Medidas de tráfego:** capacitam definir a entrega de relatórios de medida de tráfego;

- **Monitoramento de desempenho:** permite obter informações para avaliar e relatar parâmetros de desempenho da rede. Essas informações podem ser utilizadas como apoio ao diagnóstico de falhas, planejamento de rede e qualidade de serviço;

- **Controle da Gerência de Desempenho:** refere-se ao controle de parâmetros para a monitoração do desempenho e o controle de qualidade dos serviços, fornece os dados referentes aos parâmetros a serem controlados e envia comando para a reconfiguração dos mesmos;

- **Análise de Desempenho:** avalia o nível de desempenho das entidades envolvidas através de análises e processamento das informações de desempenho.

3.5.2 Gerência de Falhas

O gerenciamento de falhas engloba as funções que possibilitam a detecção, isolamento e correção de operações anormais na rede de telecomunicações. As falhas impedem que os sistemas de cumpram seus objetivos operacionais e podem ser transientes ou persistentes. As funções de gerenciamento de falhas podem ser divididas em:

- **Supervisão de alarmes:** gerenciamento de informações sobre as degradações de desempenho que afetam o serviço. Devido ao enfoque aos relatórios de alarmes dado nesse trabalho, o capítulo 4 descreve a supervisão de alarmes com mais detalhes.

- **Localização de Falhas:** quando a informação inicial, referente à localização de uma falha, for insuficiente, deverão ser adicionadas informações obtidas de rotinas de localização, sendo que essas rotinas podem fazer uso de sistemas de testes internos ou externos à TMN.

- **Correção de Falhas:** refere-se aos procedimentos executados para normalizar o serviço, ou partes do sistema com falha, através da transferência de informações aos processos, que usam recursos redundantes para substituir esses que estão acusando a falha.

- **Testes:** o usuário pode solicitar a execução de um teste específico, podendo inclusive estabelecer os parâmetros desse. Em alguns casos, o tipo e os parâmetros do teste podem ser designados automaticamente.

- **Administração de Anormalidade:** transfere os relatórios de anormalidades originados pelos clientes e o bilhete de anormalidade originado por uma verificação de detecção de uma falha pró-ativa. Também suporta ações para investigar e elucidar a anormalidade e prover acesso ao status do serviço e o progresso no esclarecimento de cada anormalidade.

3.5.3 Gerência de Configuração

O gerenciamento de configuração habilita o usuário a criar e modificar recursos físicos e lógicos da rede de telecomunicações. Suas funções são divididas em:

- **Engenharia e Planejamento da Rede:** possibilita a identificação e o controle do provisionamento de novos recursos necessários para a rede de telecomunicações. Uma Ordem de Serviço pode ser utilizada para solicitar novos recursos, físicos ou lógicos.

- **Instalação:** tem a finalidade de possibilitar que os recursos da rede possam ser criados, incluindo-os na rede de telecomunicação.

- **Planejamento e Negociação de Serviço:** introduz novos serviços e verifica a necessidade de novos serviços. Pode mudar as características de um serviço e desabilitar serviços.

- **Provisionamento:** são os procedimentos necessários para colocar um equipamento em serviço, exceto na instalação. Assim que o equipamento esteja pronto, os programas de suporte serão inicializados via TMN. O estado de uma unidade e os parâmetros selecionados também podem ser controlados pelas funções de provisionamento.

- **Status e Controle:** provê a capacidade de monitorar e controlar certos aspectos dos elementos de rede. Geralmente, a verificação do status é fornecida em conjunto com cada função de controle a fim de verificar que ação resultante foi realizada.

Informações de Recurso: funções que têm por finalidade apresentar a lista de recursos alocados, verificar a consistência da informação e obter informação sobre os recursos disponíveis.

3.5.4 Gerência de Contabilização

O gerenciamento de tarifação provê um conjunto de funções que possibilitam a determinação do custo associado ao uso da rede de telecomunicações. Algumas funções associadas ao gerenciamento de tarifação são: informar os custos associados aos recursos alocados, habilitar limites de tarifação, definir agendamentos a serem associados com a utilização dos recursos e combinar custos quando um requisito de comunicação exigir múltiplos recursos combinados:

- **Medição de Uso:** coleta dados dos recursos para serem usados a fim de determinar a carga de uso e determinar a conta dos clientes.

- **Tarifação:** pode estar centralizada numa rede inteligente ou distribuída nos comutadores para determinar o montante de serviços usados.

- **Cobrança e Finanças:** funcionalidade na trabsferência de dados para a TMN, com propósitos administrativos para as contas dos clientes, como data de pagamento e recebimento de pagamento.

- **Controle Empresarial:** fornece o fluxo de dados necessário sobre o próprio fluxo financeiro da empresa, e entre a empresa e seus credores.

3.5.5 Gerência de Segurança

A função do gerenciamento de segurança é a criação e controle de serviços e mecanismos de segurança, a distribuição de informações relevantes à segurança e o armazenamento de eventos relativos à segurança.

- **Prevenção:** é a função de prevenir a entrada de intrusos na rede.

- **Detecção:** é a função para detectar caso um intruso entre na rede.

- **Restrição e Recuperação:** são funções necessárias para cancelar o acesso a um intruso e reparar os danos causados por ele.

- **Administração de Segurança:** para o planejamento e administração do plano de ação de segurança e gerenciamento das informações relativas à segurança.

4 SUPERVISÃO DE ALARMES

A gerência de rede deve prover mecanismos para alertar situações anormais em algum ponto da rede. Esses mecanismos devem permitir verificar o funcionamento anormal ou a sobrecarga de um componente da rede para tomar as decisões que evitem problemas na mesma.

Para que a TMN possa executar a supervisão de alarmes, os Elementos de Rede (NEs) devem permitir a monitoração das condições de alarme perto do tempo real, ou em horários programados, consultas das condições de alarmes de um NE e o registro (*log*) e a recuperação das informações do histórico dos alarmes.

4.1 FUNÇÕES DE SUPERVISÃO DE ALARMES

As funções para a supervisão de alarmes permitem monitorar ou interrogar, ou ambos, os NEs sobre eventos ou suas condições. Um relatório de evento é gerado por um NE na detecção de uma condição anormal. O evento pode ser comunicado imediatamente ou armazenado em um arquivo de *log* para futuro acesso.

A seguir são apresentados os conjuntos de funções utilizadas para a gerência da supervisão de alarmes conforme descritas em [M.3400].

4.1.1 Funções de Política de Alarme

Esse conjunto de funções cria e atualiza tabelas de domínio de alarmes para centros e sistemas de supervisão. Também suporta tabelas para diversos níveis de rede que definem sob quais condições um alarme é inibido e qual o nível de severidade atribuído para determinadas condições de alarme.

4.1.2 Funções de Análise de Evento de Falha de Rede

Este conjunto de funções provê acesso a um sumário de alarmes ao nível de rede. Essas funções também permitem futuras reduções de alarmes redundantes. Essas reduções são feitas através da correlação e filtro de alarmes no contexto de um NE ou grupos de NEs. Esse conjunto de funções também suporta notificações de novos filtros, alarmes correlatos, ou mudanças no estado de um alarme previamente informados como, por exemplo, uma notificação de que um alarme já tenha sido eliminado.

4.1.3 Funções de Modificação do Estado do Alarme.

Essas funções habilitam ao usuário administrar as regras para a revisão do estado de alarme, como recebido de um NE antes de ser apresentado ou processado. As regras refletem a política de alarme que, por sua vez, refletem a experiência do campo.

4.1.4 Funções de Relatório de Alarme

Tais funções reportam alarmes e relatam informações associadas ao alarme tais como as causas do alarme, hora de ocorrência, valores limites ultrapassados.

Alarmes são tipos específicos de notificações relativas à detecção de falhas ou condições anormais. Uma notificação de alarme resulta de uma condição de alarme que persiste tempo suficiente para ser classificada como uma condição não transitória.

4.1.5 Funções de Sumarização de Alarme.

Essas funções reportam e controlam informações de um resumo das condições dos alarmes atuais de um objeto gerenciado específico. Suportam informar as condições de alarme em horários pré-determinados e/ou com base na demanda.

4.1.6 Funções de Critério para Eventos de Alarme

Gerenciam o critério usado pelos recursos para determinar uma certa condição a ser considerada um alarme. Essas funções determinam quando uma condição deve ser

considerada um alarme através da observação dos valores assumidos pelos atributos do MO que representa um NE.

4.1.7 Funções de Gerência de Indicação de Alarme

Essas funções controlam os indicativos de alarme visual e sonoro através de instruções que habilitam ou não a operação de alarmes especificados, ou a reinicialização de alarmes sonoros.

4.1.8 Funções de Controle de *Log*

As funções de controle de *log* descrevem os componentes conceituais para permitir o controle do armazenamento e recuperação das informações do histórico dos alarmes de um NE.

Para a supervisão de alarmes, se faz necessário preservar as informações dos relatórios de alarmes dos objetos gerenciados que ocorreram. O Registro de Alarme, no *log*, contém as informações de seu respectivo relatório de alarme.

4.1.9 Funções de Filtro e Correlação de Alarme

As notificações devem apresentar alarmes não-redundantes dentro do escopo de um elemento ou grupo de elementos de rede. Esse objetivo é alcançado através da correlação e filtro. A correlação envolve a interpretação e troca de estados que ocorrem na rede, elementos da rede, sistemas e equipamentos operacionais. Assim, uma troca de estado pode ter um significado por si só ou em conjunto com a troca, ou não, de outros estados.

4.1.10 Funções de Detecção e Informação de Eventos de Falha

Essas funções provêem acesso aos resultados das verificações de hardware e software, que são feitas executando as funcionalidades das telecomunicações em determinados horários através de processos em *background.*

Outras informações de funções de supervisão de alarme são apresentadas na recomendação X.734, que trata sobre Funções de Gerência de evento e na recomendação X.735, que trata sobre as Funções de Controle de Log.

4.2 SERVIÇOS DE SUPERVISÃO DE ALARMES

Define os serviços necessários para apoiar as funções de supervisão de alarmes especificados em 4.1. Supervisão de alarmes envolve os relatórios de alarmes e os sumários de alarmes, que são formulários especializados de relatórios de eventos e o registro dessas informações.

Os serviços definem o apoio às funções de supervisão de alarmes e têm sido agrupados em várias unidades funcionais para o seu uso em uma associação (durante o estabelecimento da associação) e para também permitir a referência por outras recomendações. A Negociação executada entre as unidades funcionais está descrita na recomendação [X.701]. Uma informação de usuário não específica TMN é fornecida durante a liberação ou aborto de uma associação.

4.3 CLASSES DE OBJETOS PARA SUPERVISÃO DE ALARMES

A seguir, são apresentadas as classes de objetos gerenciados para a supervisão de alarmes definidos pelas recomendações Q.821, X.721 e M.3100:

Alarm Severity Assignment Profile (M.3100) : é uma classe de objetos de suporte à gerência que especifica a severidade para um objeto gerenciado.

Alarm Record (X.721): é uma classe derivada da classe *Event log record* [X.721] que representa informações armazenadas em *logs* como resultado de relatórios de eventos do tipo alarme.

Event Forwarding Discriminator (X.721) : esta classe é usada para definir as condições que deverão ser satisfeitas por relatórios de eventos potenciais antes que os

relatórios de evento sejam repassados para um destino em particular. Essa classe é derivada da classe *Discriminator* [X.721].

Current Alarm Summary Control (Q.821) : é uma classe de suporte que provê critérios para a geração de relatórios sumarizados de alarmes correntes. Pode-se informar uma lista de objetos que deverão ser considerados para o sumário (se esta lista estiver vazia, todos os objetos serão considerados), listas de estados de alarme, severidade percebida e causa provável do alarme. Caso alguma dessas listas esteja vazia, ela não será considerada na avaliação para a geração do sumário de alarmes.

Management Operations Schedule (Q.821) : provê a habilidade de agendar serviços de gerência de forma periódica. A agenda é programada indicando-se o tempo para início do agendamento do serviço, a periodicidade com que se repetirá o serviço e o tempo para encerramento do agendamento.

4.4 RELATÓRIO DE ALARME

Três elementos estão envolvidos em um relatório de alarme, o tipo de alarme, as informações sobre o alarme e *event reply*. O *event replay, que* não será abordado neste trabalho, consiste na resposta que um gerente pode enviar para o agente emissor do alarme. O tipo de alarme e as informações evento alarme são os parâmetros que o sistema de suporte tem como objetivo atender, utilizando os relatórios de alarme emitidos pela central AXE-10. A central é descrita no capítulo 5.

4.4.1 Tipos de Alarme

Alarmes são comunicados através de registros de eventos [X.734] e podem ser guardados em registros de *log* [X.735]. Eventos e registros de *log,* na supervisão de alarmes, têm origem em relatórios de alarme reportados através de notificações. Eventos de alarmes podem ser classificados em tipos de eventos de alarme [X.733]. O relatório de cada tipo de evento de alarme se dá por uma notificação de alarme correspondente, conforme segue:

Communications alarm type. Esse tipo de alarme é associado com processos ou procedimentos responsáveis por carregar informações de um ponto a outro. Ocorre quando o objeto gerenciado detecta um erro de comunicação. A notificação que reporta esse tipo de evento é *communicationsAlarm*.

Quality of service alarm type. É associado com degradação da qualidade de serviço. A notificação que reporta esse tipo de evento é *qualityOfServiceAlarm*.

Processing error alarm type. É associado com falhas de processamento ou de software. A notificação que reporta esse tipo de evento é *processingErrorAlarm*.

Equipment alarm type. São associados a falhas em equipamentos. A notificação que reporta esse tipo de evento é *equipmentAlarm*.

Environmental alarm type. É principalmente associado com condições referentes ao ambiente onde está o equipamento. Ocorre quando um objeto gerenciado detecta um problema no ambiente. A notificação que reporta tal evento é *environmentalAlarm*.

4.4.2 Informações do Evento Alarme

Os eventos de notificação de alarme, definidos na recomendação X.733, seguem uma estrutura padrão que consta dos seguintes parâmetros:

- probableCause
- specificProblems
- perceivedSeverity
- backedUpStatus
- backUpObject
- trendIndication
- thresholdInfo
- notificationIdentifier
- correlatedNotifications
- stateChangeDefinition
- monitoredAttributes
- proposedRepairActions

- additionalText
- additionalInformation

Cada um dos parâmetros citados é explicado a seguir:

Probable Cause. Indica a causa ou causas prováveis que levaram a um relatório de um alarme. Podem ser classificados em um dos tipos de alarme definidos para *Alarm Type*. Quando não é possível estabelecer a causa provável do alarme, deverá ser usado o valor *indeterminate* para ProbableCause.

A seguir estão listadas as causas prováveis definidas na recomendação X.733:

- adapter error;
- application subsystem failure: A failure in an application subsystem has occurred (an application subsystem may include software to support the Session, Presentation or Application layers);
- bandwidth reduced: The available transmission bandwidth has decreased;
- call establishment error: An error occurred while attempting to establish a connection;
- communications protocol error: A communication protocol has been violated;
- communications subsystem failure: A failure in a subsystem that supports communications over telecommunications links, these may be implemented via leased telephone lines, by X.25 networks, token-ring LAN, or otherwise;
- configuration or customisation error: A system or device generation or customization parameter has been specified incorrectly, or is inconsistent with the actual configuration;
- congestion: A system or network component has reached its capacity or is approaching it;
- corrupt data: An error has caused data to be incorrect and thus unreliable;
- CPU cycles limit exceeded: A Central Processing Unit has issued an unacceptable number of instructions to accomplish a task;
- dataset or modem error: An internal error has occurred on a dataset or modem;
- degraded signal: The quality or reliability of transmitted data has decreased;
- DTE-DCE interface error: A problem in a DTE-DCE interface, which includes the interface between the DTE and DCE, any protocol used to communicate between the DTE and DCE and information provided by the DCE about the circuit;
- enclosure door open;
- equipment malfunction: An internal machine error has occurred for which no more specific Probable cause has been identified;
- excessive vibration: Vibrator or seismic limits have been exceeded;
- file error: The format of a file (or set of files) is incorrect and thus cannot be used reliably in processing;
- fire detected;
- flood detected;
- framing error: An error in the information that delimits the bit groups within a continuous stream of bits;
- heating/ventilation/cooling system problem;

- humidity unacceptable: The humidity is not within acceptable limits;
- I/O device error: An error has occurred on the I/O device;
- input device error: An error has occurred on the input device;
- LAN error: An error has been detected on a local area network;
- leak detected: A leakage of (non-toxic) fluid or gas has been detected;
- local node transmission error: An error occurred on a communications channel between the local node and an adjacent node;
- loss of frame: An inability to locate the information that delimits the bit grouping within a continuous stream of bits;
- loss of signal: An error condition in which no data is present on a communications circuit or channel;
- material supply exhausted: A supply of needed material has been exhausted;
- multiplexer problem: An error has occurred while multiplexing communications signals;
- out of memory: There is no program-addressable storage available;
- output device error: An error has occurred on the output device;
- performance degraded: Service agreements or service limits are outside of acceptable limits;
- power problem: There is a problem with the power supply for one or more resources;
- pressure unacceptable: A fluid or gas pressure is not within acceptable limits;
- processor problem: An internal machine error has occurred on a Central Processing Unit;
- pump failure: Failure of mechanism that transports a fluid by inducing pressure differentials within the fluid;
- queue size exceeded: The number of items to be processed (configurable or not) has exceeded the maximum allowable;
- receive failure;
- receiver failure;
- remote node transmission error: An error occurred on a communication channel beyond the adjacent node;
- resource at or nearing capacity: The usage of a resource is at or nearing the maximum allowable capacity;
- response time excessive: The elapsed time between the end of an inquiry and beginning of the answer to that inquiry is outside of acceptable limits;
- retransmission rate excessive: The number of repeat transmissions is outside of acceptable limits;
- software error: A software error has occurred for which no more specific Probable cause can be identified;
- software program abnormally terminated: A software program has abnormally terminated due to some unrecoverable error condition;
- software program error: An error has occurred within a software program that has caused incorrect results;
- storage capacity problem: A storage device has very little or no space available to store additional data;
- temperature unacceptable: A temperature is not within acceptable limits;
- threshold crossed: A limit (configurable or not) has been exceeded;
- timing problem: A process that requires timed execution and/or co-ordination cannot complete, or has completed but cannot be considered reliable;
- toxic leak detected: A leakage of toxic fluid or gas has been detected;

- transmit failure;
- transmitter failure;
- underlying resource unavailable: An entity upon which the reporting object depends has become unavailable;
- version mismatch: There is a conflict in the functionality of versions of two or more communicating entities which may affect any processing involving those entities.

Specific problems. Esse parâmetro, quando presente, identifica maiores detalhes para a causa provável. Pode ser um *ObjectIdentifier* ou um valor inteiro que identifique uma causa provável. A estrutura é semelhante à *ProbableCause* e usa as mesmas definições. Tal parâmetro auxilia no detalhamento da causa provável do alarme.

PerceivedSeverity. Define qual a severidade do alarme dentre seis níveis definidos a saber:

Cleared: indica que um ou mais relatórios de alarme foram limpos (*cleared*). Esse alarme limpa todos os alarmes para um objeto gerenciado que tem o mesmo *alarm type*, *probable cause* e *specific problems* (se existirem). Se o parâmetro de notificações correlatas (*Correlated Notifications*) não for nulo, os alarmes pendentes a serem limpos dizem respeito a todos aqueles identificados na lista de identificações de notificações (*Notification Identifier*). Essa lista contém a identificação de todas as notificações que têm correlação com o alarme que está em foco.

Indeterminate: Indica que o nível de severidade não pode ser determinado.

Critical: indica que ocorreu uma condição de alarme, que afeta o serviço e uma ação corretiva é necessária imediatamente.

Major: indica que a condição de alarme que está afetando o serviço se desenvolveu e uma ação urgente é requerida.

Minor: indica a existência de uma falha que não afeta as condições de serviço e que ações corretivas deverão ser tomadas para evitar falhas mais sérias.

Warning: Indica a localização de uma falha em potencial (ou *impending*) que pode afetar o serviço, antes que qualquer efeito mais significativo possa ser sentido. Nesse caso, é recomendável executar ações para um diagnóstico mais apurado e a conseqüente correção do problema para prevenir falhas mais sérias que afetem os serviços.

