Dissertação de Mestrado

Uso de Recursos de Orientação a Objetos Para a Construção de Clientes e Servidores em Centrais Públicas de Comutação Telefônica

Universidade Estadual de Campinas
UNICAMP
FEEC - DCA

Aluno: **Austerli Nunes Vieira**

Orientador: **Prof. Dr. Manuel de Jesus Mendes**

Banca Examinadora:

- **Prof. Dr. Eleri Cardozo**
- **Prof. Dr. Ivan Luiz Marques Ricarte**
- **Prof. Dr. Juan Manuel Adán Coello**
- **Prof. Dr. Manuel de Jesus Mendes**

Campinas, 22/Agosto/2000

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

V673u

Vieira, Austerli Nunes
Uso de recursos de orientação a objetos para a construção de clientes e servidores em centrais públicas de comutação telefônica / Austerli Nunes Vieira.--Campinas, SP: [s.n.], 2000.

Orientador: Manuel de Jesus Mendes
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. JAVA (Linguagem de programação de computador). 2. CORBA (Arquitetura de computador). 3. Banco de dados orientado a objetos. 4. Interação homem-máquina. 5. Interfaces de usuário (Sistema de computador). 6. Software de comunicação. 7. Centrais telefônicas. I. Mendes, Manuel de Jesus. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Resumo

São apresentados aqui alguns conceitos, metodologias e tecnologias para a implementação de software para a interação entre operadores de centrais telefônicas de comutação e as bases de dados associadas às centrais telefônicas. São apresentados os detalhes associados à criação de enlaces de sinalização número 7: uma interface gráfica para a comunicação homem-máquina, um cliente e um servidor que usam a interface IDL da OMG e outro cliente e servidor que usam invocação remota de método em Java (RMI). São também apresentadas conclusões e sugestões para trabalhos posteriores.

Abstract

Here are presented some concepts, methodologies and technologies that are needed for implementing software for the interaction between the operators of the telephonic switching offices and the associated telephonic switching offices data bases. The details associated with the creation of number 7 signaling links are presented: a graphical user interface for the man-machine communication, a client and a server which use the OMG IDL and another client and server which use Java remote method invocation (RMI). The conclusions and the suggestions for future works are also presented.

Agradecimentos

Ao professor **Manuel de Jesus Mendes** por ter sido quem me deu a oportunidade de ter um orientador neste assunto, mesmo sabendo que eu somente me dedicava a este tema nas minhas horas vagas, fora do trabalho.

Ao professor **Eleri Cardozo**, que me ajudou de forma ressaltada a concluir este trabalho.

À **FEEC/DCA/UNICAMP** e a cada um dos seus professores que me deram a chance de concluir meus créditos como aluno especial.

À banca examinadora pela valiosa contribuição feita.

Aos meus colegas e chefes de trabalho que me ajudaram em vários momentos a poder me dedicar a este trabalho.

À minha esposa – **Elenice Haider Nunes Vieira** - e às minhas filhas - **Marina** e **Eliane Haider Nunes Vieira** - pela compreensão, apoio, paciência e incentivo.

A meu pai – **Nílson Nunes** - e à minha mãe – **Ébia Vieira Nunes** - que sempre me incentivaram ao estudo e à persistência.

A **Aquele** que me deu o ser. Sem Ele nada posso mas com Ele tudo é possível. A Ele graças e louvores por tudo que me dá e que por mim faz.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1 - Introdução

É necessário começar por uma apresentação geral do contexto da grande maioria das redes de comutação pública de telefonia no Brasil para se poder entender o problema a ser tratado, a motivação e os objetivos deste trabalho.

As centrais telefônicas são imensas bases de dados (BD) distribuídas, estruturadas de diferentes formas, naturezas, tecnologias e fornecedores, separadas por distâncias muito variáveis: um prédio, um bairro, uma cidade, um estado, etc. Uma operadora de telefonia tem que ter gerência sobre todo este contexto.

Estas BD são repletas de informações essenciais para as empresas operadoras do sistema telefônico. Embora já existam soluções no mercado para resolver a maioria ou mesmo todos os problemas que a seguir estão citados, estas informações ainda não são disponíveis para serem gerenciadas pelas empresas de telefonia de forma ampla, rápida, adequada, suficiente, amigável ou eficiente. Boa parte deste problema se deve ao custo das soluções que dão acesso a estas bases de dados de forma ampla e aberta.