Backed-up status : esse parâmetro, quando presente, especifica se o objeto que emitiu o alarme foi passado para um *back-up* e os serviços, devido a isso, não foram interrompidos. O uso desse campo, em conjunto com o campo de indicação de severidade do alarme, provê informações de forma independente para qualificar a severidade do alarme e a habilidade do sistema de continuar trabalhando e fornecendo serviços.

Back-up object : Indica qual a instância de objeto que proveu os serviços no lugar do objeto com falha. É um atributo útil quando existe um *pool* de objetos disponíveis para operar como substituto. Nesse caso, o atributo indica qual deles foi usado.

Trend indication : Quando presente, esse parâmetro especifica a tendência da severidade atual especificada. Se presente, ele indica que existem um ou mais alarmes (fora esse) que não foram limpos (*cleared*) e pertencem ao mesmo objeto gerenciado. A indicação de tendência pode assumir três valores:

More severe : o *alarm severity* no alarme corrente é maior que o reportado em qualquer dos outros alarmes pendentes.

No change : indica que a severidade do alarme corrente é igual ao maior alarme existente entre os demais alarmes pendentes.

Less Severe : indica que a severidade do alarme corrente é menor que o maior alarme existente no conjunto de alarmes pendentes, ou seja, existe, pelo menos, um alarme mais grave nos pendentes que a severidade do alarme corrente.

Threshold information: tal parâmetro está presente quando o alarme é resultante de uma ultrapassagem de um *threshold* e tem quatro subparâmetros:

Triggered threshold : identificador do atributo de *threshold* que causou a notificação.

Threshold level : o valor de *threshold*, ou no caso de um atributo *gauge*, o valor do *threshold* ultrapassado e a correspondente *hysteresis*.

Observed value: valor do *gauge* ou contador que ultrapassou o *threshold*.

Arm time : Utilizado em um indicador de *threshold*, indica a hora em que o valor foi reinicializado, no caso de contador; ou hora em que o *threshold* foi re-armado, logo após a última ultrapassagem.

Notification Identifier : quando presente, identifica a notificação e poderá ser usado como parâmetro de notificações correlatas (*Correlated Notifications Parameter*) em futuras notificações.

Correlated notifications : quando presente, contém a identificação de outras notificações e, se necessário, os nomes dos objetos gerenciados associados.

State change definitions : quando presente, é usado para indicar a transição de estado associado com o alarme.

Monitored attributes : quando presente, define um ou mais atributos do objeto gerenciado e seus valores correspondentes na hora do alarme. Quem define os objetos gerenciados deve especificar quais os atributos de interesse, se houver.

Proposed repair actions : quando presente, é usado para situações onde a causa é conhecida e o sistema gerenciado pode sugerir uma ou mais soluções. Na recomendação [X.733], são sugeridos dois valores para este atributo: *no repair action required* e *repair action required.*

Additional text : quando presente, permite uma descrição em texto livre.

Additional information : quando presente, inclui um conjunto de informações adicionais, contendo um identificador de objeto, um indicador de significância (falso/verdadeiro) e informações a respeito do problema. Três parâmetros são incluídos

pela recomendação [Q.821]: *Log Record Id, Correlated Record Name* e *Suspect Object List.*

4.5 CONSIDERAÇÕES SOBRE A ESTRUTURA DOS ALARMES

Como pode ser visto neste capítulo, o formato padronizado dos alarmes é bastante completo e complexo. Segundo o padrão, diversas informações deveriam ser disponibilizadas quando um NE emitisse um alarme. Na prática, os recursos existentes atualmente nas redes de telecomunicações seguem modelos proprietários e, na maioria das vezes, fornecem apenas um subconjunto destas informações. O capítulo 5 apresenta um exemplo desta problemática.

A questão que se coloca é a dificuldade de desenvolvimento de aplicações que tratem alarmes provenientes de equipamentos de diferentes fornecedores, isto é, com estruturas de alarmes diversificadas.

Para amenizar este problema, este trabalho apresenta, no capítulo 6, uma proposta de migração do formato proprietário em que se encontram os alarmes emitidos pelas centrais AXE-10 (descritas no capítulo 5) para o formato padronizado.

5 CENTRAL AXE

Neste capítulo é feita a descrição da central AXE-10, que é a tecnologia utilizada para o desenvolvimento deste trabalho, esta escolha foi feita devido a abrangência de uso nas operadoras. Esse capítulo apresenta as partes que compões a central, os seus subsistemas e o formato dos relatórios de alarmes emitidos pela central.

AXE é uma central telefônica. A sigla é uma codificação utilizada pela Ericsson e todos os seus produtos são identificados através de um código de três letras. Essa sigla já era usada nas primeiras centrais da Ericsson. Mesmo com a mudança do hardware entre as primeiras centrais e as atuais, a sua semelhança está na estrutura interna que é independente da tecnologia utilizada.

5.1 ESTRUTURA DO SISTEMA AXE

O sistema AXE é controlado por programa armazenado (CPA) conforme ilustra a Figura 8. Todas as operações a serem executadas pela central são armazenadas na memória do computador dessa e a modificação de uma função ou controle é feita através da modificação do conteúdo da memória do computador.

O equipamento de comutação é controlado por dois computadores, sendo um deles o computador reserva, que assume o controle em caso de falha do principal.

O sistema AXE é dividido em duas partes principais: APT, que é o equipamento de comutação, e o APZ, que é a parte do computador. O APZ consiste do hardware do computador e o software necessário para operar os dispositivos de I/O, memória e o programa para administrar o trabalho executado pelo computador. O APT é formado pelo equipamento da central, placas de circuito impresso, linhas e os programa para o controle da central, que estão armazenados no APZ.

O trabalho executado pela central telefônica pode ser dividido em dois grupos. O primeiro é a tarefa de varredura dos equipamentos para detectar mudanças. O segundo grupo consiste em análises complexas e diagnósticos que requeiram alta capacidade de processamento e grande volume de dados.

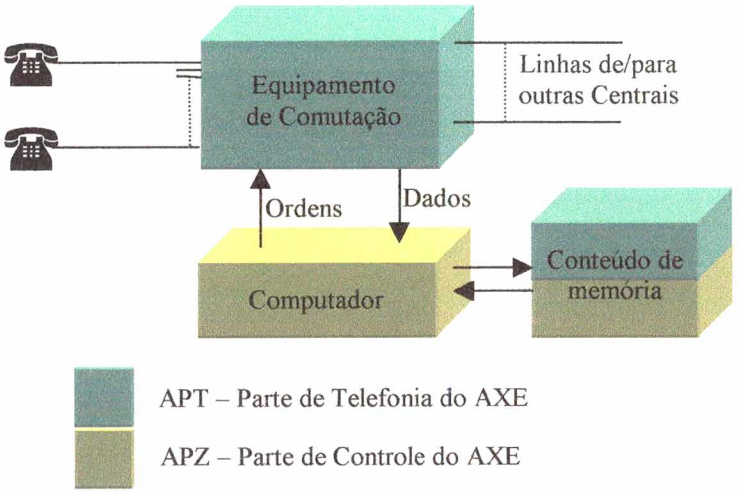


Figura 8 : Uma Central CPA

Como a execução destes dois grupos exige velocidade de atendimento, o sistema conta com dois tipos de processadores, um Processador Central (CP) e vários Processadores Regionais (RPs). Os RPs auxiliam o CP na execução de tarefas rotineiras e informam quando algum evento importante ocorre, mas as decisões são feitas pelo CP. Esse tipo de configuração facilita as modificações no sistema quanto a sua capacidade, através da modificação do número de RPs até o limite da capacidade do CP.

5.1.1 APT

O APT abrange a parte da telefonia no sistema AXE, que é dividido em vários subsistemas para facilitar o trabalho de todas as atividades relacionadas ao sistema AXE, sendo as divisões feitas de acordo com as funções.

Listas dos subsistemas do APT:

TCS *Subsistema de Controle de tráfego*

TSS *Subsistema de Troncos e Sinalização*

GSS *Subsistema de Seleção de Grupo*

OMS *Subsistema de Operação e Manutenção*

SSS *Subsistema de Comutação de Assinantes*

CHS *Subsistema de Tarifação*

SUS *Subsistema de Serviços de Assinantes*

OPS *Subsistema de Operadora*

CCS *Subsistema de Canal Comum*

MTS *Subsistema de Telefonia Móvel*

NMS *Subsistema de Gerenciamento de Rede*

A seguir é apresentado cada subsistema individualmente.

5.1.1.1 TCS - Subsistema de Controle de Tráfego

Consiste apenas de software e é a parte central do ponto de vista de encaminhamento de tráfego. Fazendo uma comparação, podemos dizer que o TCS substitui as operadoras dos sistemas manuais, incluindo as tarefas de controle das fases de estabelecimento e desconexão das chamadas telefônicas. A Figura 9 ilustra a interação do TCS com os outros subsistemas da central.

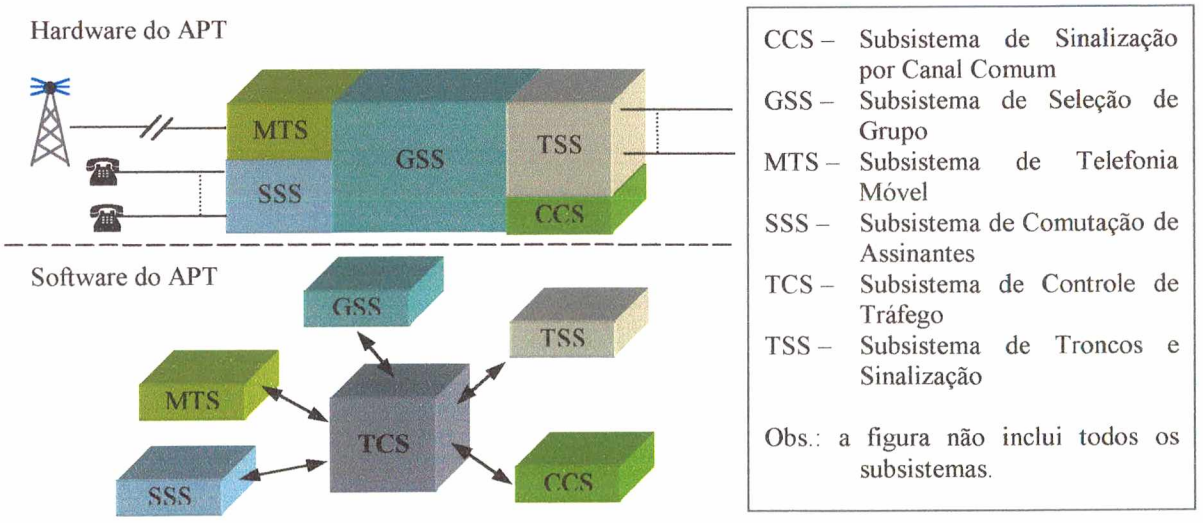


Figura 9 : interação do TCS com os outros subsistemas

5.1.1.2 TSS - Subsistema de Troncos e Sinalização

O TSS é constituído de hardware e software. Sua função é de supervisionar a sinalização nas linhas tronco, bem como a cooperação do sistema AXE com os vários sistemas de sinalização de registrador.

5.1.1.3 GSS Subsistema de seleção de grupo

Constituído de hardware e software, tem a tarefa de estabelecer, supervisionar e desligar as conexões através de módulos de comutação. Os módulos de comutação existentes dentro do GSS são o Módulo de Comutação Temporal (TSM) e o Módulo de Comutação Espacial (SPM).

Um comutador temporal é composto de:

- Uma memória de conversação para armazenamento temporário de amostras de voz codificadas. Cada canal no comutador temporal tem uma posição definida na memória de conversação.
- Uma memória de controle usada para controlar a leitura da memória de conversação.

A seqüência das amostras pode ser trocada após as suas leituras, pois a memória de controle irá ter a seqüência para direcionar cada amostra, como mostra a Figura 10.

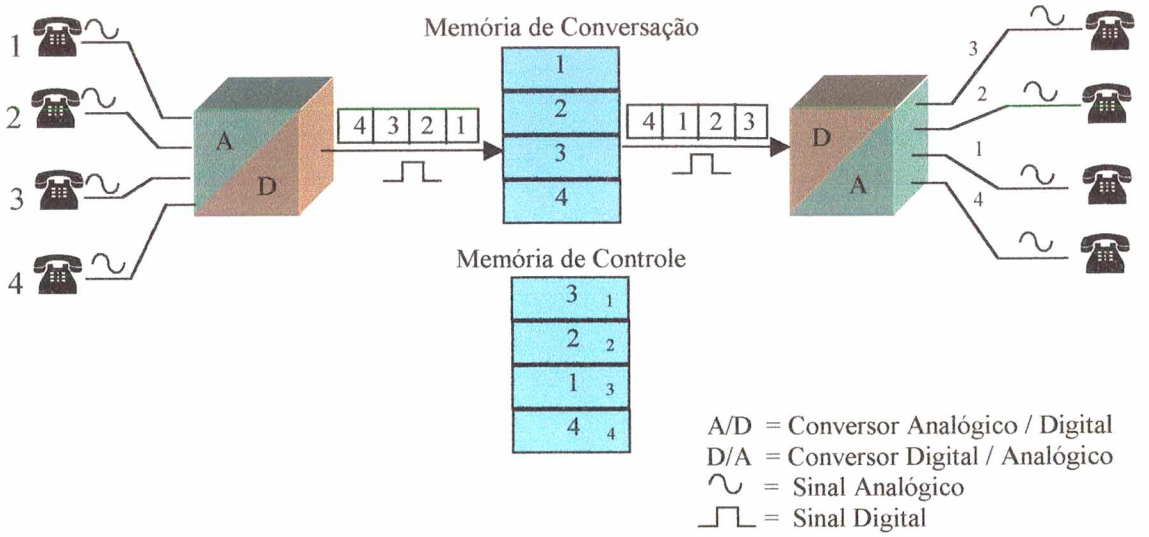


Figura 10 : Comutador Temporal simplificado

Teoricamente, poderia se ter um único comutador temporal, mas devido ao grande número de entradas que são necessárias, a freqüência das escritas e leituras estaria numa taxa em que os componentes de hoje não suportam, assim se fazem necessários vários módulos de comutação temporal (TSM). Para estabelecer a conexão de um módulo temporal para outro, é usado um modulo de comutação espacial (SPM).

A estrutura de um SPM é muito simples e pode ser apresentada como uma matriz simples com pontos de comutação, onde cada ponto de comutação representa uma porta lógica que se abre e fecha muito rapidamente.

No AXE, a capacidade de cada TSM é de 512 entradas e, no máximo, 32 TSMs podem ser conectados à um SPM.

Como o GSS é uma parte vital do sistema AXE, existem requisitos importantes quanto à sua confiabilidade funcional, pois, se um SPM fosse bloqueado, até 16.000 assinantes seriam desconectados. Assim, o AXE foi equipado com dois grupos completos de comutadores, um chamado de plano A e o outro de plano B. Uma amostra de voz é enviada sempre através de ambos os planos, mas no retorno para a linha de transmissão é utilizada a amostra de um dos planos, geralmente do plano A. Quando um deles apresentar falhas, esse será bloqueado e o equipamento correspondente ao outro plano assume o encaminhamento de tráfego, sem perturbar o tráfego que está sendo encaminhado.

5.1.1.4 OMS Subsistema de Operação e Manutenção

Constituído na sua totalidade de software, tem várias funções relacionadas à estatística e supervisão.

5.1.1.5 SSS Subsistema de Comutação de Assinantes

O SSS encaminha o tráfego entre os assinantes. O estágio de assinantes no AXE é digital, assim toda a comutação na central é digital. O sinal analógico vindo da linha de assinante é convertido para a forma digital. Esta tarefa é feita pelo Circuito de Interface de Linha do Assinante (LIC).

As funções básicas do estágio de assinantes são:

- a alimentação de linha de assinante;

- concentração do tráfego para o GSS;
- recepção de dígitos de telefone com disco (pulsos);
- recepção de dígitos de telefones com teclas (freqüências);
- emissão de corrente de toque para o assinante;
- emissão de diferentes tons para os assinantes;
- execução de medições na linha de assinante.

Algumas dessas funções são comuns a vários assinantes, outras são individuais. Todas as funções individuais são concentradas no LIC, que são:

- alimentação de corrente;
- inversão de polaridade;
- recepção de pulso de disco;
- relê para conexão equipamento de teste;
- conversão analógico-digital;
- relê para conexão da corrente de toque.

A Figura 11 ilustra um esquema simplificado de um SSS.

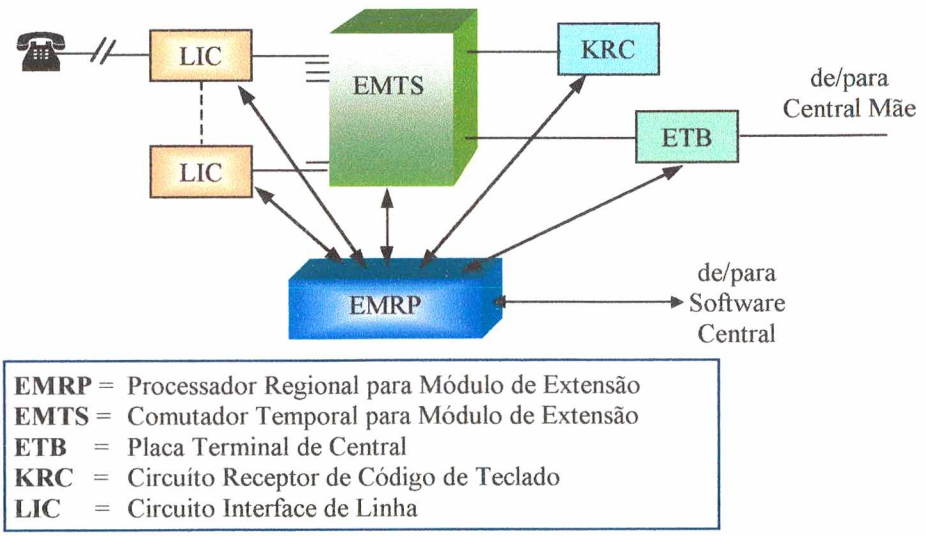


Figura 11 : Esquema simplificado do Subsistema de Comutação de Assinantes

O LIC não tem capacidade para recepção dos dígitos vindos dos telefones a teclado por freqüência. O equipamento para essa função é o Circuito de Recepção de Código de Teclado (KRC), sendo esse comum a vários assinantes. Para conectar o

KRC ao assinante chamador, é utilizado um Comutador Temporal para Módulo de Extensão (EMTS).

Para conectar os assinantes ao GSS é necessário um equipamento adicional, denominado Placa Terminal de Central (ETB), que contém 32 canais digitais.

Para coordenar as funções do SSS, foi projetado o Juntor Combinado, que também atua como interface com o TCS.

A um EMTS podem ser conectados 128 assinantes, 8 KRCs e um ETB com 32 canais. Esse conjunto de órgãos compõem um Módulo de Comutação de Linha (LSM).

O software regional para estágio de assinantes é memorizado e processado em um processador específico, chamado Processador Regional para Módulo de Extensão (EMRP).

O trabalho de varredura do hardware é feito por microprocessadores pequenos e simples, localizados em diferentes partes do hardware. Esse tipo de processador é denominado de Processador de Órgãos (DP) e são varridos por um EMRP. O programa no DP não tem funções de decisão, ele somente informa ao EMRP as mudanças que ocorrem no hardware.

5.1.1.6 CHS Subsistema de Tarifação

O CHS é responsável pela análise dos dados de tarifação, identificação da central que deve tarifar a chamada, o método a ser usado na tarifação e administrar o avanço dos contadores de assinantes.

5.1.1.7 SUS Subsistema de Serviços de Assinantes

É constituído de software, sendo que nesse subsistema são implementadas as facilidades de assinantes, tais como discagem abreviada, transferência de chamada, linha direta, e outros serviços que sejam disponibilizados para o assinante.

5.1.1.8 OPS Subsistema de Operadora

Constituído apenas de software, tem a função fazer a conexão e desconexão da telefonista.

5.1.1.9 CCS Subsistema de Canal Comum

Constituído de hardware e software, tem as funções para sinalização, roteamento, supervisão e correção de mensagens de acordo com o CCITT Nº 6 e Nº 7 [Ericsson 87].

O CCS utiliza terminais de sinalização (ST), que são usados para a sinalização de acordo com o CCITT Nr. 7. Os STs são conectados aos GSS através de um PCD-D, que serve como órgão de adaptação e conexão com o GSS. A informação que chega de um ST é enviada através do GSS para um canal de um ECT. Esse canal é usado exclusivamente para a sinalização. A vantagem de conectar STs através do GSS é que alguns órgãos podem ser colocados como reservas e substituirem, automaticamente, órgãos com defeito. A Figura 12 ilustra as conexões descritas acima.

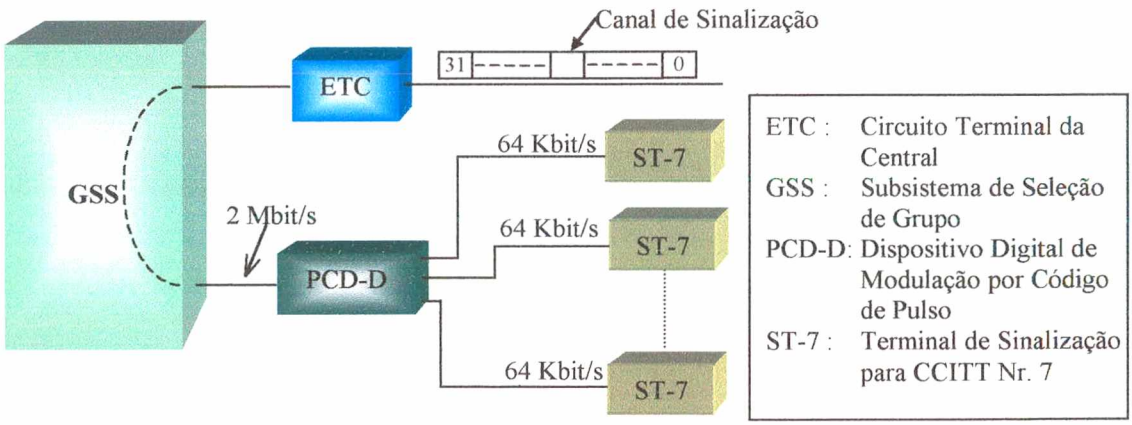


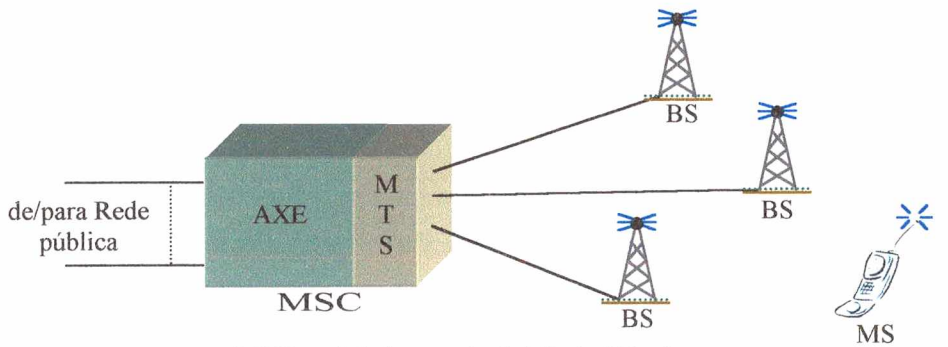
Figura 12 : Terminais de Sinalização

O CCITT Nr.6 é um sistema de sinalização usado para conexões internacionais. O princípio básico é o mesmo do CCITT Nr.7, mas o sistema é projetado para ser utilizado nas conexões com enlaces analógicos, com uma taxa de transmissão de 2400 bits/s, que é uma taxa baixa comparada com as taxas de transmissão usadas no CCITT Nr.7, que trabalha com uma taxa de 56 ou 64 kbit/s.

5.1.1.10 MTS - Subsistema de Telefonia Móvel

O objetivo do MTS é disponibilizar os serviços de telefonia móvel para a central AXE. Uma central AXE, que contenha o MTS, é chamada de Centro de Comutação de Serviço Móvel (MSC), que geralmente é instalado separadamente, mas pode fazer parte de uma central de trânsito nacional.

Um MSC é conectado a algumas estações de base (BS), cada uma servindo a uma área definida, sendo que seu tamanho varia conforme as condições geográficas e as freqüências utilizadas. Essa área é chamada de Área de Serviço e o equipamento de um assinante móvel é chamado de Estação Móvel (MS), como mostra a Figura 13.



MTS : Subsistema de Telefonia Móvel
MSC : Centro de Comutação de Serviço Móvel
BS : Estação de Base
MS : Estação Móvel

Figura 13 : Esquema simplificado de uma rede de telefonia móvel

5.1.1.11 NMS - Subsistema de Gerenciamento de Rede

Esse subsistema supervisiona e executa modificações temporárias no tráfego, podendo ser utilizado em qualquer nível da rede (central internacional, central de trânsito nacional e central local).

O principal objetivo das funções do NMS é detectar sobrecarga da rede e limitar suas conseqüências. A sobrecarga pode ser causada por problemas técnicos ou aumento drástico no tráfego em uma área definida. Uma vez detectada a sobrecarga, o NMS pode tomar uma decisão preventiva, limitando um certo percentual ou tipo de tráfego na área sobrecarregada. A outra decisão é a ação expansiva, re-roteando o tráfego para áreas cuja capacidade não esteja em situação crítica.

5.1.2 APZ

O APZ abrange toda a parte do computador dentro da central. A capacidade de um processador para controlar centrais telefônicas é expressa em uma unidade denominada BHCA - Tentativas de Chamada na Hora de Maior Movimento (*Busy Hour Call Attempts*). O APZ é composto por vários subsistemas que são apresentados a seguir.

5.1.2.1 Subsistemas no APZ

CPS – Subsistema de Processamento Central

Composto de hardware e software, tem as funções de administração dos trabalhos de processamento, carga, mudanças de programas e manuseio de memória.

MAS – Subsistema de Manutenção

É composto de software e hardware. Sua tarefa é de localizar falhas de hardware e software e procurar minimizar os efeitos das falhas.

RPS – Subsistemas de Processamento Regional

Subsistema que abrange todos os processadores regionais (RPs). O hardware é composto pelos processadores regionais e o software são os programas localizados nestes processadores.

MCS – Subsistema de Comunicação Homem-Máquina

Administra a comunicação entre os órgãos E/S e o restante do sistema. Os órgãos de E/S podem ser monitores, impressoras, painéis de alarmes, microcomputadores que usem sistema de menus em vez de comandos AXE.

SPS – Subsistema de Processamento de Suporte

É composto por um processador de grande capacidade para a comunicação com todos os órgãos de E/S. Também administra as funções de bloqueio, desbloqueio e supervisão de órgãos de E/S.

DCS – Subsistema de Comunicação de Dados

Administra a comunicação entre os blocos do CP e do SP. A estrutura segue os padrões definidos pela ISO para sistemas de E/S. O DCS também administra a comunicação sobre enlaces de dados de acordo com os protocolos do CCITT para dados (x.25, x.75 e x.28) [Ericsson 87].

FMS – Subsistema de Gerência de Arquivos

Administra todos os arquivos usados no sistema. Os blocos de dados do sistema devem sempre consultar o FMS antes de armazenar informações em meio de armazenamento externo.

5.1.2.2 Funções do APZ

A duplicação do hardware é utilizada para minimizar os efeitos das falhas. São utilizados dois processadores idênticos, cada um contendo suas memórias próprias. Os dois processadores são chamados CP-A e CP-B. Os dois lados do CP executam os mesmos programas, instrução por instrução, e ambos os lados são comparados permanentemente a fim de detectarem as falhas imediatamente. Não há redução na capacidade de processamento se um dos dois parar.

Para diagnosticar uma falha no CP, o APZ compara continuamente os dois lados do CP. Se houver alguma diferença entre os dois lados, indicando uma falha, será verificado qual lado apresenta a falha. Isso é feito por um programa de aproximadamente 20 ms que testará cada um dos lados. O lado sem falhas continua a processar e é colocado em EXECUTIVE, e o outro lado é bloqueado. Após o bloqueio, é feito a tentativa de atualização do lado que apresentou a falha. Esse procedimento serve para verificar se a falha permanece e localizar a unidade que apresentou o problema. Se a falha não for temporária, será emitido um alarme após a fase de atualização.

Após o reparo do CP que apresentou uma falha, ele deve retornar a operar em paralelo com o lado em funcionamento. Para isso são transferidos os dados do lado em funcionamento para o lado recém reparado. Isso é chamado de atualização e deve assegurar que ambos os lados tenham os mesmos dados e os mesmos programas. Ao término da atualização, o lado A será o EXECUTIVE e o lado B, o STANDBY. Como

ambos os lados receberão as mesmas informações dos processadores regionais, eles conterão os mesmos dados e portanto executarão exatamente os mesmos trabalhos.

Os CPs são auxiliados pelos RPs, Processadores Regionais. A comunicação entre o CP e os RPs é feita através do Barramento de Processamento Regional (RPB). Por questões de confiabilidade, todos os RPs são duplicados. Dois RPs operam em divisão de carga, cada um controla metade do equipamento. Caso ocorra uma falha em um dos RP's, o outro assume o controle sem perda de performance.

Os equipamentos controlados pelos RPs são agrupados em módulos denominados de Módulos de Extensão (EM). Cada RP controla em média de 8 a 16 EMs. A comunicação entre os RPs e os EMs é feita através do Barramento de Módulo de Extensão (EMB). A Figura 14 ilustra as conexões entre os CPs, RPs e EMs.

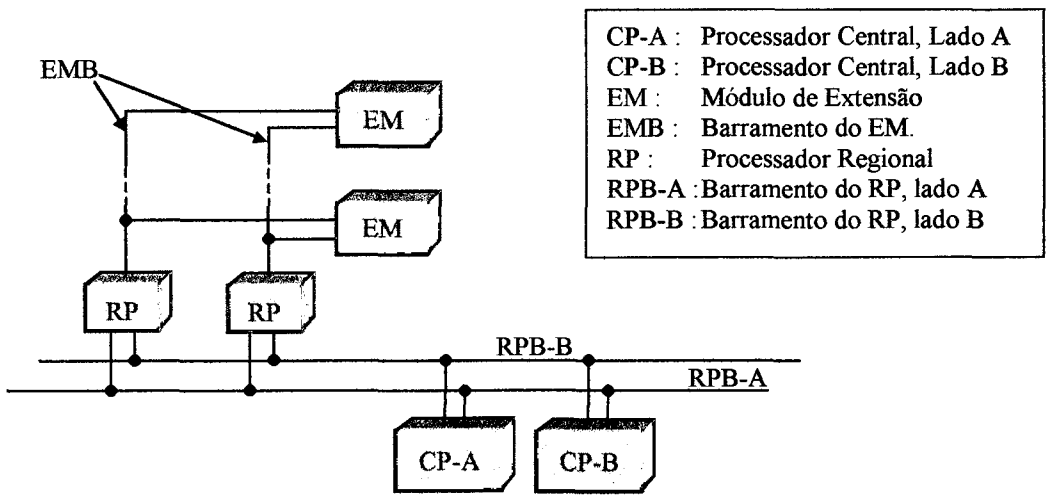


Figura 14 : Comunicação entre CP - RP – EM

Um RP é composto de 5 placas de circuito impresso. Uma para alimentação, duas para comunicação com os CPs, lado A e B. As outras duas é o próprio processador. Uma das placas do RP, denominada de MEU, é equipada com uma memória e com circuitos para comunicação com o EMB. A outra placa, denominada de PRO, contém microprogramas e circuitos para cálculos de endereço.

5.2 ALARMES DE FALHAS

O sistema AXE foi projetado para ter um alto grau de detecção de falhas, utilizando processos de supervisão para a detecção das mesmas e defeitos na central, em tempo para que medidas sejam tomadas a fim de que não ocorram distúrbios mais sérios no tráfego.

Através da emissão dos alarmes, é possível identificar a unidade afetada. A unidade é indicada na listagem por meio de seu código funcional, também usado como rótulo de hardware na central. Esse mesmo código é utilizado pelos operadores nos comandos de diagnósticos nas respectivas unidades.

As listagens de alarmes podem ser iniciadas tanto por falhas ocorridas no sistema, quanto por níveis de supervisão, previamente estabelecidos, que tenham sido excedidos.

A Figura 15 apresenta um exemplo de um alarme emitido pela central, e a seguir a explicação das partes que compõe o alarme.

```
WO      103/211/02/8/5A1 CCCROI   AT-0    TIME 991027 1740  PAGE    1

*** ALARM 294 A2/APT "103/211/02/8/5A"A 981027 1740
MOBILE TELEPHONY CELL SERVICE SUPERVISION

CELL      DEVTYPE  NBD  NCD
BAG2      CC         1    2
END
```

Figura 15 : Alarme da Central AXE-10

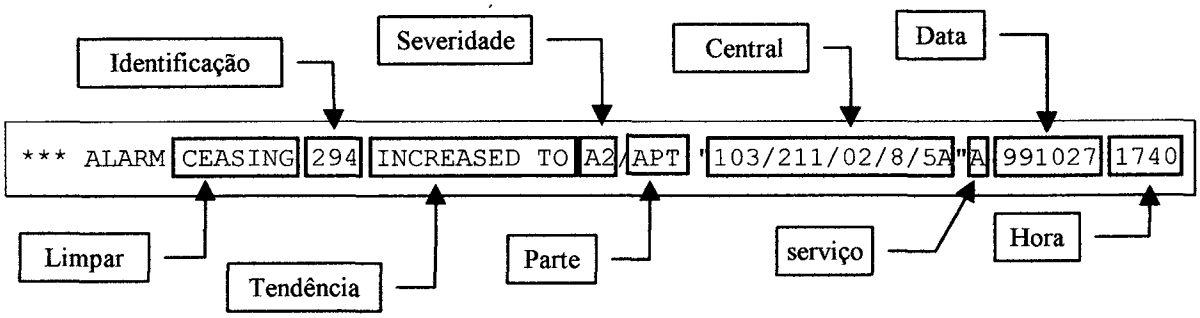
5.2.1 Linha de Salto de página

```
WO      103/211/02/8/5A1 CCCROI   AT-0    TIME 991027 1740  PAGE    1
```

Essa linha é apresentada no início de cada página e pode ser inserida entre as linhas de um alarme. Para fins de alarme, tal linha não tem utilidade, pois é apenas um componente estético no relatório da central.

5.2.2 Linha Inicial

A primeira linha dos alarmes da central AXE10, apresentada abaixo, contem algumas informações úteis que são detalhadas abaixo :



→ Limpar

A palavra *CEASING* só esta presente no relatório de alarme se esse indicar que o alarme com o mesmo número de identificação foi limpo.

→ Identificação

Número de identificação do alarme. Um relatório utiliza o número de alarme já existente caso este indique que o mesmo é um relatório de alarme de mudança de severidade de um relatório emitido anteriormente (ver nos itens **Limpar** e **Tendência**).

→ Tendência

Esta informação indica que houve uma mudança de severidade no alarme de mesmo número de identificação. Esta mudança de severidade pode ser para maior (*INCREASED TO*) ou para menor (*DECREASED TO*)

→ Severidade

Indica a severidade do alarme. Os relatórios de alarme da central AXE10 podem conter uma das cinco severidades abaixo:

A1 - Atendimento imediato, a qualquer hora do dia.

A2 - Atendimento, logo que possível, durante o horário normal de trabalho.

A3 - Atendimento em até uma semana.

O1 - Examinar, logo que possível, no horário normal de trabalho.

O2 - Examinar em momento apropriado.

→ Parte

Indica a parte a qual o alarme se refere. Um relatório de alarme pode indicar uma das partes descritas abaixo:

APT : parte de comutação da central.

APZ : parte do computador da central.

EXT : parte externa da central, como fonte alimentação ou ambiente.

→ Central

Identificação da central que emitiu o alarme.

→ Serviço

Indica se a central está em serviço (A) ou fora de serviço (U).

→ Data

Data da emissão do alarme.

→ Hora

Hora da emissão do alarme.

5.2.3 Identificação do Alarme

```
MOBILE TELEPHONY CELL SERVICE SUPERVISION
```

Essa linha contém o texto que identifica o problema que o alarme está indicando.

5.2.4 Informações complementares do alarme

```
CELL      DEVTYPE  NBD  NCD
BAG2      CC         1    2
```

No caso de um alarme de equipamento, essas linhas identificam o dispositivo afetado. Com a identificação do alarme e a identificação do dispositivo, o técnico tem condições de localizar na central onde ocorreu o problema indicado pelo alarme.

Em outros tipos de alarme, nessas linhas são inseridas informações para detalhar o alarme.

5.2.5 Linha de Finalização

```
END
```

Essa linha indica que o relatório do alarme está concluído. Todo o alarme da central AXE10 é finalizado pela linha END.

Através do estudo do formato do alarme emitido pela central, verifica-se as limitações das informações contidas em um alarme em relação aos parâmetros exigidos pela recomendação [X.733] (item 4.4). Essa diferença faz com que o desenvolvimento de um sistema de adaptação dos alarmes emitidos pela central seja complexa, consumindo recursos consideráveis no desenvolvimento de um projeto de supervisão de alarmes. O sistema desenvolvido nesse trabalho, capítulo 6, já executa a parte de adaptação do alarme da central AXE-10 à recomendação [X.733]. Com isso a etapa de adaptação não necessita ser implementada em um novo projeto.

6 DESENVOLVIMENTO

Este capítulo apresenta a estrutura de um sistema desenvolvido para adequar os alarmes emitidos pelas centrais telefônicas AXE-10 da Ericsson, aos padrões da TMN conforme definido na recomendação [X.733]. Esse sistema recebe os alarmes da central, faz a adaptação dos dados do alarme às informações adicionais especificadas pelos parâmetros de informações definidos na recomendação [X.733] e emite o relatório de alarme padronizado.

Para adaptar as informações vindas da central, fez-se necessário a implementação de um banco de dados, pois muitas informações são emitidas de forma implícita, não atendendo desta forma a recomendação [X.733]. Sendo assim, o banco de dados foi construído previamente para fazer uma associação dos dados do alarme com os parâmetros da recomendação utilizada.

Porém, devido à limitações das informações contidas no relatório dos alarmes, quatro parâmetros não puderam ser atendidos. São eles: Informações de Threshold, Notificações Correlatas, Definição de Mudança de Estado e Atributos Monitorados, os quais serão descritos posteriormente no decorrer deste capítulo.

Antes do início do desenvolvimento do sistema, se fez necessário o estudo sobre redes de telecomunicações, descrito no capítulo 2; da TMN, descrito no capítulo 3; Supervisão de alarmes, descrito no capítulo 4; e da Central AXE10, descrito no capítulo 5.

O sistema foi desenvolvido na linguagem C++, utilizando a programação orientada a objetos. Além das classes desenvolvidas, o sistema contém um banco de dados no qual são armazenadas informações para auxiliar o programa na decodificação de um alarme, adaptação de informações ao padrão da recomendação e informações auxiliares para poder atender a alguns parâmetros da recomendação. A Figura 16 apresenta um esquema genérico do sistema desenvolvido.

O sistema faz a leitura do alarme emitido pela central e, através de máscaras e formatos definidos no banco de dados, são extraídas as informações existentes no relatório de alarme. Com essas informações e tabelas com informações complementares, é feito o levantamento dos demais parâmetros do alarme. Após essa fase, o sistema emite o alarme contendo os parâmetros que foram possíveis de identificar no alarme emitido pela central.

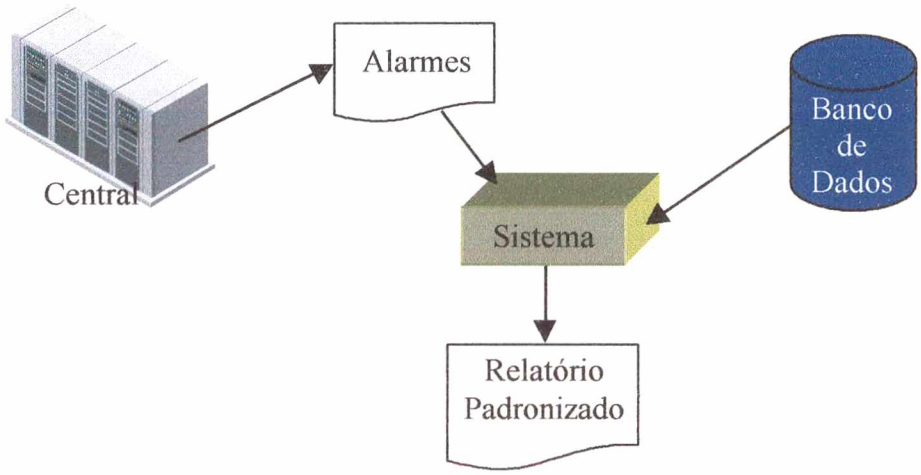


Figura 16 : Diagrama do Sistema

A Figura 17 ilustra, através de uma linha de alarme da central, como são definidas as máscaras e formatos para que o sistema possa obter os dados necessários. A linha utilizada para o exemplo é a primeira linha do alarme.