Há diferentes necessidades de gerenciamento nas centrais telefônicas públicas: comutação, transmissão, energia, etc. Mesmo sem falar das necessidades da Rede Inteligente (que por si só já justificam uma abordagem única) e somente nos restringindo à comutação, há vários tipos de gerenciamento possíveis, hoje já amplamente estudados pelo ITU-T dentro da abordagem da TMN, em aspectos tais como gerenciamento de tráfego, configuração, falhas, desempenho, aprovisionamento, etc. No entanto, o alto custo da solução TMN dificulta a utilização em massa deste tipo de solução. Há hoje, no entanto, novos recursos na tecnologia da informação que poderiam ser usados de forma alternativa. Este trabalho vem propor uma outra forma para tratar muitos dos problemas abordados pela solução TMN do ITU-T sem, no entanto, excluí-la.
Para isto é importante considerar que:

- em cada tipo de gerenciamento há vários problemas técnicos distintos que precisam ser resolvidos através de diferentes abordagens;
- não é objetivo deste trabalho ter a pretensão de querer substituir a TMN mas tão somente propor soluções alternativas parciais dentro de uma forma alternativa de solução - que até mesmo pode vir a usar a TMN como um sistema herdado.

Alguns exemplos de tipos de aplicações são apresentadas a seguir. Note-se que, de uma forma genérica, podem se referir à manipulação de bases de dados distribuídas de diferentes centrais, de diferentes fornecedores e de diferentes tecnologias.

Obtenção de dados de desempenho de tráfego

Consiste em observar a qualidade do serviço para diferentes indicadores.
Vários indicadores são definidos pelos órgãos competentes (tal como o ITU-T, a ANATEL, etc.). São de grande utilidade para avaliar a qualidade do tráfego cursado nas redes telefônicas. Servem de referências para dimensionamentos, aplicação de penalidades para provedores de serviços telefônicos, etc. Levam em conta vários parâmetros tais como o tipo da sinalização usada (canal associado ou canal comum), as origens e os destinos do tráfego que se quer observar, etc. Exemplos de indicadores:

- número de chamadas perdidas em uma certa rota de escoamento de tráfego;

- número de chamadas completadas corretamente em uma certa rota de escoamento de tráfego;
- número de mensagens que foram recebidas em um certo enlace de sinalização N7;
- número de mensagens que foram enviadas por um certo enlace de sinalização N7.

Os conceitos de SN7 serão detalhados mais a frente.

Levantamento da árvore de encaminhamento da central

Consiste em mapear todos os prefixos que puderem ser discados por um assinante de uma certa central telefônica, registrando o tratamento que deve ser dado ao número discado, isto é, por exemplo, se será permitido ou impedido o escoamento do tráfego associado ao número que tiver sido discado, se será encaminhado a uma mensagem gravada, etc.
É um levantamento muito importante não só para perceber erros de encaminhamento na rede em instalação ou mesmo na já existente, como também em situações de modificações. Em ampliação, por exemplo, são feitas inclusões de novos prefixos ou de novos tipos de serviços no plano de numeração da rede telefônica existente que precisam ser conferidos exaustivamente. O trabalho fica bem mais fácil com o uso de recursos computacionais.

Obtenção de dados de tarifação

Em geral isto se refere a arquivos de dados formatados de formas diferentes para diferentes objetivos. Os dados podem estar armazenados em contadores de impulsos de medição do uso que estejam associados a cada assinante. Neste caso, cada contador é incrementado em função do número de vezes que o assinante conseguir completar uma chamada, do tempo da conversação e da distância entre os assinantes.
No caso da bilhetagem automática é usado, para cada chamada feita, um registro com os números dos assinantes envolvidos na chamada telefônica, discriminando quem originou a chamada, quem paga, quanto tempo falaram e quando, além de outras informações que servem para definir, dentre outras coisas, por exemplo, se a chamada se completou ou não e porquê.
A transferência destes dados para o centro de processamento de dados, responsável pelo faturamento na empresa de telefonia, em geral é feita através de um conjunto de soluções particulares, formatos específicos de equipamentos, tecnologias e fabricantes, adaptações que envolvem vários fornecedores de software e de equipamentos, várias tecnologias e vários protocolos. No fundo o que toda empresa operadora busca aqui é uma padronização simples que possa ser usada em todos os casos.