A linha máscara, em azul, indica as variáveis que existem na linha de alarme, cada variável irá receber um valor que está no alarme. No exemplo, estes valores são indicados através das setas em verde, que indicam o valor que será armazenado em cada variável. A exceção é a variável **limpar** que ficará sem valor (vazio), pois esta variável apenas recebe o valor *CEASING* no caso de alarme de limpeza.

As variáveis entre colchetes indicam que são variáveis opcionais. Essas variáveis são necessárias pois nem sempre haverá um valor correspondente a elas no alarme. Por exemplo, na Figura 17, após a aplicação da máscara a variável **limpar** está vazia, uma vez que o valor a ser armazenado nela não aparece nesse alarme.

As máscaras são armazenadas em uma tabela com nome de MASCARA, onde cada registro contém a máscara de uma linha do alarme.

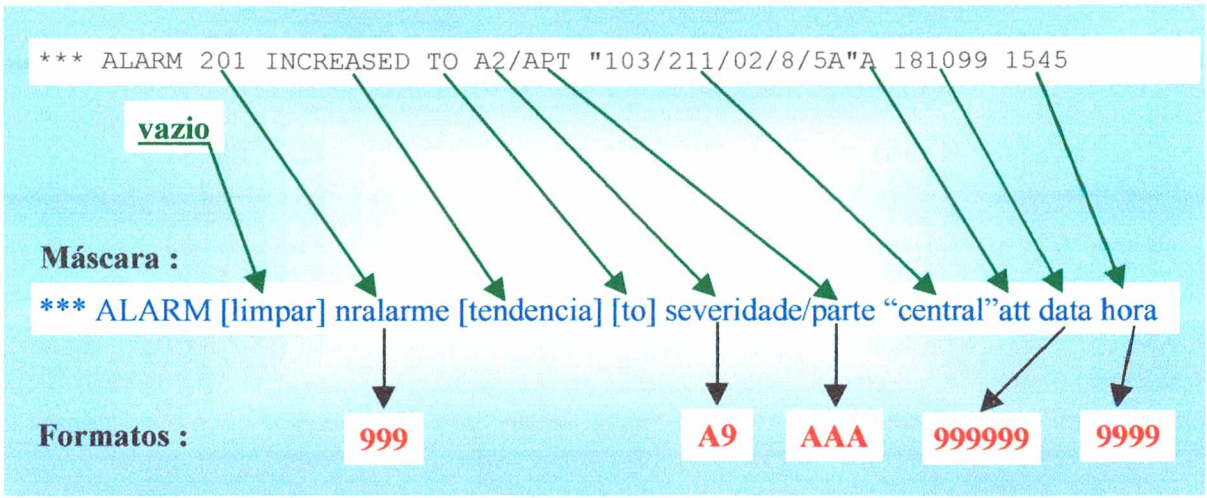


Figura 17 : Associação do conteúdo do Relatório de Alarme às variáveis e seus respectivos formatos.

Além das máscaras, o sistema utiliza formatos para extrair os valores com maior eficiência, esses formatos são apresentados, no exemplo, em letras de cor vermelha. Estes formatos permitem informar ao sistema o formato do dado a ser armazenado em uma variável. No formato são utilizados apenas três símbolos, o "A" que indica uma letra, o "9" que indica um número e o "X" que indica qualquer caractere. Os formatos não são obrigatórios e só deve ser informado no caso em que o dado a ser armazenado em uma variável segue um padrão que possa ser formatado.

Esses formatos são armazenados na tabela PICTURE (Anexo B, item 3) onde cada registro armazena o formato de uma variável.

6.1 Os Objetos do Sistema.

O sistema é composto por um conjunto de objetos que fazem o processamento de um alarme emitido. A Figura 18 e a Figura 19 mostram os diagramas da hierarquia de contenção dos objetos do sistema. No anexo A apresenta-se a descrição de todas as classes utilizadas no sistema e no anexo C estão os fontes dessas classes.

O objeto *central* é o objeto que representa uma central telefônica. Este objeto permite que se identifique: o modelo da central que está representando; o arquivo de

entrada onde é feita a leitura dos alarmes; o arquivo de saída, onde são armazenados os relatórios de alarmes após a padronização; identificação do usuário e senha para acesso ao banco de dados no Oracle.

O objeto *alarme*, que está contido no objeto *central*, faz a leitura e atribuição dos dados obtidos do alarme às variáveis. Através dos objetos *mascara*, *picture* e *linhas_nulas* é feita a identificação dos dados que estão no relatório de alarme e devem ser armazenados nas variáveis.

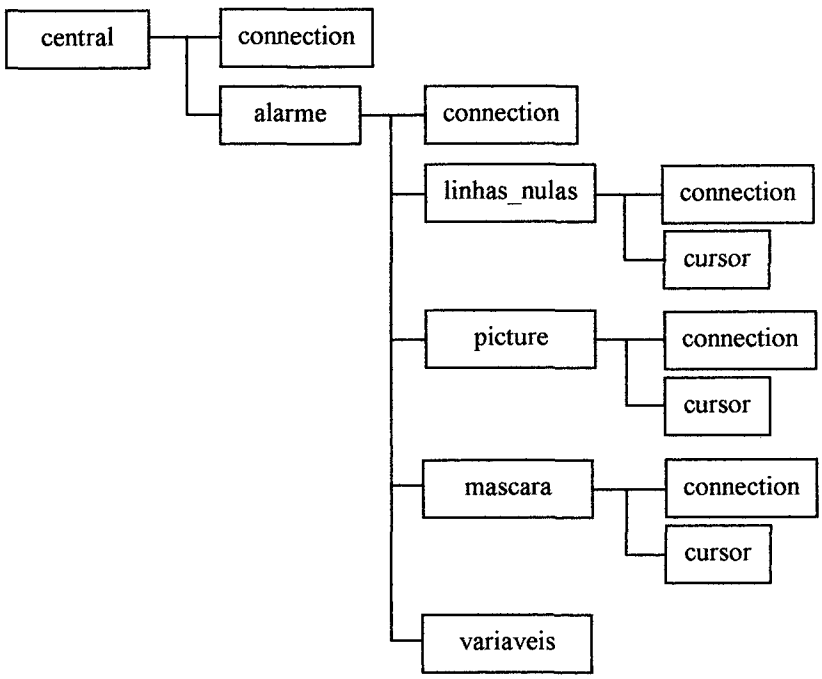


Figura 18 : Diagrama da Hierarquia de Contenção dos objetos do sistema.

O objeto *variaveis*, constituído por vários objetos (ver Figura 19), armazena as variáveis. Ao concluir a leitura de um alarme esse objeto complementa os dados da recomendação utilizando-se de vários objetos (ver figura 18). Em seguida, faz a emissão do relatório de alarme conforme o formato padronizado e por fim limpa as variáveis para um novo relatório de alarme.

O objeto *connection* faz a conexão entre o sistema e o Oracle. O objeto *cursor* é uma área de dados para o envio de comandos SQL e a recepção dos resultados desses

comandos. Ambos os objetos, *connection* e *cursor*, foram desenvolvidos pela Oracle e distribuídos junto com Sistema de Gerenciamento de Bases de Dados Relacionais (RDBMS) da Oracle para permitir a comunicação entre a linguagem de programação C++ e o RDBMS.

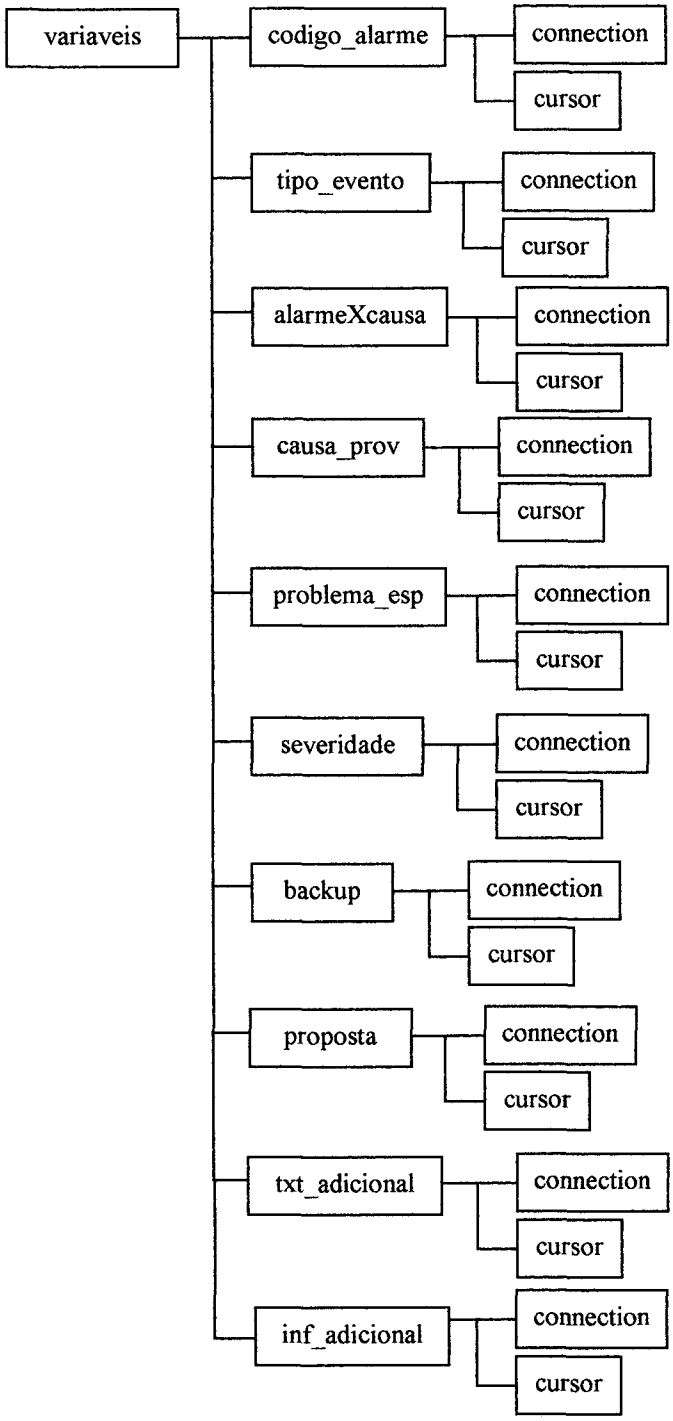


Figura 19 : Diagrama da Hierarquia de Contenção do objeto ***variaveis.***

6.2 AS TABELAS DO SISTEMA

O sistema conta com um conjunto de tabelas, descritas no anexo B, que estão armazenadas no RDBMS da Oracle. Essas tabelas auxiliam o sistema na padronização dos alarmes emitidos pela central. A escolha de um banco de dados se deve a facilidade de alteração de características do sistema sem a necessidade de alteração do código fonte. Isso permite que o usuário possa adaptar o sistema a uma outra central, através da inclusão de máscaras e formatos adequados ao novo modelo de central desejado e também complementando os dados nas tabelas referentes aos parâmetros do relatório de alarme definido na recomendação [X.733].

Desta forma, antes de colocar o sistema em funcionamento precisou-se cadastrar no banco de dados todas as informações que estão subentendidas de forma clara, para que fosse possível atender às recomendações.

6.3 INFORMAÇÕES ABRANGIDAS PELO SISTEMA.

Nesse tópico, são descritas as informações envolvidas num relatório de alarme. Juntamente, é indicado como essas informações são obtidas ou adaptadas e as que não são abordadas pelo sistema.

6.3.1 Tipo de Evento:

Essa informação é obtida na tabela ALARMES, que está descrita no Anexo B item 1. Nesta tabela encontra-se o código do tipo do evento (COD_TIPO). Através deste código é localizada na tabela TIPO_EVENTO (Anexo B item 5) a sua descrição.

6.3.2 Informações do Evento:

A seguir são relacionados os parâmetros da notificação de alarme.

6.3.2.1 Causa Provável :

Essa informação não consta no alarme da central em estudo, sendo necessário que seja definido para cada alarme da central a sua respectiva causa provável.

A causa provável é obtida através da tabela ALARME_CAUSA (Anexo B item 6), que é uma associação das tabelas ALARMES (Anexo B item 1) e CAUSAPROV (Anexo B item 7).

6.3.2.2 Problemas específicos :

Esse parâmetro, quando presente, permite apresentar informações complementares para detalhar a Causa Provável. No sistema, tal parâmetro foi implementado através da tabela PROB_ESP (Anexo B item 8), que contém o código do alarme e a descrição do problema específico.

6.3.2.3 Severidade :

A severidade informada no alarme segue uma codificação própria da central AXE10, e essa é convertida para um dos seis níveis estabelecidos pela recomendação [X.733] (*cleared, indeterminate, major, minor, warning*) com o uso da tabela SEVERIDADE (Anexo B item 9) onde, a cada nível de severidade da central é associado a um nível de severidade correspondente à recomendação.

6.3.2.4 Backed-up status :

Parâmetro que especifica se o objeto que emitiu o alarme foi passado para back-up. Caso o valor do parâmetro seja verdadeiro (*true*), deve ser especificada a instância do objeto de back-up, e, no caso de valor falso (*false*), não há objeto de back-up. Esse parâmetro é fornecido pelo objeto do sistema *backup*, utilizando a tabela BACKUP (Anexo B item 10).

6.3.2.5 Back-up object :

Especifica a instância do objeto gerenciado que provê os serviços de back-up. Essa informação só é necessária no caso do parâmetro *Backed-up status* ser verdadeiro

(*true*). Tal parâmetro é fornecido pelo objeto do sistema *backup*, utilizando a tabela BACKUP (Anexo B item 10).

6.3.2.6 Indicação de Tendência :

Indica a tendência de severidade de um alarme ainda não limpo (*cleared*) no mesmo objeto gerenciado. Essa informação é apresentada no alarme como *INCREASED*, caso o alarme indique que houve um incremento de severidade, ou *DECREASED*, no caso do alarme indicar que houve um decremento na severidade do alarme.

6.3.2.7 informações de Threshold :

Essa informação está presente apenas no caso do alarme indicar que um limite pré-estabelecido foi excedido. Esse parâmetro é composto de 4 sub-parâmetros, os quais não são possíveis de serem obtidos através das informações contidas num alarme emitido pela central AXE-10. Dessa forma, as informações complementares que variam de alarme para alarme, são inseridas no parâmetro de texto adicional. A informações complementares são apresentadas no capítulo 5, no item 5.2.4, Informações complementares do alarme

6.3.2.8 Identificador da Notificação :

Identifica a notificação. No caso da central, AXE-10, cada alarme emitido tem um identificador numérico.

Esse parâmetro é fornecido diretamente no alarme da central, porém a central faz o reuso desses identificadores quando os mesmos forem liberados (cleared). Dessa forma, foram adicionadas a data e a hora de ocorrência do alarme ao identificador numérico informado pela central, gerando assim um identificador único de alarme.

6.3.2.9 Notificações Correlatas :

Esse parâmetro não foi implementado no sistema, pois a central não fornece informações que permitam a identificação de notificações correlatas.

6.3.2.10 Definição de Mudança de Estado :

Este parâmetro não se encontra nos alarmes da central AXE-10. Quando presente, indica a transição de estado do alarme. Contudo, para saber se houve transição é preciso obter os históricos passados. Como o trabalho proposto não tem por finalidade gerar logs dos alarmes, o mesmo não será implementado.

6.3.2.11 Atributos Monitorados

Esse parâmetro define um ou mais atributos do objeto gerenciado e seus valores no momento do alarme. Por se referir a componentes de um objeto, tal parâmetro encontra-se fora do escopo do trabalho.

6.3.2.12 Proposta de ações de reparo :

Esse parâmetro existe caso o sistema possa sugerir uma proposta de ação de reparo para o problema detectado. Tal parâmetro foi implementado na tabela PROPOSTA.

6.3.2.13 Texto Adicional :

Permite uma descrição de formato livre que será reportado juntamente com o alarme. Para esse parâmetro foi criada a tabela TXTADICIONAL.

6.3.2.14 Informações Adicionais :

Esse parâmetro permite a inclusão de informações adicionais no relatório de evento de alarme.

No sistema, o mesmo será utilizado para a inserção de informações apresentadas no alarme emitido pela central e que não se enquadram em nenhum dos parâmetros anteriores.

7 CONCLUSÃO

As informações contidas nos relatórios de alarmes, emitidos pelos equipamentos de uma rede de telecomunicações, devem permitir a identificação do problema e identificar a possível causa, de forma que a gerência possa tomar decisões para que os mesmos possam ser eliminados ou minimizados antes que sejam percebidos pelo usuário. Para atingir esses objetivos, não basta ter um grande número de informações. Essas devem disponibilizar todos os dados de forma padronizada à tomada de decisões.

Nesse trabalho, pode-se observar não só a importância das informações como também a forma como essas são fornecidas, pois, para o sistema de gerência integrada, as informações devem seguir um padrão. Neste trabalho o padrão adotado foi o definido pelo ITU-T através da recomendação [X.733]. Caso o equipamento não utilize esse padrão, os dados devem ser ajustados conforme as exigências do mesmo.

O sistema desenvolvido permitiu a adaptação dos relatórios de alarmes emitidos pela central AXE-10 ao padrão definido na recomendação [X.733] do ITU-T, com a utilização de um programa que acessa e um banco de dados desenvolvido com a finalidade de produzir os relatórios padronizados. Essa opção é viável, pois apresenta um custo reduzido quando comparado à troca do parque instalado para que fosse possível padronizar tais relatórios.

O conceito da TMN surgiu no ano de 1985 através do Study Group IV do CCITT quando a telefonia já tinha mais de 100 anos. As instalações existentes já eram em quantidades demasiadas e de custo elevado quando surgiu esse conceito. Dessa forma não se pode esperar que as operadoras trocassem os equipamentos existentes, devido aos custos de sua troca e também por exigir mudanças nos sistemas já instalados. A implementação de sistemas de adaptação se mostrou neste trabalho uma solução mais viável, pois o mesmo faz apenas a adaptação dos atuais sistemas.

Perspectivas Futuras

O sistema desenvolvido conta apenas com a adaptação dos alarmes, sendo que a interface com o usuário ainda precisa ser desenvolvida para permitir a inclusão de dados de forma mais amigável no banco de dados utilizado no sistema.

O sistema abrange apenas a área de suporte à gerência de alarmes, sendo que na TMN ainda existem outras quatro áreas funcionais, as quais também podem ser estudadas a fim de se desenvolver um sistema que permita a padronização dos dados nessas áreas também.

O trabalho apresentado poderá ser evoluído também para mapear os alarmes de ou para modelos emergentes, como por exemplo do Open-CORBA atuando como agente ou para o Open-CORBA atuando como gerente respectivamente. O Open-CORBA é um modelo de informações cliente/servidor desenvolvido pelo TMF Open CORBA Forum [TMF] onde participam diversas empresas do segmento de telecomunicações, como por exemplo: Nortel, Siemens, Lucent, Alcatel, Tellabs e Vertel entre outras. O Open-CORBA é um modelo de informações em UML e especificado em CORBA IDL [OMG] para ser usado como modelo multi-vendor para gerência de redes SONET/SDH. Uma versão pública é apresentada no documento NMF 509 v1.1 [TMF 509]. Este modelo já vem sendo implementado por empresas como Siemens e Lucent que apresentaram na CeBit 2000 [CeBit 2000]. O modelo está adquirindo estabilidade e continua sendo expandido para novas tecnologias como DWDM [TMF 509].

Aproveitando o estudo realizado sobre a central AXE-10, uma outra proposta é a de adaptar o sistema ao padrão SNMP, convertendo os relatórios de alarme da central em traps.

REFERÊNCIAS

[Aidarous 98] Aidarous, S., e Plevyak, T., *Telecommunications Network Management - Technologies and implementations.* IEEE Press, Piscataway-NJ, Jan 1998

[BRISA 92] BRISA, *Gerenciamento de Redes - Uma abordagem de sistemas abertos.* São Paulo, Makron Books do Brasil Editora Ltda, 1992

[Carne 95] Carne, E. Bryan, *Telecommunication primer: signals, building blocks, and networks.* Prentice-Hall, Inc. - New Jersey, 1995

[CeBit 2000] CeBit 2000 de 24/02 a 01/03/2000 Hannover - Germany

[Ericsson 87] *Introdução ao AXE, Manual de Treinamento,* Ericsson, 1987

[G.711] ITU-T Recommendation G.711, *General Aspects of Digital Transmission Systems, Pulse Code Modulation (PCM) of Voice Frequencies,* ITU-T, 1993

[M.3010] ITU-T Recommendation M.3010, *Principles for a telecommunication management network,* ITU-T, Helsinki, May 1996

[M.3100] ITU-T Recommendation M.3100, *Generic network information model,* ITU-T, Helsinki, Jul. 1995

[M.3400] ITU-T Recommendation M.3400, *Generic network information model – Function Definition,* ITU-T, Helsinki, Apr. 1997

[Minoli 95] Minoli, Daniel, *Telecommunication technology handbook.* Artech House, Inc. - Norwood , 1995

[OMG] página da internet da OMG (www.omg.org)

[Pras 99] Pras, A.; Beijnum, B. e Sprenkels, R., *Introduction to TMN, CTIT Technical Report,* University of Twente, The Netherlands, Apr 1999

[Q.821] ITU-T Recommendation Q.821, *Stage 2 and stage 3 description for the Q3 interface-Alarm surveillance,* ITU-T, Helsinki, Mar 1993

[TMF 509] TeleManagement Forum – *NML-EML Interface Business Agreement For Management of SONET/SDH Transport Networks,* Public Evaluation Version 1.1 - Morristown, NJ 07960 USA - October 2000

[TMF] Página da internet do TMF (www.tmforum.org)

[X.701] CCITT Recomendation X.701 – *Information Tchnology – Open Systems Interconnection – System Management Overview,* CCITT, Geneva, Jan 1992.

[X.710] CCITT Recommendation X.710, *Common management information service definition for CCITT applications,* CCITT, Geneva, Jan 1992

[X.711] CCITT Recommendation X.711, *Common management information protocol specification for CCITT applications,* CCITT, Geneva, Mar 1991

[X.720] CCITT Recommendation X.720, *Information technology - Open Systems Interconnection, Structure of management information : Management information model.* CCITT, Geneva, Jan 1991

[X.721] CCITT Recommendation X.721, *Information technology - Open Systems Interconnection, Structure of management information : Definition of management information.* CCITT, Geneva, Feb 1992

[X.733] CCITT Recommendation X.733, *Information technology - Open Systems Interconnection, System management: Alarm reporting function* CCITT, Geneva, Feb 1992

[X.734] CCITT Recommendation X.734, *Information technology - Open Systems Interconnection, System management: Event report management function,* CCITT, Geneva, Sep 1992

[X.735] CCITT Recommendation X.735, *Information technology - Open Systems Interconnection, System management: Log control function,* CCITT, Geneva, Sep, 1992

ANEXO A – DEFINIÇÃO DAS CLASSES

1. Classe : **central**

Representa a central que emite os alarmes e da qual o sistema irá receber os alarmes.

Atributos :

usuario : identificação do usuário para conexão com o Oracle.

senha : senha do usuário para conexão com o Oracle.

central_ID : identificação do modelo da central.

arquivo_in : identificação do arquivo de entrada, arquivo do qual será lido os alarmes emitidos pela central.

arquivo_out : identificação do arquivo de saída no qual serão gravados os relatórios de alarmes padronizados.

Objetos Contidos : connection
alarme

Funções-membro :

Inicializacao : faz a atribuição das variáveis do objeto e faz a conexão com o Oracle.

start : faz a chamada das funções-membro *inicializacao* e *start* do objeto *alarme*.

2. Classe : **alarme**

Efetua o processo de leitura e decodificação dos alarmes com auxílio dos objetos da classe *mascara* e *picture*.

atributos :

central_ID : identificação do modelo da central.

ln_alm : contém a linha do alarme que esta sendo analisado.

ln_msc : contém a mascara referente a linha do alarme que esta sendo analisado.

ult_msc : contém a mascara da última linha do alarme

variavel : nome da variável que esta sendo analisada.

conteudo : conteúdo da variável que está sendo analisada

vPicture : formato (picture) da variável que está sendo analisada, caso exista um formato definido para ela..

nome_arquivo : nome do arquivo de entrada para a leitura dos alarmes emitidos pela central.

arq : ponteiro do arquivo de entrada para a leitura dos alarmes.

Objetos Contidos : connection
mascara
linhasnulas
variaveis
picture

Funções membro :

inicializacao : efetua a atribuição das variáveis do objeto, abrir o arquivo do qual será feita a leitura dos alarmes, fazer a chamada de inicialização dos objetos *variaveis*, *linhas_nulas*, *mascara* e *picture*.

start : inicia o processo de leitura dos alarmes, decodificação e adequação através das funções membros do objeto *alarme* e dos objetos inclusos nesse.

3. Classe : **tabela**

Classe da qual são derivadas as classes que fazem pesquisa em alguma tabela no banco de dados do Oracle.

Atributos :

SQL : comando SQL a ser executado.

pvar : vetor de ponteiros associado às variáveis declaradas nas classes derivadas, onde cada variável esta viculada a um campo de saída do comando SQL.

var_len : vetor que contém o tamanho máximo do campo de cada coluna do comando SQL

qvar : indica a quantidade de colunas no comando SQL.

Resp : indica se a execução do comando SQL teve sucesso (1) ou não (0).

Objetos Contidos : connection
cursor

Funções-membro :

connectar : faz a conexão utilizando o objeto do tipo conn, que é do tipo *connection*, criando o cursor, necessário para a comunicação entre o C++ e o Oracle.

executar_SQL : Executa um comando SQL armazenado em uma variavel SQL.

4. Classe : **linhas_nulas** derivada de **tabela.**

Classe para acessar os registros da tabela LINHASNULAS que armazena as linhas que devem ser ignoradas no relatório de alarme, por não serem significativas.

Atributos :

central_ID : identificação do modelo da central.

ln_nulas : vetor contendo as linhas nulas obtidas da tabela LINHASNULAS.

Funções-membro :

leitura : faz a leitura das linhas cadastradas na tabela LINHASNULAS.

diferente : verifica se a linha do alarme é diferente de todas as linhas descartáveis cadastradas na tabela LINHASNULAS.

5. Classe : **mascara** derivada de **tabela.**

Classe para acessar as máscaras cadastradas na tabela MASCARA.

Atributos :

central_ID : identificação do modelo da central

linha_ID : contém o identificador de linha de máscara referente ao último acesso à tabela MASCARA através do comando SQL.

linha_ID_ant : contém o identificador de linha de máscara referente ao acesso anterior à tabela MASCARA.

linha : contém a máscara referente ao último acesso à tabela MASCARA

Funções-membro :

inicializar : inicializar as variáveis do objeto.

proxima_linha : pesquisa a próxima linha de máscara, baseado na última linha pesquisada.

ultima_linha : localiza a última linha de máscara (linha final).

inicializar_linha_ID : atribui a variável de pesquisa para que a função *proxima_linha* localize a primeira linha de máscara.

set_linha_ID_anterior : atribui a variável de pesquisa para que a função *proxima_linha* localize a mesma máscara da pesquisa anterior.

6. Classe : **variaveis**

Armazena as variáveis detectadas no alarme, adequar ou localizar informações do alarme contidas no banco de dados e gravar o relatório padronizado em um arquivo de saída.

Atributos :

nome : vetor que contém o nome das variáveis identificadas em um alarme.

conteudo : vetor com o valor associado às variáveis.

nome_arquivo : nome do arquivo no qual será gravado o relatório de alarme padronizado.

arq_saida : ponteiro para o arquivo no qual é gravado o relatório de alarme padronizado.

n : indica a próxima posição na qual será armazenado uma variável nos vetores *nome* e *conteudo*.

ini : indica que as variáveis armazenadas nas posições anteriores a *ini* nos vetores *nome* e *conteudo* já estão "gravadas", ou seja confirmadas.

p : vetor de classificação das variáveis para a emissao do relatório padrão.

Objetos: codigo_alarme
tipo_evento
alarmeXcausa
causa_prov
severidade
backup
proposta
problema_esp
txt_adicional
inf_adicional

Funções-membro :

define_arq_saida : permite definir o nome do arquivo no qual será gravado o resultado.

inicializar : limpar as variáveis e inicializar os objetos contidos dentro do objeto *variaveis*.

limpar : limpar as variáveis.

atribuir : inserir uma variável e seu conteúdo no objeto.

emitir : grava as variáveis no arquivo especificado em *define_arq_saida*.

gravar : confirmar as variáveis incluídas desde a última gravação.

rejeitar : exclui todas as variáveis atribuídas desde a última gravação.

completar : complementa as variáveis através do uso dos objetos inclusos no objeto *variaveis*.

7. Classe : **picture** derivada de **tabela**.

Classe para acesso ao formatos das variáveis armazenados na tabela PICTURE.

Atributos :

central_ID : identificação do modelo da central.

variavel : nome da variável da qual se deseja localizar o formato.

vpicture : contém o formato da variável, caso exista.

Funções-membro :

inicializar : inicializar as variáveis e abrir o cursor.

set_variavel : atribuir o nome da variável a ser pesquisada.

get_picture : obter a formatação correspondente ao último nome de variável indicado em *set_variavel* e pesquisado em *pesquisar*.

pesquisar : pesquisar o formato de uma variável indicada em *set_variavel*.

8. Classe : **codigo_alarme** derivada de **tabela.**

Classe que acessa o código do alarme armazenado na tabela ALARMES informando o texto do alarme, que é a identificação do alarme.

Atributos :

central_ID : identificação do modelo da central.

alarme : identificação do alarme contido no relatório de alarme emitido pela central.

codigo_alarme : código do alarme referente ao alarme.

codigo_tipo : código do tipo do evento associado ao alarme.

Funções-membro :

inicializar : inicializar variáveis, e abrir o cursor

set_alarme : atribuir valor a variável que contem a identificação do alarme vindo da central.

get_cod_alarme : obter o valor do código de alarme pesquisado.

get_codigo_tipo_evento : obter o valor do código do tipo do evento.

pesquisar : efetuar pesquisa com o último alarme informado na função *set_alarme.*

9. Classe : **tipo_evento** derivada de **tabela.**

Classe para acessar a descrição do tipo do evento.

Atributos :

codigo_tipo : contém o código do tipo para ser pesquisado.

descricao : contém a descrição do tipo evento referente ao código pesquisado.

Funções-membro :

inicializar : inicializar as variáveis e abrir o cursor.

set_cod_tipo : atribuir valor ao código do tipo do evento que se deseja localizar.

get_tipo_evento : obter o valor do tipo de evento correspondente ao código do tipo do evento atribuído em *set_cod_tipo*.

pesquisar : pesquisar registro com o código do tipo de evento.

10. Classe : **alarmeXcausa** derivada de **tabela.**

Classe para acessar a causa provável de um determinado alarme.

Atributos :

central_ID : identificação do modelo da central.

cod_alarme : código do alarme a ser pesquisado.

cod_causa : código da causa provável associada ao alarme.

Funções-membro :

inicializar : inicializar variáveis e abrir o cursor

set_cod_alarme : atribuir valor a variável de pesquisa que contém o código do alarme.

get_cod_causa : obter o valor do último código de causa localizado junto com o alarme por *pesquisar.*

pesquisar : localizar um registro com o código de alarme informado em *set_cod_alarme.*

11. Classe : **causa_prov** derivada de **tabela.**

Classe para acesso a descrição da causa provável.

Atributos :

cod_causa : código da causa provável a ser pesquisada.

causa : descrição da causa provável

complemento : texto com maiores detalhes sobre a causa provável.

Funções-membro :

inicializar : inicializar variáveis e abrir o cursor.

set_cod_causa : atribuir valor ao código de causa.

get_causa : obter o valor da causa provável.

get_complemento : obter valor de texto complementar da causa provável.

pesquisar : localizar um registro com o código de causa atribuído em *set_cod_causa.*

12. Classe : **problema_esp** derivada de **tabela.**

Classe para acessar os dados do problema específico referente a um alarme.

Atributos :

central_ID : identificação do modelo da central.

cod_alarme : código do alarme para a pesquisa do problema específico.

object_id : identificação do objeto associado ao problema específico.

cod_causa : código da causa provável que indica o problema espacífico.

causa : causa provável.

descricao : descrição com mais detalhes da causa provável.

Objetos Contidos : causa_prov

Funções-membro :

inicializar : inicializar as variáveis e abrir o cursor.

set_cod_alarme : atribuir valor ao código do alarme.

get_object_id : retorna o identificador do objeto, caso o problema específico seja identificado por um objeto.

get_cod_causa : retorna o código da causa provável, caso o problema específico seja identificado por um dos itens de causa provável.

get_causa : retorna a descrição da causa provável identificada pelo campo COD_CAUSA da tabela PROB_ESP.

get_descricao : retorna o texto complementar da causa provável indicada com problema específico.

pesquisar : pesquisar o registro do problema específico referente ao código do alarme informado.

13. Classe : **severidade** derivada de **tabela.**

Classe para determinar a severidade dentro do padrão TMN referente a severidade apresentada no relatório de alarme da central..

Atributos :

central_ID : identificação do modelo da central.

severidade_central : severidade que consta no alarme da central.

severidade_tmn : severidade no padrão TMN correspondente à severidade do alarme da central.

Funções-membro :

inicializar : inicializar as variáveis e abrir o cursor.

set_severidade_central : atribuir valor à severidade da central.

get_severidade_tmn : obter o valor da severidade TMN correspondente à severidade da central atribuída em *set_severidade_central.*

pesquisar : pesquisar registro com severidade da central informado.

14. Classe : **backup** derivada de **tabela.**

Classe que verifica se existe um objeto de backup para o alarme em questão, Essa classe atende aos parâmetros *Backed-up Status* e *Backup object*. O parâmetro *Backed-up Status* é indicado *true* se existir ou *false* se não existir um objeto de backup.

Atributos :

central_ID : identificação do modelo da central.

cod_alarme : código do alarme do qual se deseja verificar se o mesmo tem objeto de backup.

objeto_ID : quando existir, contém a identificação do objeto de backup para o alarme.

Funções-membro :

inicializar : inicializar as variáveis e abrir o cursor.

set_cod_alarme : atribuir valor ao código do alarme.

get_objeto : obter o identificador do objeto que irá ser usado de backup.

pesquisar : pesquisar registro com o objeto de backup referente ao código do alarme informado.

15. Classe : **proposta** derivada de **tabela.**

Classe para verificar se um determinado alarme necessita de uma ação de reparo do gerente ou não.

Atributos :

central_ID : identificação do modelo da central.

cod_alarme : código do alarme do qual se deseja verificar se há de cessidade de uma ação de reparo.

proposta : contém um valor indicando se uma ação de reparo é necessária (*repair action required*) ou não (*no repair action required*)

Funções-membro :

inicializar : inicializar as variáveis e abrir o cursor.

set_cod_alarme : atribuir valor ao código do alarme.

get_proposta : obter o texto que indica se uma ação de reparo é necessária (*repair action required*), ou se uma ação de reparo não é necessária (*no repair action required*).

pesquisar : pesquisar registro da proposta de ação de reparo, referente ao código do alarme informado.

16. Classe : txt_adicional derivada de tabela.

Classe para acessar o texto adicional referente a um código de alarme..

Atributos :

central_ID : identificação do modelo da central

cod_alarme : código do alarme para verificar a existência de texto adicional

descricao : contém a descrição do texto adicional.

Funções-membro :

inicializar : inicializar as variáveis e abrir o cursor.

set_cod_alarme : atribuir valor ao código do alarme.

get_txt_adicional : obter o valor do texto adicional, correspondente ao código do alarme pesquisado.

pesquisar : pesquisar registro do texto adicional, referente ao código do alarme informado.

17. Classe : **inf_adicional** derivada de **tabela**

Classe para acessar a informação adicional referente a um código de alarme..

Atributos :

central_ID : identificação do modelo da central.

cod_alarme : código do alarme do qual se deseja acessar a informação adicional.

object_id : contém a identificação do objeto referente a informação adicional

significancia : indica a significancia da informação adicional

informacao : texto que complementa a informação adicional.

Funções-membro :

inicializar : inicializar as variáveis e abrir o cursor.

set_cod_alarme : atribuir valor ao código do alarme.

get_object_id : retorna o identificador de objeto da informação adicional.

get_significancia : retorna a significância da informação adicional.

get_informacao : obter o valor do texto correspondente às informações a respeito do problema.

pesquisar : pesquisar registro do texto adicional referente ao código do alarme informado.

18. Classe : connection

Classe para fazer a conexão com o Oracle, através de um usuário e senha previamente cadastrados no banco de dados.

Atributos :

lda : LDA (*login data area*)
Um LDA é uma área de dados associada com uma conexão ativa.

hda : HDA (*host data área*)
O HDA é uma estrutura de dados de 256 bytes que é necessário para conectar com um servidor Oracle.

state : indica se o objeto esta conectado à um servidor Oracle ou não.

Funções-membro :

connect : faz a conexão.

disconnect : encerra a conexão.

19. Classe : **cursor**

O cursor é uma área de transferência de dados entre o programa em C++ e o Oracle.

Atributos :

cda : CDA (*cursor data area*)
Um CDA provê o mapeamento entre o cursor do programa em C++ e a representação de um comando SQL no servidor, e através do qual é enviado o comando SQL e recebido o resultado do SQL.

state : indica se o cursor esta conectado a um servidor Oracle.

Funções-membro :

open : abrir o cursor para permitir executar comandos SQL, através da função *oopen* que associa uma área de dados do programa com uma área de dados no Oracle.

isopen : verificar se o cursor já está aberto.

close : fechar o cursor.

parse : faz o parse de um comando SQL, que é necessário antes de executá-lo.

bind_by_position : associa o endereço de uma variável de programa com uma variável em um comando SQL.

define_by_pocision : vincula um campo de saída do comando SQL com uma variável do programa.

describe : retorna o tipo e tamanho de um campo de saída do comando SQL.

execute : executa um comando SQL.

fetch : retorna uma linha do resultado obtido na execução de um comando SQL.

get_error_code : retorna o código de erro, quando houver, após a execução de um comando SQL.

display_erro : apresentar a mensagem de erro.

ANEXO B – DEFINIÇÃO DAS TABELAS

A seguir são descritas as tabelas apresentadas as tabelas que constituem o banco de dados utilizado no sistema.

1. ALARMES

Contém a relação dos alarmes emitidos pela central e um desses um código associado. A tabela permite converter essa identificação textual do alarme para um código, o que é mais eficiente para o uso em um banco de dados. Nessa tabela consta também o código do tipo de evento.

campo	**descrição**
CENTRAL	identificação do modelo da central a que pertence o alarme.
ALARME	identificação do alarme
CODALARME	código do alarme a ser utilizado no sistema.
COD_TIPO	código do tipo do evento ao qual o alarme está associado.

2. MASCARA

Nessa tabela, cada registro representa uma linha do relatório de alarme emitido pela central. Nessas linhas é indicado os dados a serem extraídos desse relatório de alarme através da inclusão de identificadores, denominados de variáveis.

As variáveis são identificadas na máscara por serem uma seqüência de letras (de a até z em minúsculo. Não é aceito outros caracteres como número ou caracteres especiais. Essa restrição vem em decorrência que um caractere diferente de uma letra minúscula é considerado pelo sistema como um caractere do alarme.

Caso exista alguma informação que não consta em todos os alarmes, essa deve ser identificada como uma variável opcional. Uma variável opcional é identificada na máscara por estar entre colchetes (**[limpar]**), como é mostrado no exemplo acima com a variável *limpar*, pois essa informação apenas aparece em um alarme que tem como objetivo de informar o cancelamento de um alarme anterior (*cleared*).

campo	**descrição**
CENTRAL	identificação do modelo da central a que pertence a mascara.
LINHA	indica a linha a que se refere no alarme. Esse campo indica a seqüência das linhas de máscara em função das linhas do relatório do alarme.
CONTEUDO	contém a máscara.

3. PICTURE

Essa tabela auxilia o objeto *alarme* a identificar o conteúdo das variáveis através da definição de um formato para a informação que está associada à uma variável.

Nessa tabela, é registrado para cada variável, quando necessário, o nome da variável e um formato. Esse formato é utilizado para avaliar se o conteúdo lido do alarme corresponde ao formato definido para a variável.

Para se definir um formato são utilizados apenas três símbolos:

A – indica que só é aceito letras.

9 – indica que só é aceito dígitos.

X – indica que é aceito qualquer caractere.

Como exemplo, será definido o formato de duas variáveis definidas na máscara da primeira linha do relatório de alarme da central AXE-10. As variáveis utilizadas serão *nralarme* e *severidade.*

nralarme = 999 (aceita três dígitos)

severidade = A9 (o primeiro caractere deve ser uma letra e o segundo caractere deve ser um dígito)

Quando um formato é indicado para uma variável, além do sistema avaliar o tipo de caractere, é verificado a quantidade de caracteres informados no formato.

campo	**descrição**
CENTRAL	identificação do modelo da central a que pertence o formato.
VARIAVEL	nome da variável que deve obedecer ao formato.
PICTURE	formato utilizado para avaliar o conteúdo de uma variável.

4. LINHASNULAS

Nem todas as linhas vindas da central referem-se a linhas de alarme, por exemplo, as linhas de cabeçalho ou linhas de salto de páginas. Essas linhas podem aparecer em qualquer linha do alarme.

Para que essas linhas sejam ignoradas pelo sistema, deve-se cadastrá-las na tabela LINHASNULAS. Não há necessidade de informar toda a linha, basta apenas informar os caracteres iniciais que sejam suficientes para identificar a linha a ser descartada.

campo	**descrição**
CENTRAL	identificação do modelo da central a qual pertence a linha a ser desconsiderada pelo sistema.
CONTEUDO	contém o texto da linha que deve ser desconsiderada.

5. TIPO_EVENTO

No alarme da central AXE10, o tipo do evento não é informado, assim é necessário que esse seja cadastrado previamente. O código do tipo do evento fica registrado na tabela ALARMES, apresentada no item 1, e a descrição do tipo do evento é obtida na tabela TIPO_EVENTO.

campo	**descrição**
COD_EVENTO	código que identifica o tipo do evento.
DESCRICAO	descrição do tipo do evento conforme recomendação [X.733]

6. ALARME_CAUSA

A causa provável não é informada pela central. No sistema, ela foi incluída através das tabelas ALARME_CAUSA e CAUSAPROV.

A tabela ALARME_CAUSA contém a identificação da central, o código do alarme e o código da causa provável, associando cada alarme a uma determinada causa provável.

campo	**descrição**
CENTRAL	identificação do modelo da central
COD_ALARME	código do alarme.
COD_CAUSA	código da causa provável.

7. CAUSAPROV

A tabela CAUSAPROV permite decodificar o código da causa provável indicado na tabela ALARME_CAUSA. Esta contém as causa prováveis sugeridas pela recomendação [X.733].

campo	**descrição**
CODIGO	Código da causa provável
CAUSA	A causa provável.
DESCRICAO	texto com detalhamento ou explicações referente a causa provável.

8. PROB_ESP

Permite cadastrar um problema específico referente ao alarme. Quando existente, o problema específico é um detalhamento da causa provável.

campo	**descrição**
CENTRAL	identificação do modelo da central.
COD_ALARME	código do alarme ao qual se refere o problema específico
OBJECT_ID	caso exista, identifica o objeto para o problema específico.
COD_CAUSA	caso exista, identifica o código da causa provável.

9. SEVERIDADE

Essa tabela se faz necessário para que o sistema possa adequar a severidade apresentada no alarme, pois possui uma codificação própria e não conforme a recomendação da TMN.

campo	**descrição**
CENTRAL	Identificação do modelo da central.
SEVCENTRAL	identificação da severidade emitida pela central.
SEVTMN	contém a severidade conforme definido na recomendação [X.733] relacionada à severidade da central.

10. BACKUP

Essa tabela permite informar parada cada alarme, se necessário, o objeto de backup do mesmo.

campo	**descrição**
CENTRAL	Identificação do modelo da central.
COD_ALARME	código do alarme ao qual o objeto está associado.
OBJETO	identificação do objeto de backup.

11. PROPOSTA

Nessa tabela são cadastradas as indicações da necessidade ou não de uma ação de reparo pelo gerente para o alarme.

campo	**descrição**
CENTRAL	Identificação do modelo da central.
CODALARME	Código do alarme ao qual a proposta esta vinculada.
PROPOSTA	caso o conteúdo seja 1 indica que uma ação de reparo se faz necessário, caso o conteúdo seja 0 não é necessário uma ação de reparo.

12. TXTADICIONAL

Essa tabela contém um texto de formato livre que será reportado juntamente com o alarme, que é referente ao parâmetro texto adicional.

campo	**descrição**
CENTRAL	Identificação do modelo da central.
CODALARME	Código do alarme ao qual a proposta está vinculada.
DESCRICAO	texto adicional.

13. INFADICIONAL

Essa tabela contém os dados das informações adicionais para os alarmes, quando necessário.

campo	**descrição**
CENTRAL	Identificação do modelo da central.
CODALARME	Código do alarme ao qual a informação adicional está vinculada.
OBJECT_ID	identificado do objeto da informação adicional.
SIGNIFICANCIA	indica a significância da informação adicional (*true/false*)
INFORMACAO	informações referente a informação adicional.

ANEXO C – CÓDIGO FONTE DO SISTEMA

sistema.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

class central;

main()
{
    central AXE10;

    //                      objeto | ident.    | arquivo      | arquivo
    //                      conexao| central   | entrada      | saida
    AXE10.inicializacao( "system" , "manager" ,  // Conexao no Oracle
                         "AXE10"  ,              // Ident. Central
                         "alarm02.txt" ,         // arquivo entrada
                         "teste.out" );          // arquivo saida

    AXE10.start();
}

void err_report(FILE * file , text *errmsg, sword func_code)
{
   fprintf(file,"\n-- ORACLE error--\n\n%s\n",errmsg);
   if(func_code > 0)
      fprintf(file,"Processing OCI function %s\n",
              oci_func_tab[func_code]);
}
```

central.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}


/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class alarme;
extern void err_report(FILE *file, text *errmsg, sword func_code);

class central
{
   private:
      connection conn;
      alarme Alarme;

      FILE * arq;
      char usuario[20],
           senha[20],
           central_ID[25],
           arquivo_in[64],
           arquivo_out[64];

      char pai[80];

   public:
      ~central();

   void inicializacao( char * usuario_param,
                       char * senha_param,
                       char * central_ID_param ,
                       char * nome_arq_in_param ,
                       char * nome_arq_out_param );

      void start();

};
```

```
void central::inicializacao( char * usuario_param,
                             char * senha_param,
                             char * central_ID_param ,
                             char * nome_arq_in_param ,
                             char * nome_arq_out_param )
{
   strcpy(pai,central_ID_param);

    strcpy( usuario     , usuario_param      );
    strcpy( senha       , senha_param        );
    strcpy( central_ID  , central_ID_param   );
    strcpy( arquivo_in  , nome_arq_in_param  );
    strcpy( arquivo_out , nome_arq_out_param );

    if (conn.connect( usuario , senha , central_ID ))
    {
       conn.display_error(stderr);
       return;
    };

}

void central::start()
{
   Alarme.inicializacao(&conn, central_ID, arquivo_in, arquivo_out);
   Alarme.start();
}

central::~central()
{
    conn.disconnect();
}
```

connection.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2
```

```
extern void err_report(FILE *file, text *errmsg, sword func_code);
//#########################################################################

class connection
{
  friend class cursor;

  private:
    ub1 hda[HDA_SIZE];
    enum conn_state
    {
      not_connected,
      connected
    };
    conn_state state;

  public:

    Lda_Def lda;

    connection()
      { state = not_connected;
        memset(&lda,0,sizeof(Lda_Def));
        memset(&hda,0,HDA_SIZE);
      }
    ~connection();
    sword connect(const text *username, const text *password,
                  char *ppai );
    sword disconnect();
    void display_error(FILE* file) const;

};
//#########################################################################

/*
 * Codigo dos Erros
 */
#define CONERR_ALRCON -1            /* already connected */
#define CONERR_NOTCON -2            /* not connected */
#define CURERR_ALROPN -3            /* cursor is already open */
#define CURERR_NOTOPN -4            /* cursor is not opened */


/* exit status upon failure */
#define  EXIT_FAILURE       1

/* connection destructor */
connection::~connection()
{

  // disconnect if connection exists
  if (state == connected)
  {
    if (disconnect())
    {
      display_error(stderr);
    }
  }
}
```

```
/* connect to ORACLE */
sword connection::connect(const text *username,
                          const text *password, char * ppai)
{
  sword status;
  if (state == connected)
  {
    // this object is already connected
    return (CONERR_ALRCON);
  }

  if ((status = olog(&lda, hda, (text *)username, -1,
                     (text *)password, -1, (text *) 0, -1,
                     OCI_LM_DEF)) == 0)
  {
    // successful login
    state = connected;
    printf("Connected to ORACLE as %s\n", username);
  }
  return (status);
}
//###########################################################################

/* disconnect from ORACLE */
sword connection::disconnect()
{

  sword status;
  if (state == not_connected)
  {
    // this object has not been connected
    return (CONERR_NOTCON);
  }
  if ((status = ologof(&lda)) == 0)
  {
    // successful logout
    state = not_connected;
  }
  return (status);
}
//###########################################################################

/* write error message to the given file */
void connection::display_error(FILE *file) const
{

  if (lda.rc != 0)
  {
    sword n;
    text msg[512];
    n = oerhms((cda_def *)&lda, lda.rc, msg, (sword) sizeof(msg));
    err_report(file, msg, lda.fc);
  }
}
//###########################################################################
```

cursor.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1

#define  NATIVE             1
#define  VERSION_7          2

//class connection;
extern void err_report(FILE *file, text *errmsg, sword func_code);

/*
 * This class represents an ORACLE cursor.
 *
 * NOTE: This cursor class is just given as an example and all
 * possible operations on a cursor have not been defined.
 */
class cursor
{
  private:
    Cda_Def cda;
    connection *conn;
    enum cursor_state
    {
      not_opened,
      opened
    };
    cursor_state state;

  public:
    cursor()
    {
       state = not_opened;
      // conn = (connection *) 0;
    }
    ~cursor();
    sword open(connection *conn_param);
    sword close();
    sword isopen();

/*---------------------------------------------------------*/
    sword parse(const text *stmt)
      { return (oparse(&cda, (text *)stmt, (sb4)-1,
                       DEFER_PARSE, (ub4) VERSION_7)); }
/*---------------------------------------------------------*/
```

```
    /* bind an input variable */
    sword bind_by_position(sword sqlvnum, ub1 *progvar,
                           sword progvarlen, sword datatype,
                           sword scale, sb2 *indicator)
      { return (obndrn(&cda, sqlvnum, progvar, progvarlen,
                       datatype, scale, indicator, (text *)0,
                       -1, -1)); }
/*---------------------------------------------------------*/
    /* define uma variavel de saida */
    sword define_by_position(sword position, ub1 *buf,
                             sword bufl, sword datatype,

      sword scale, sb2 *indicator,
                             ub2 *rlen, ub2 *rcode)
      { return (odefin(&cda, position, buf, bufl, datatype,
                       scale, indicator,
                       (text *)0, -1, -1, rlen, rcode)); }
/*---------------------------------------------------------*/
    sword describe(sword position, sb4 *dbsize, sb2 *dbtype,
                   sb1 *cbuf, sb4 *cbufl, sb4 *dsize, sb2 *prec,
                   sb2 *scale, sb2 *nullok)
      { return (odescr(&cda, position, dbsize, dbtype,
                       cbuf, cbufl, dsize,  prec, scale, nullok));
      }
/*---------------------------------------------------------*/
    sword execute()
      { return (oexec(&cda)); }
/*---------------------------------------------------------*/
    sword fetch()
      { return (ofetch(&cda)); }
/*---------------------------------------------------------*/
    sword get_error_code() const
      { return (cda.rc); }
/*---------------------------------------------------------*/
    void display_error( FILE* file) const;
/*---------------------------------------------------------*/

};
/*
 * codigo dos Erros
 */
#define CONERR_ALRCON -1            /* ja conectado */
#define CONERR_NOTCON -2            /* nao conectado */
#define CURERR_ALROPN -3            /* o cursor ja esta aberto */
#define CURERR_NOTOPN -4            /* o cursor nao esta aberto*/


/* exit status upon failure */
#define  EXIT_FAILURE       1
```

```
/* cursor destructor */
cursor::~cursor()
{
  if (state == opened)
  {
    if (close())
    {
      display_error(stderr);
    }
  }
}
/* open the cursor */
sword cursor::open(connection *conn_param)
{
  sword status;
  if (state == opened)
  {
    // this cursor has already been opened
    return (CURERR_ALROPN);
  }
  if ((status = oopen(&cda, &conn_param->lda, (text *)0, -1, -1,
                      (text *)0, -1)) == 0)
  {
    // successfull open
    state = opened;
    conn = conn_param;
  }
  return (status);
}

/* close the cursor */
sword cursor::close()
{
  sword status;
  if (state == not_opened)
  {
    // this cursor has not been opened
    return (CURERR_NOTOPN);
  }
  if ((status = oclose(&cda)) == 0)
  {
    // successful cursor close
    state = not_opened;
    conn = (connection *)0;
  }
  return (status);
}

/* Verifica se o cursor ja esta aberto */
sword cursor::isopen()
{
  sword status;
  if (state == opened)
     status = 1;
  else
     status = 0;

  return (status);
}
```

```
/* escreve a mensagem de em um determinado dispositivo */
void cursor::display_error(FILE *file) const
{
  if (cda.rc != 0)
  {
    sword n;
    text msg[512];
    n = oerhms(&conn->lda, cda.rc, msg, (sword) sizeof(msg));
    err_report(file, msg, cda.fc);
  }
}
```

alarme.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}


/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class linhas_falhas;
class connection;
class mascara;
class variaveis;
//#####################################################################

class alarme
{
   private:
      linhas_nulas   LinhasNulas;
      connection     *conn;
      mascara        Mascara;
      variaveis      Variaveis;
      picture        Picture;

      char ln_msc[136]    , ln_alm[136], ult_msc[136],
           central_ID[25] , nome_arquivo[64];

      char variavel[35], conteudo[136],vPicture[68];
      int p,pAl,pMa,Resp;


      FILE * arq;
      void Tira_Espacos();
```

```
      int proxima_linha();
      int Eh_Alarme();
      int Pega_Var();
      int Tira_Var( int N );
      void ler_variavel();
      void ler_variavel_opcional();
      void ler_conteudo();
   public:
      ~alarme();
      void inicializacao( connection *conn_param,
                          char * central_ID_param ,
                          char * nome_arq_ent_param ,
                          char * nome_arq_sai_param );
      void start();
};
//#########################################################################

alarme::~alarme()
{
   fclose(arq);
   conn = (connection *)0;
}
//#########################################################################

void alarme::inicializacao( connection *conn_param ,
                            char * central_ID_param ,
                            char * nome_arq_ent_param ,
                            char * nome_arq_sai_param )
{
   // armazenar os parametros de inicializacao.
   conn = conn_param;

   strcpy(central_ID , central_ID_param );
   strcpy(nome_arquivo , nome_arq_ent_param );

   // abrir o arquivo para leitura dos alarmes.
   arq=fopen(nome_arquivo,"r");

   // Determinar em que arquivo sera armazendado o
   // resultado dos alarmes
   Variaveis.define_arq_saida( nome_arq_sai_param );
   Variaveis.inicializar( conn , central_ID );

   // Leitura da definicao da linhas falsas.
   LinhasNulas.leitura( conn , central_ID );

   // Definicao e inicializacao da Mascara.
   Mascara.inicializar( conn , central_ID);
   Mascara.inicializar_linha_ID();

   // Definicao e Inicializacao da Picture.
   Picture.inicializar( conn , central_ID);

}
//#########################################################################
```

```
void alarme::start()
{

   Mascara.ultima_linha( ult_msc );
   Mascara.inicializar_linha_ID();

   while ( 1 )
   {
      if ( !Mascara.proxima_linha( ln_msc ) )
      {
          Mascara.inicializar_linha_ID();
          Variaveis.emitir();
          Variaveis.limpar();
          continue;
      }

      if (proxima_linha()) // le linha de alarme
      {
         if ( strcmp(ln_alm,ult_msc)==0 ) // fim do alarme
         {
            Mascara.inicializar_linha_ID();
            Variaveis.emitir();
            Variaveis.limpar();
         }
         else
         {
            if (Eh_Alarme())
               Variaveis.gravar();
            else
               Mascara.set_linha_ID_anterior();

            if ( strcmp(ln_msc,"txtadicional") == 0 )
            {
               Mascara.set_linha_ID_anterior();
            }
         }
      }
      else
          break;
   }

Mascara.mascara::~mascara();
LinhasNulas.linhas_nulas::~linhas_nulas();
Variaveis.~variaveis();
Picture.~picture();
}
//#####################################################################
```

```
int alarme::proxima_linha()
{
   while (fgets(ln_alm,132,arq) != NULL ) // le linha de alarme
   {
      Tira_Espacos();
      if (strlen( ln_alm ) > 1)
         if ( LinhasNulas.diferente( ln_alm ) )
         {
             return 1;
         }
   }

return 0;
}
//#########################################################################

void alarme::Tira_Espacos()
{
    unsigned int pi,pn;
    ln_alm[ strlen( ln_alm )-1 ] = '\0'; // excluir o caracter de
                                         // retorno de linha
    if ( ln_alm[0]==' ' )
    {
       pi=0; pn=0;
       while ( ln_alm[pi] == ' ' ) pi++;

       while ( ln_alm[pi] != '\0') ln_alm[pn++] = ln_alm[pi++];
       ln_alm[pn]='\0';
    }
}
//#########################################################################

int alarme::Eh_Alarme()
{
   char MaCop[133];
   int Tentativa,i,Resp,TemVar;

   Resp=0;
   TemVar=1;
   Tentativa = 0;
   strcpy( MaCop , ln_msc );
   while ( 1 )
   {
      Resp = Pega_Var();

      if ((Resp == 2) && (TemVar == 0)) { Resp=0; break; }
      if  (Resp == 1) break;

      Variaveis.rejeitar();
      Tentativa++;
      strcpy( ln_msc , MaCop );
      TemVar = Tira_Var( Tentativa );
   }
   strcpy( ln_msc , MaCop );

return Resp;
}  // Fim Eh_Alarme
//#########################################################################
```

```
/* Funcao para extrair as variaveis da mascara e seu conteudo
   do alarme.
   A funcao retorna os seguintes valores:
   0 = nao foi possivel extrair, fazer outra tentativa;
   1 = obtencao correta;
   2 = Erro, o alarme nao se enquadra na mascara.
*/
int alarme::Pega_Var()
{
   // variaveis utilizadas --> pAl,pMa,p,Resp=1;
   char c;

   Resp=1;

   strcpy(variavel,"\0");
   strcpy(conteudo,"\0");

   pAl=0;
   pMa=0;
   while ( ( ln_alm[pAl] != NULL ) &&
           ( ln_msc[pMa] != NULL ) &&
           ( Resp != 2 ) )
   {
      if (ln_alm[pAl] == ln_msc[pMa] )
      {
         pAl++;
         pMa++;
      }
      else
      {
         if (ln_alm[pAl] == ' ') while (ln_alm[pAl] == ' ') pAl++;
         else

         if (ln_msc[pMa] == ' ') while (ln_msc[pMa] == ' ') pMa++;
         else

         if ( (ln_msc[pMa] == '[') &&
              ((ln_msc[pMa+1] >= 'a') && (ln_msc[pMa+1] <= 'z')) )
         {
             // Ler o nome da variavel.
             ler_variavel_opcional();

             ler_conteudo();

             Variaveis.atribuir(variavel , conteudo );

             // (*) verifica se na mascara apos a variavel seguia-se
             // um caracter significativo (<>' '), caso afirmativo
             // este deve ser igual ao do alarme, indicando o lugar
             // correto da variavel.
             if ((ln_msc[pMa] != ' ') && (ln_msc[pMa] != ln_alm[pAl]))
             {
                Resp = 0;
                break;
             }

         } // Ma='[' e Ma='a..z'
         else
```

```
        if ((ln_msc[pMa] >= 'a') && (ln_msc[pMa] <= 'z'))
        {
            // Ler o nome da variavel.
            ler_variavel();

            // Ler o conteudo da vaiavel
            ler_conteudo();

            Variaveis.atribuir(variavel , conteudo );

            // ver (*) acima.
            if ((ln_msc[pMa] != ' ') && (ln_msc[pMa] != ln_alm[pAl] ))
            {
               Resp = 0;
               break;
            }

        } // Ma='a..z'
        else
        {
           //======= ERRO ======
           Resp = 2;
           break;
        }
     } // Al = Ma
  } // while Ma e Al <> NULL

return Resp;
} // Pega_Var
//#########################################################################

int alarme::Tira_Var( int N )
{
   unsigned int p,pc,v;
   int Resp = 0;

   p=0;
   pc=0;
   v=1;
   while (ln_msc[p] != NULL)
   {
      if (ln_msc[p] == '[')   // eh uma variavel opcional
      {
         if (( N & v ) == v)  // verifica se a variavel esta na vez
         {
            Resp=1;
            while (ln_msc[p++] != ']');
         }
         v*=2;          // incrementar o contador de variavel atraves
                        // de um deslocamento de um bit a esquerda.
      }
      ln_msc[pc++] = ln_msc[p++];
   } // while

   ln_msc[pc]='\0';
return Resp;
} // tira_var.
//#########################################################################
```

```
void alarme::ler_variavel()
{
   p=0;
   strcpy(variavel,"\0");

   do    variavel[p++]=ln_msc[pMa++];
   while ((ln_msc[pMa] >= 'a') && (ln_msc[pMa] <= 'z'));

   variavel[p]='\0';
}
//#########################################################################

void alarme::ler_variavel_opcional()
{

   pMa++;
   p=0;
   strcpy(variavel,"\0");

   do variavel[p++]=ln_msc[pMa++];
   while ((ln_msc[pMa] != ']') && (ln_msc[pMa] != '\0'));

   pMa++;
   variavel[p]='\0';

}
//#########################################################################
```

```
void alarme::ler_conteudo()
{

   strcpy(conteudo,"\0");

   Picture.set_variavel( variavel );
   if (Picture.pesquisar()) // obter o conteudo de uma
   {                        // varial com picture

      strcpy(vPicture,Picture.get_picture());

      p=0;
      do
      {
         if ( vPicture[p]=='\0' )
         {
            //======= ERRO ======
            Resp = 2;
            break;
         }
         else
         if (( vPicture[p] == 'A' ) &&
             ( ln_alm[pAl] >= 'A' && ln_alm[pAl] <= 'Z'))
               conteudo[p]=ln_alm[pAl];
         else
         if (( vPicture[p] == '9' ) &&
             ( ln_alm[pAl] >= '0' && ln_alm[pAl] <= '9'))
               conteudo[p]=ln_alm[pAl];
         else
         if ( vPicture[p] == 'X' )
            conteudo[p]=ln_alm[pAl];
         else
         {
            //======= ERRO ======
            Resp = 2;
            break;
         }
            p++; pAl++;
      }
      while (((ln_alm[pAl] != ln_msc[pMa]))
               && (ln_alm[pAl]!='\0'));

      conteudo[p]='\0';
   }
   else // obter o conteudo de uma varial sem picture
   {
      p=0;
      do conteudo[p++]=ln_alm[pAl++];
      while (((ln_alm[pAl] != ln_msc[pMa]))
              && (ln_alm[pAl]!='\0'));
         conteudo[p]='\0';
   }

}
//#########################################################################
```

linhas_nulas.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2


class connection;
class cursor;

class linhas_nulas
{
   private:
      connection *conn;
      cursor Cursor;

      char  central_ID[25];
      char  ln_falsas[15][30];

   public:
      ~linhas_nulas()
      {
         conn = (connection *)0;
         Cursor.close();
            }

      int leitura( connection *conn_param,
                   char * central_ID_param );
      int diferente( char * linha );

};

//=================================================================
```

```
int linhas_nulas::leitura( connection *conn_param,
                           char * central_ID_param )
{
   char  SQL[127];
   text *vconteudo;
   sb1 name_buf[20];
   sb2 db_type;
   sb4 conteudolen;
   sword len,dsize;
   int Resp,i,p;

   conn = conn_param;

   strcpy(central_ID,central_ID_param);

   strcpy( SQL,"SELECT conteudo FROM linhasfora WHERE central='");
   strcat( SQL, central_ID);
   strcat( SQL, "'");


   // abrir o cursor
   if ( Cursor.open( conn ))
   {
      conn->display_error(stderr);
      return(EXIT_FAILURE);
   }

   // Parse do SQL
   if (Cursor.parse( SQL ))
   {
      Cursor.display_error(stderr);
      return(EXIT_FAILURE);
   }

   // descrever o campo linha.
   len = sizeof (name_buf);
   if ( Cursor.describe( 1 , (sb4 *) &conteudolen, &db_type,
                         name_buf, (sb4 *) &len, (sb4 *) &dsize,
                         (sb2 *) 0, (sb2 *) 0, (sb2 *) 0)
      )
   {
      Cursor.display_error(stderr);
      return(EXIT_FAILURE);
   }


   //alocar o buffer para linha
   vconteudo = new text( (int) conteudolen + 1);

   //define a variavel de saida
   if( Cursor.define_by_position( 1 , (ub1 *) vconteudo,
                                  conteudolen+1,
                                  STRING_TYPE, -1 , (sb2 *) 0,
                                  (ub2 *) 0, (ub2 *) 0 ) )
   {
      Cursor.display_error(stderr);
      return(EXIT_FAILURE);
   }

   // executar o SQL.
```

```
   if ( ( Cursor.execute() || Cursor.fetch() ) &&
        ( Cursor.get_error_code() != NO_DATA_FOUND ) )
   {
      Cursor.display_error(stderr);
      return(EXIT_FAILURE);
   }

   if (Cursor.get_error_code() == NO_DATA_FOUND) Resp = 0;
   else                                          Resp = 1;

   ln_falsas[0][0] = '\0';

   if (Resp == 1)
   {
      for (i=0 ; i<15  ; i++)
      {

         p = strlen( vconteudo )-1;
         while (vconteudo[p] == ' ') p--;
         p++;
         vconteudo[p] = '\0';
         strcpy( ln_falsas[i],vconteudo);

         if (Cursor.fetch())
         {
            if (Cursor.get_error_code() == NO_DATA_FOUND)
               break;
            if (Cursor.get_error_code() != NULL_VALUE_RETURNED)
               Cursor.display_error(stderr);
         }
      }
      if ( i < 14)
         ln_falsas[i+1][0]  = '\0';
      else
         ln_falsas[14][0]  = '\0';
   } /* Resp = 1 */

   // fechar o cursor
   if ( Cursor.close() )
   {
      Cursor.display_error(stderr);
      return(EXIT_FAILURE);
   }

   conn = (connection *)0;

return Resp;
}
//#########################################################################
```

```
/*
   Verifca se a linha enviada pelo paramentro eh ou nao
   igual a uma das linhas falsas (nao alarmes).
   Se for diferente retorna 1
   ou
   se for igual retorna 0;
*/
int linhas_nulas::diferente( char * linha )
{
//printf("\n lf.diferente");
   int Resp,i,l,p;

   i = 0;
   while ( ln_falsas[i][0] != '\0' )
   {
      l = strlen( ln_falsas[i] );
      Resp = 0;

      for (p=0 ; p < l ; p++ )
         if ( ln_falsas[i][p] != linha[p] )
         {  Resp = 1;  break; }

      if ( Resp == 0 ) break;

      i++;
   }
return(Resp);
}
//#########################################################################
```

variaveis.cpp

```
/*
   Programa para ler uma linha de Alarme e Mascara, e verificar a
   existencia de variaveis.
*/
#include <string.h>
#include <stdio.h>
#include <conio.h>

#define n_var 40    // numero de variaveis possiveis de ser
                    // armazenado.

class variaveis
{
   private:
      codigo_alarme Cod_Alarme;
      tipo_evento   Tipo_Evento;
      alarmeXcausa  Alarme_X_Causa;
      causa_prov    Causa_Prov;
      severidade    Severidade;
      backup        BackUp;
      proposta      Proposta;
      problema_esp  ProblemaEspecifico;
      txt_adicional TxtAdicional;
      inf_adicional InfAdicional;

      char nome[n_var][32],
           conteudo[n_var][128],
           nome_arquivo[64];

      int  n,ini,p[n_var];

      FILE * arq_saida;

   public:
      ~variaveis();
      void define_arq_saida( char * nome_arquivo_param );
      void inicializar( connection *conn_param ,
                        char * central_ID_param );
      void limpar();
      void atribuir( char * nome_param , char * conteudo_param );
      void emitir();
      void gravar();
      void rejeitar();
      void completar();

};
//#####################################################################
```

```
variaveis::~variaveis()
{
   Cod_Alarme.codigo_alarme::~codigo_alarme();
   Tipo_Evento.tipo_evento::~tipo_evento();
   Alarme_X_Causa.alarmeXcausa::~alarmeXcausa();
   Causa_Prov.causa_prov::~causa_prov();
   Severidade.severidade::~severidade();
   BackUp.backup::~backup();
   Proposta.proposta::~proposta();
   ProblemaEspecifico.problema_esp::~problema_esp();
   TxtAdicional.txt_adicional::~txt_adicional();
}
//#########################################################################
void variaveis::define_arq_saida( char * nome_arquivo_param )
{
   strcpy(nome_arquivo , nome_arquivo_param );
   arq_saida = fopen( nome_arquivo , "a"); // a = append.
}
//#########################################################################

void   variaveis::inicializar(   connection   *conn_param   ,   char   *
central_ID_param )
{
   limpar();

   Cod_Alarme.inicializar( conn_param , central_ID_param );
   Tipo_Evento.inicializar( conn_param );
   Alarme_X_Causa.inicializar( conn_param , central_ID_param );
   Causa_Prov.inicializar( conn_param );
   Severidade.inicializar( conn_param , central_ID_param );
   BackUp.inicializar( conn_param , central_ID_param );
   Proposta.inicializar( conn_param , central_ID_param );
   ProblemaEspecifico.inicializar( conn_param , central_ID_param );
   TxtAdicional.inicializar( conn_param , central_ID_param );
   InfAdicional.inicializar( conn_param , central_ID_param );

}
//#########################################################################

void variaveis::limpar()
{
   int i;
   for( i=0 ; i<n_var ; i++ )
   {
      strcpy(nome[i]     ,"\0");
      strcpy(conteudo[i],"\0");
   }
   n=0;
   ini=0;

}
//#########################################################################
```

```
void variaveis::atribuir(char * nome_param , char * conteudo_param )
{
   strcpy( nome[n]     , nome_param );
   strcpy( conteudo[n] , conteudo_param );
   n++;

}
//#########################################################################

void variaveis::gravar()
{
   ini = n;
}
//#########################################################################

void variaveis::rejeitar()
{
   int i;
   for( i=ini ; i<n_var ; i++ )
   {
      strcpy(nome[i]    ,"\0");
      strcpy(conteudo[i],"\0");
   }
   n=ini;
}
//#########################################################################

void variaveis::emitir()
{
   completar();

   int i;
   char s[15];

   for (i=0 ; i < p[0] ; i++)
   {
      fputs("\n"         , arq_saida);
      fputs(nome[i]      , arq_saida);
      fputs(" = "        , arq_saida);
      fputs(conteudo[i] , arq_saida);
   }
   fputs("\n----------------------------------------\n",arq_saida);

   for (i=0; i<n_var ; i++ )
   {
      if (p[i]>=0)
      {
         fputs("\n"           , arq_saida);
         fputs(nome[p[i]]     , arq_saida);
         fputs(" = "          , arq_saida);
         fputs(conteudo[p[i]] , arq_saida);
      }
   }
   fputs("\n########################################\n",arq_saida);
}
//#########################################################################
```

```
void variaveis::completar()
{
   int i,
       clean =  0,
       pdata = -1,
       pnum  = -1,
       phora = -1,
       pinf  = 11;

   for (i=0; i<n_var ; i++ ) p[i]=-1;
   p[0] = n;

   for (i=0; i<n ; i++ )
      if (strcmp( nome[i] , "tipo" ) == 0 )
      {
         if (strcmp( conteudo[i] , "CEASING" ) == 0 )
            clean=1;
         break;
      }

   for (i=0; i<n ; i++ )
   {
      if (strcmp( nome[i] , "tendencia" ) == 0 )
      {
         p[7]=i;
         if (strcmp( conteudo[i] , "INCREASED" ) == 0 )
            strcpy( conteudo[i] , "more severe" );
         else
         if (strcmp( conteudo[i] , "DECREASED" ) == 0 )
            strcpy( conteudo[i] , "less severe" );
      }
      else
      if (strcmp( nome[i] , "severidade" ) == 0 )
      {
         p[4]=i;
         if ( clean ) strcpy(conteudo[i],"CL");

         Severidade.set_severidade_central( conteudo[i] );
         if ( Severidade.pesquisar() )
            strcpy( conteudo[i] , Severidade.get_severidade_tmn() );
      }
      else
      if ( strcmp( nome[i] , "alarme" ) == 0 )
      {

         Cod_Alarme.set_alarme( conteudo[i] );
         if ( Cod_Alarme.pesquisar() )
         {
            p[0] = n;
            strcpy( nome[n]      , "Codigo Alarme" );
            strcpy( conteudo[n] , Cod_Alarme.get_cod_alarme() );
            n++;
```

```
Tipo_Evento.set_codigo(Cod_Alarme.get_codigo_tipo_evento());
if ( Tipo_Evento.pesquisar() );
{
   p[1]=n;
   strcpy( nome[n]      , "Tipo Evento");
   strcpy( conteudo[n] , Tipo_Evento.get_descricao() );
   n++;
}

Alarme_X_Causa.set_cod_alarme(Cod_Alarme.get_cod_alarme());
if ( Alarme_X_Causa.pesquisar() )
{
   strcpy( nome[n]      , "Codigo Causa");
   strcpy( conteudo[n] , Alarme_X_Causa.get_cod_causa() );
   n++;
}

Causa_Prov.set_cod_causa( Alarme_X_Causa.get_cod_causa() );
if ( Causa_Prov.pesquisar() )
{
   p[2]=n;
   strcpy( nome [n]     , "Causa Provavel");
   strcpy( conteudo[n] , Causa_Prov.get_causa() );
   n++;
}

Proposta.set_cod_alarme( Cod_Alarme.get_cod_alarme() );
if ( Proposta.pesquisar() )
{
   p[9]=n;
   strcpy( nome[n]      , "Proposta");
   strcpy( conteudo[n] , Proposta.get_proposta() );
   n++;
}

ProblemaEspecifico.set_cod_alarme( Cod_Alarme.get_cod_alarme());
if ( ProblemaEspecifico.pesquisar() )
{
   p[3]=n;
   strcpy( nome[n]      , "Problema Especifico");
   if ( strlen( ProblemaEspecifico.get_cod_causa() ) > 0 )
   {
      strcpy( conteudo[n] , ProblemaEspecifico.get_cod_causa());
      strcat( conteudo[n] , "; ");
      strcat( conteudo[n] , ProblemaEspecifico.get_causa());
   }
   else
      strcpy( conteudo[n] , ProblemaEspecifico.get_object_id());
   n++;
}

TxtAdicional.set_cod_alarme( Cod_Alarme.get_cod_alarme());
if ( TxtAdicional.pesquisar() )
{
   p[10]=n;
   strcpy( nome[n]      , "Texto Adicional");
   strcpy( conteudo[n] , TxtAdicional.get_txt_adicional());
   n++;
}
```

```
            InfAdicional.set_cod_alarme( Cod_Alarme.get_cod_alarme() );
            if ( InfAdicional.pesquisar() )
            {
               p[11]=n;
               strcpy( nome[n]      , "Informacao Adicional");
               strcpy( conteudo[n] , InfAdicional.get_object_id() );
               strcat( conteudo[n] , "; " );
               if ( strcmp(InfAdicional.get_object_id(),"0") == 0 )
                  strcat( conteudo[n] , "false; " );
               else
                  strcat( conteudo[n] , "true; " );

               strcat( conteudo[n] , InfAdicional.get_informacao() );
               n++;
            }

         } // pesquisar cod_alarme.

      } // if nome[i]="alarme"
      else
      if ( strcmp( nome[i] , "nralarme" ) == 0 ) pnum = i;
      else
      if ( strcmp( nome[i] , "data" ) == 0 ) pdata=i;
      else
      if ( strcmp( nome[i] , "hora" ) == 0 ) phora=i;
      else
      if ( strcmp( nome[i] , "infadicional" ) == 0 ) p[ pinf++ ] = i;

   } // for

   BackUp.set_cod_alarme( Cod_Alarme.get_cod_alarme() );
   if ( BackUp.pesquisar() )
   {
      p[5]=n;
      strcpy( nome[n]      , "Backed-up Status");
      strcpy( conteudo[n] , "true" );
      n++;
      p[6]=n;
      strcpy( nome[n]      , "Back-up Object");
      strcpy( conteudo[n] , BackUp.get_objeto() );
      n++;
   }
   else
   {
      strcpy( nome[n]      , "Backed-up Status");
      strcpy( conteudo[n] , "false" );
      n++;
      strcpy( nome[n]      , "Back-up Object");
      strcpy( conteudo[n] , "\0" );
      n++;
   }
```

```
   if ( (pnum >= 0) && (pdata >= 0) )
   {
      p[8]=pnum;
      strcpy( nome[pnum] , "Identificador de Notificacao" );
      strcat( conteudo[pnum] , "-");
      strcat( conteudo[pnum] , conteudo[pdata]);
      if ( phora>=0)
      {
         strcat( conteudo[pnum] , "-");
         strcat( conteudo[pnum] , conteudo[phora]);
      }
   }
   strcpy( nome[n]     , "\0");
   strcpy( conteudo[n] , "\0");

}
//#########################################################################
```

tabela.cpp

```
#include <conio.h>
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class cursor;
//#########################################################################

class tabela
{
   protected:
      cursor     Cursor;
      connection *conn;

      char SQL[120];

      ub1 *pvar[5];
      sb4 var_len[5];
      int qvar;
```

```
      sb1   name_buf[20];
      sb2   db_type;
      sword len , dsize;
      int Resp;

      int   conectar(connection * conn_param );
      int   executar_SQL();
   public:
      ~tabela();
};
//#########################################################################

int tabela::conectar( connection *conn_param )
{
   conn = conn_param;
   if (Cursor.open( conn ))
   {
      Cursor.display_error(stderr);
      return (EXIT_FAILURE);
   }
return(0);
}
//#########################################################################

int tabela::executar_SQL()
{
   int i;

   // Parse do SQL
   if ( Cursor.parse( SQL ) )
   {
      Cursor.display_error( stderr );
      return(EXIT_FAILURE);
   }

   // descrever os campos.
   len = sizeof (name_buf);
   for ( i=1 ; i <= qvar; i++ )
      if ( Cursor.describe( i , (sb4 *) &var_len[i], &db_type,
                            name_buf, (sb4 *) &len, (sb4 *) &dsize,
                           (sb2 *) 0, (sb2 *) 0, (sb2 *) 0) )
      {
         Cursor.display_error( stderr );
         return(EXIT_FAILURE);
      }

   //define as variaveis de saida
   for ( i=1 ; i <= qvar ; i++ )
      if( Cursor.define_by_position( i , (ub1 *) pvar[i],
                                     var_len[i]+1,
                                     STRING_TYPE, -1 , (sb2 *) 0,
                                     (ub2 *) 0, (ub2 *) 0 ) )
      {
         Cursor.display_error( stderr );
         return(EXIT_FAILURE);
      }
```

```
   // executar o SQL.
   if ( Cursor.execute() )
   {
      Cursor.display_error( stderr );
      return(EXIT_FAILURE);
   }

   if ( Cursor.fetch() && (Cursor.get_error_code() != NO_DATA_FOUND ))
   {
      Cursor.display_error( stderr );
      return(EXIT_FAILURE);
   }

   if (Cursor.get_error_code() == NO_DATA_FOUND) Resp = 0;
   else                                          Resp = 1;

return(Resp);
}
//#########################################################################

tabela::~tabela()
{
   conn = (connection *)0;
   Cursor.close();
}
//#########################################################################
```

cod_alarme.cpp

```
#include <stdio.h>
□
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class cursor;
//#########################################################################
```

```
class codigo_alarme : tabela
{
   private:

      char central_ID[25];
      char alarme[133];
      char codigo_alarme[10];
      char codigo_tipo[5];

   public:
      int   inicializar( connection * conn_param ,
                         char * central_param);
      void  set_alarme( char * alarme_param );
      char* get_cod_alarme();
      char* get_codigo_tipo_evento();
      int   pesquisar();
};
//#########################################################################

int codigo_alarme::inicializar( connection *conn_param ,
                                char * central_ID_param )
{
   strcpy(central_ID , central_ID_param );
   strcpy(codigo_alarme,"\0");

   pvar[1] = codigo_alarme;
   pvar[2] = codigo_tipo;
   qvar = 2;

   conectar( conn_param );

return(0);
}
//#########################################################################

void  codigo_alarme::set_alarme( char * alarme_param )
{
   strcpy( alarme , alarme_param );
}

//#########################################################################

char* codigo_alarme::get_cod_alarme()
{
   return (char*) codigo_alarme;
}
//#########################################################################

char* codigo_alarme::get_codigo_tipo_evento()
{
   return (char*) codigo_tipo;
}
//#########################################################################
```

```
int codigo_alarme::pesquisar()
{
   strcpy( SQL ,"SELECT cod_alarme, cod_tipo FROM alarmes WHERE central='");
   strcat( SQL , central_ID);
   strcat( SQL , "' and alarme='");
   strcat( SQL , alarme );
   strcat( SQL , "'");

   executar_SQL();

return(Resp);
}
//#########################################################################
```

tipo_evento.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

//#########################################################################

class tipo_evento : tabela
{
   private:

      char codigo_tipo[5];
      char descricao[55];

   public:

      int inicializar(connection * conn_param );
      int pesquisar();
      void set_codigo( char * cod_tipo_param );
      char* get_descricao();
};
//#########################################################################
```

```
int tipo_evento::inicializar( connection *conn_param )
{
   conn = conn_param;
   strcpy(codigo_tipo,"\0");
   strcpy(descricao,"\0");

   pvar[1] = descricao;
   qvar = 1;

   conectar( conn_param );

return(0);
}
//#########################################################################

void tipo_evento::set_codigo( char * cod_tipo_param )
{
   strcpy( codigo_tipo , cod_tipo_param );
}

//#########################################################################

char* tipo_evento::get_descricao()
{
   return (char*) descricao;
}
//#########################################################################

int tipo_evento::pesquisar()
{
   strcpy( SQL ,"SELECT descricao FROM tipo_evento WHERE cod_tipo='");
   strcat( SQL , codigo_tipo);
   strcat( SQL , "'");

   executar_SQL();

return(Resp);
}
//#########################################################################
```

alarme_causa.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}
```

```
/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class cursor;
//########################################################################

class alarmeXcausa : tabela
{
   private:
      char central_ID[25];
      char cod_causa[10];
      char cod_alarme[10];

   public:

      int   inicializar(connection * conn_param , char * central_param);
      void  set_cod_alarme( char * cod_alarme_param );
      char* get_cod_causa();
      int   pesquisar();
};
//########################################################################

int alarmeXcausa::inicializar( connection *conn_param ,
                               char * central_ID_param )
{
   conn = conn_param;
   strcpy(central_ID , central_ID_param );
   strcpy(cod_causa,"\0");
   strcpy(cod_alarme,"\0");

   pvar[1] = cod_causa;
   qvar = 1;

   conectar( conn_param );

return(0);
}
//########################################################################

void alarmeXcausa::set_cod_alarme( char * cod_alarme_param )
{
   strcpy( cod_alarme , cod_alarme_param );
}

//########################################################################

char* alarmeXcausa::get_cod_causa()
{
   return (char*) cod_causa;
}
//########################################################################
```

```
int alarmeXcausa::pesquisar()
{
   strcpy( SQL ,"SELECT cod_causa FROM alarme_causa WHERE central='");
   strcat( SQL , central_ID);
   strcat( SQL , "' and cod_alarme='");
   strcat( SQL , cod_alarme );
   strcat( SQL , "'");

   executar_SQL();

return(Resp);
}
//#########################################################################
```

causa_prov.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class cursor;
//#########################################################################

class causa_prov : tabela
{
   private:
      char cod_causa[7];
      char causa[55];
      char complemento[405];

   public:

      int   inicializar(connection * conn_param );
      void  set_cod_causa( char * cod_causa );
      char* get_causa();
      char* get_complemento();
      int   pesquisar();

};
//#########################################################################
```

```
int causa_prov::inicializar( connection *conn_param )
{

   conn = conn_param;
   strcpy(causa,"\0");
   strcpy(complemento,"\0");

   pvar[1] = causa;
   pvar[2] = complemento;
   qvar = 2;

   conectar( conn_param );

return(0);
}
//#########################################################################

void causa_prov::set_cod_causa( char * cod_causa_param )
{
   strcpy( cod_causa , cod_causa_param );
}

//#########################################################################

char* causa_prov::get_causa()
{
   return (char*) causa;
}
//#########################################################################

char* causa_prov::get_complemento()
{
   return (char*) complemento;
}
//#########################################################################

int causa_prov::pesquisar()
{
   strcpy( SQL ,"SELECT causa,descricao FROM causaprov WHERE codigo='");
   strcat( SQL , cod_causa ); // coverter int em char
   strcat( SQL , "'");

   executar_SQL();

return(Resp);
}
//#########################################################################
```

problema_esp.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class cursor;
class causa_prov;
//###########################################################################

class problema_esp : tabela
{
   private:

      causa_prov  CausaProv;

      char central_ID[35];
      char cod_alarme[5];
      char object_id[63];
      char cod_causa[6];
      char causa[55];
      char descricao[405];

   public:

      int   inicializar(connection * conn_param ,
                        char * central_param);
      void  set_cod_alarme( char * cod_alarme_param );

      char* get_object_id();
            char* get_cod_causa();
            char* get_causa();
      char* get_descricao();

      int   pesquisar();

};
//###########################################################################
```

```
int problema_esp::inicializar( connection *conn_param ,
                               char * central_param)
{
   conn = conn_param;
   strcpy(central_ID,central_param);
   strcpy(cod_alarme,"\0");
   strcpy(object_id,"\0");
   strcpy(cod_causa,"\0");
   strcpy(causa,"\0");
   strcpy(descricao,"\0");

   pvar[1] = object_id;
   pvar[2] = cod_causa;
   qvar = 2;

   conectar( conn_param );

   CausaProv.inicializar( conn_param );

return(0);
}
//#########################################################################

void problema_esp::set_cod_alarme( char * cod_alarme_param )
{
   strcpy( cod_alarme , cod_alarme_param );
}

//#########################################################################

char* problema_esp::get_object_id()
{
   return (char*) object_id;
}
//#########################################################################

char* problema_esp::get_cod_causa()
{
   return (char*) cod_causa;
}
//#########################################################################

char* problema_esp::get_causa()
{
   return (char*) causa;
}
//#########################################################################

char* problema_esp::get_descricao()
{
   return (char*) descricao;
}
//#########################################################################
```

```
int problema_esp::pesquisar()
{

   strcpy( SQL , "SELECT object_id,cod_causa FROM prob_esp WHERE central='");
   strcat( SQL , central_ID );
   strcat( SQL , "' AND cod_alarme='");
   strcat( SQL , cod_alarme);
   strcat( SQL , "'");

   executar_SQL();

   if (Resp == 1 )
   {
      if ( strlen( cod_causa ) > 0 ) // Tem causa provavel!
      {
         CausaProv.set_cod_causa( cod_causa );
         if ( CausaProv.pesquisar() )
         {
            strcpy( causa , CausaProv.get_causa());
            strcpy( descricao , CausaProv.get_complemento());
         }
      }
   } // endif Resp = 1

return(Resp);
}
//#########################################################################
```

---

severidade.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class cursor;
//#########################################################################
```

```
class severidade : tabela
{
   private:

      char central_ID[20];
      char severidade_tmn[20];
      char severidade_central[20];

   public:

      int inicializar(connection * conn_param , char * central_param);
      int pesquisar();
      void set_severidade_central( char * sev_central_param );
      char* get_severidade_tmn();
};
//#########################################################################

int severidade::inicializar( connection *conn_param ,
                             char * central_param)
{
   conn = conn_param;
   strcpy(central_ID , central_param);
   strcpy(severidade_tmn,"\0");
   strcpy(severidade_central,"\0");

   pvar[1] = severidade_tmn;
   qvar = 1;

   conectar( conn_param );

return(0);
}
//#########################################################################

void severidade::set_severidade_central( char * sev_central_param )
{
   strcpy( severidade_central , sev_central_param );
}

//#########################################################################

char* severidade::get_severidade_tmn()
{
   return (char*) severidade_tmn;
}
//#########################################################################

int severidade::pesquisar()
{
   strcpy( SQL ,"SELECT sevtmn FROM severidade WHERE sevcentral='");
   strcat( SQL , severidade_central);
   strcat( SQL , "'");

   executar_SQL();

return(Resp);
}
//#########################################################################
```

backup.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class cursor;
//#########################################################################

class backup : tabela
{
   private:
      char central_ID[35];
      char cod_alarme[5];
      char descricao[257];

   public:

      int   inicializar(connection * conn_param ,
                        char * central_param);
      void  set_cod_alarme( char * cod_alarme_param );
      char* get_objeto();
      int   pesquisar();

};
//#########################################################################

int backup::inicializar( connection *conn_param ,
                         char * central_param)
{
   strcpy(central_ID,central_param);
   strcpy(cod_alarme,"\0");
   strcpy(descricao,"\0");

   pvar[1] = descricao;
   qvar = 1;

   conectar( conn_param );

return(0);
}
//#########################################################################
```

```
void backup::set_cod_alarme( char * cod_alarme_param )
{
   strcpy( cod_alarme , cod_alarme_param );
}

//#########################################################################

char* backup::get_objeto()
{
   return (char*) descricao;
}
//#########################################################################

int backup::pesquisar()
{
   strcpy( SQL ,"SELECT objeto FROM backup WHERE central='");
   strcat( SQL , central_ID );
   strcat( SQL , "' AND cod_alarme='");
   strcat( SQL , cod_alarme );
   strcat( SQL , "'");

   executar_SQL();

return(Resp);
}
//#########################################################################
```

proposta.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class cursor;
//#########################################################################
```

```
class proposta : tabela
{
   private:

      char central_ID[35];
      char cod_alarme[5];
      char proposta[50];
      char n_prop[3];

   public:

      int   inicializar(connection * conn_param ,
                        char * central_param);
      void  set_cod_alarme( char * cod_alarme_param );
      char* get_proposta();
      int   pesquisar();

};
//#########################################################################

int proposta::inicializar( connection *conn_param ,
                           char * central_param)
{
   conn = conn_param;
   strcpy(central_ID, central_param);
   strcpy(cod_alarme,"\0");
   strcpy(n_prop,"\0");

   pvar[1] = n_prop;
   qvar = 1;

   conectar( conn_param );

return(0);
}
//#########################################################################

void proposta::set_cod_alarme( char * cod_alarme_param )
{
   strcpy( cod_alarme , cod_alarme_param );
}

//#########################################################################

char* proposta::get_proposta()
{
   return (char*) proposta;
}
//#########################################################################

int proposta::pesquisar()
{
   strcpy( SQL ,"SELECT proposta FROM proposta WHERE central='");
   strcat( SQL , central_ID );
   strcat( SQL , "' AND cod_alarme='");
   strcat( SQL , cod_alarme );
   strcat( SQL , "'");

   executar_SQL();
```

```
   if ( n_prop[0] == '1')
      strcpy( proposta , "repair action required" );
   else
      strcpy( proposta , "no repair action required" );

return(Resp);
}
//#####################################################################
```

txt_adicional.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

//#####################################################################

class txt_adicional : tabela
{
   private:
      char central_ID[35];
      char cod_alarme[5];
      char descricao[257];

   public:

      int   inicializar(connection * conn_param ,
                        char * central_param);
      void  set_cod_alarme( char * cod_alarme_param );
      char* get_txt_adicional();
      int   pesquisar();

};
//#####################################################################
```

```
int txt_adicional::inicializar( connection *conn_param ,
                                char * central_param)
{
   conn = conn_param;
   strcpy(central_ID,central_param);
   strcpy(cod_alarme,"\0");
   strcpy(descricao,"\0");

   pvar[1] = descricao;
   qvar = 1;

   conectar( conn_param );

return(0);
}
//#########################################################################

void txt_adicional::set_cod_alarme( char * cod_alarme_param )
{
   strcpy( cod_alarme , cod_alarme_param );
}

//#########################################################################

char* txt_adicional::get_txt_adicional()
{
   return (char*) descricao;
}
//#########################################################################

int txt_adicional::pesquisar()
{
   strcpy( SQL ,"SELECT descricao FROM txtadicional WHERE central='");
   strcat( SQL , central_ID );
   strcat( SQL , "' AND cod_alarme='");
   strcat( SQL , cod_alarme );
   strcat( SQL , "'");

   executar_SQL();

return(Resp);
}
//#########################################################################
```

inf_adicional.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class cursor;
//########################################################################

class inf_adicional : tabela
{
   private:

      char central_ID[35];
      char cod_alarme[7];
      char object_id[63];
      char significancia[3];
      char informacao[255];

   public:

      int   inicializar(connection * conn_param ,
                        char * central_param);
      void  set_cod_alarme( char * cod_alarme_param );

      char* get_object_id();
      char* get_significancia();
      char* get_informacao();
      int   pesquisar();

};
//########################################################################
```

```
int inf_adicional::inicializar( connection *conn_param ,
                                char * central_param)
{
   conn = conn_param;
   strcpy(central_ID,central_param);
   strcpy(cod_alarme,"\0");
   strcpy(object_id,"\0");
   strcpy(significancia,"\0");
   strcpy(informacao,"\0");

   pvar[1] = object_id;
   pvar[2] = significancia;
   pvar[3] = informacao;
   qvar = 3;

   conectar( conn_param );

return(0);
}
//#########################################################################

void inf_adicional::set_cod_alarme( char * cod_alarme_param )
{
   strcpy( cod_alarme , cod_alarme_param );
}
//#########################################################################

char* inf_adicional::get_object_id()
{
   return (char*) object_id;
}
//#########################################################################

char* inf_adicional::get_significancia()
{
   return (char*) significancia;
}
//#########################################################################

char* inf_adicional::get_informacao()
{
   return (char*) informacao;
}
//#########################################################################

int inf_adicional::pesquisar()
{
   strcpy( SQL , "SELECT object_id,significancia,informacao ");
   strcat( SQL , "FROM inf_adicional WHERE central='");
   strcat( SQL , central_ID );
   strcat( SQL , "' AND cod_alarme='");
   strcat( SQL , cod_alarme );
   strcat( SQL , "'");

   executar_SQL();
return(Resp);
}
//#########################################################################
```

mascara.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class cursor;
//#########################################################################

class mascara : tabela
{
   private:

      char central_ID[35];
      char linha_ID[20], linha_ID_ant[20], linha[136];

   public:

      int inicializar(connection * conn_param ,
                      char * central_param);
      int proxima_linha( char * linha_param );
      int ultima_linha( char * linha_param );
      void inicializar_linha_ID();
      void set_linha_ID_anterior();

};
//#########################################################################

int mascara::inicializar( connection *conn_param ,
                          char * central_ID_param )
{
   strcpy(central_ID , central_ID_param );
   strcpy(linha_ID,"\0");
   strcpy(linha_ID_ant,"\0");

   pvar[1] = linha_ID;  // identificador da linha
   pvar[2] = linha;     // conteudo da linha
   qvar = 2;

   conectar( conn_param );
return(0);
}
//#########################################################################
```

```
int mascara::proxima_linha( char * linha_param )
{
   strcpy(linha_ID_ant , linha_ID );

   strcpy( SQL ,"SELECT linha,conteudo FROM mascara WHERE central='");
   strcat( SQL , central_ID);
   strcat( SQL , "' and linha>'");
   strcat( SQL , linha_ID );
   strcat( SQL , "'");

   executar_SQL();

   if (Resp) strcpy( linha_param , linha );
   else      strcpy( linha_param , "\0"  );

   int p=0;
   while ( linha_ID[p] != ' ' ) p++;
   linha_ID[p]='\0';

return(Resp);
}
//#########################################################################

int mascara::ultima_linha( char * linha_param )
{
   strcpy( SQL ,"SELECT linha,conteudo FROM mascara WHERE central='");
   strcat( SQL , central_ID);
   strcat( SQL , "' and linha='FIM'");

   executar_SQL();

   if (Resp) strcpy( linha_param , linha );
   else      strcpy( linha_param , "\0"  );

return(Resp);
} // ultima_linha
//#########################################################################

void mascara::inicializar_linha_ID()
{
   strcpy(linha_ID,"  ");
}
//#########################################################################

void mascara::set_linha_ID_anterior()
{
   strcpy(linha_ID,linha_ID_ant);
}
//#########################################################################
```

picture.cpp

```
#include <stdio.h>
#include <windows.h>

extern "C"
{
#include <oratypes.h>
#include <ociapr.h>
/* demo constants and structs */
#include <ocidem.h>
#include <ocidfn.h>
}

/* oparse flags */
#define  DEFER_PARSE        1
#define  NATIVE             1
#define  VERSION_7          2

class connection;
class cursor;
//########################################################################

class picture : tabela
{
   private:

      char central_ID[35];
      char variavel[35];
      char vpicture[68];

   public:
      int   inicializar(connection * conn_param ,
                        char * central_param);
      void  set_variavel( char * variavel_param );
      char* get_picture();
      int   pesquisar();

};
//########################################################################

int picture::inicializar( connection *conn_param ,
                          char * central_param)
{
   conn = conn_param;
   strcpy(central_ID,central_param);
   strcpy(variavel,"\0");
   strcpy(vpicture,"\0");

   pvar[1] = vpicture;
   qvar = 1;

   conectar( conn_param );

return(0);
}
//########################################################################
```

```
void picture::set_variavel( char * variavel_param )
{
   strcpy( variavel , variavel_param );
}

//#########################################################################

char* picture::get_picture()
{
   return (char*) vpicture;
}
//#########################################################################

int picture::pesquisar()
{
   strcpy( SQL ,"SELECT picture FROM picture WHERE central='");
   strcat( SQL , central_ID );
   strcat( SQL , "' AND variavel='");
   strcat( SQL , variavel );
   strcat( SQL , "'");

   executar_SQL();

return(Resp);
}
//#########################################################################
```

LISTA DE SIGLAS E ABREVIATURAS

Sigla	Descrição
APT	Equipamento de Comutação da Central AXE-10
APZ	Parte do Computador da Central AXE-10
ASN.1	Abstract Syntax Notation One
AXE-10	Sigla de Identificação da Central da Ericsson
BHCA	Busy Hour Call Attempts
CCITT	The International Telegraph and Telephone Consultative Committee
CCS	Subsistema de Canal Comum
CHS	Subsistema de Tarifação
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CP	Processador Central
CPA	Central Controlada por Programa Armazenado
CPS	Subsistema de Processamento Central
DCF	Data Communication Function
DCN	Data Communication Network
DCS	Subsistema de Comunicação de Dados
DN	Distinguished Name
EM	Módulo de Extensão
EMB	Barramento de Módulo de Extensão
FMS	Subsistema de Gerência de Arquivos
GDMO	Guideline for Definitions of Managed Objects
GSS	Subsistema de Seleção de Grupo
ITU-T	International Telecommunications Union - Telecommunications Standardization Sector
LLA	Logical Layered Architecture
MAS	Subsistema de Manutenção
MCF	Message Communication Function
MCS	Subsistema de Comunicação Homem-Máquina
MIB	Management Information Base
MIT	Management Information Tree
MO	Management Object
MTS	Subsistema de Telefonia Móvel
NE	Network Element
NMS	Subsistema de Gerenciamento de Rede
OMS	Subsistema de Operação e Manutenção
OPS	Subsistema de Operadora
OS	Operational System
OSI	Open System Interconnection

PCM	Pulse Code Modulation
QoS	Quality of Service
RDBMS	Sistema de Gerenciamento de Bases de Dados Relacionais
RDN	Relative Distinguished Name
RP	Processador Regional
RPB	Barramento de Processamento Regional
RPS	Subsistema de Processamento Regional
SP	Processador de Suporte
SPS	Subsistema de Processamento de Suporte
SSS	Subsistema de Comutação de Assinantes
SUS	Subsistema de Serviços de Assinantes
TCS	Subsistema de Controle de Tráfego
TMN	Telecommunications Management Network
TSS	Subsistema de Troncos e Sinalização