Mudar a configuração dos recursos da central telefônica

Pode ocorrer em função de vários tipos de necessidades, tais como necessidades de expansão, ocorrência de falhas, melhorias de desempenho, necessidades operacionais corriqueiras, etc.
Muitas destas necessidades podem ser feitas diretamente através de comandos homem-máquina. Exemplos: substituição de partes com problemas, criação de novas rotas de encaminhamento de tráfego, modificação de características funcionais de assinantes (inclusão de capacidade de conferência telefônica, direcionar chamadas a outro número, restrições de tráfego, etc.), etc. O problema aqui é que cada tecnologia tem sua forma própria de atuação, demandando equipamentos específicos e operadores treinados para cada tecnologia. É comum, num mesmo centro de controle de uma operadora de comutação

telefônica conviverem várias equipes e vários equipamentos lado a lado. O ponto chave é usar um mesmo equipamento com uma linguagem comum que possa atender a todas as tecnologias.

Sistemas Orientados a Soluções Específicas

Existe, nas centrais telefônicas, uma grande variedade de tais tipos de sistemas.
Em geral têm por objetivo permitir acesso a uma pequena parcela das informações dos bancos de dados das centrais de forma isolada, não padronizada e em geral simples e eficiente. Podem (mas não necessariamente) ser soluções herdadas de uma certa tecnologia, de um certo fabricante, de uma certa implementação, etc. As soluções herdadas ainda existem, seja por motivos históricos ou por motivos funcionais. Conseguir informações das bases de dados das centrais telefônicas, através de aplicações não padronizadas, leva a desenvolvimentos inflexíveis, restritos. Estes somente funcionam como sistemas dedicados ao objetivo único para o qual a solução foi estruturada. Em geral não permitem interação com outros sistemas correlacionados. Dentre outros fatores, a pressa e os custos conduzem, ainda hoje, à existência e à proliferação de soluções orientadas a soluções específicas. Adaptar as interfaces, mesmo quando possível, pode não valer a pena devido à evolução da tecnologia, à diversidade de fornecedores, à diversidade de versões de equipamentos de um mesmo fabricante, aos custos associados, aos benefícios advindos, etc. O ponto é: não haveria uma forma de reutilizar esses sistemas ?

Separação de Áreas de Atuação

Em uma primeira análise pode parecer que funções de gerenciamento de contextos diferentes (aqui chamados de domínios) não precisem interagir com as dos outros domínios. Assim, por exemplo, funções de tarifação aparentam não ter qualquer relação com as de encaminhamento ou com as de desempenho. Isto leva a definir domínios de gerenciamentos claros e específicos, - o que é bastante útil para estruturar e limitar o escopo da solução. No entanto, revendo a questão com mais cuidado, a título de exemplo, uma alta taxa de chamadas não completadas observada nos campos de detalhes da tarifação por bilhetagem automática pode levar a uma análise que conduza a várias causas, entre elas, por exemplo, o desempenho de tráfego ou o plano de encaminhamento presente na árvore de numeração de uma certa central. Isto, por sua vez, pode levar à necessidade de corrigir ou reconfigurar a configuração de uma certa central telefônica. Ou seja, a definição de domínios de gerenciamentos não deve excluir a troca de informações e a cooperação entre estes domínios. Este fato conduz ao estabelecimento de domínios claros e bem definidos e à necessidade de compatibilidade e cooperação entre eles. Neste ponto, os sistemas não padronizados, orientados a soluções específicas e independentes, em geral, apresentam grandes problemas de compatibilidade entre si. Isto é, o desenvolvimento de um sistema para um certo domínio deve prever a interação entre domínios diferentes. Como um modelo novo poderia dar apoio à solução destes problemas ?

Separação Funcional Hierárquica

As centrais telefônicas se interconectam umas com as outras e formam, inicialmente, uma rede que permite o fluxo do tráfego telefônico, por exemplo, dentro de um mesmo país, dentro de uma mesma região ou dentro de um mesmo município. As redes municipais são interligadas entre si através de redes estaduais, nacionais e internacionais. Além do gerenciamento de uma central isolada, como um elemento desta rede, há também a

necessidade de gerenciar a rede em si. Isto conduz à necessidade de estabelecer diferentes níveis de gerenciamento: elemento de rede, rede, serviço e negócio. Em cada um destes níveis, diferentes problemas devem ser abordados e, para a interação entre os níveis, deve haver uma interface comum que permita o fluxo das informações. Para tal são necessárias padronizações de vários tipos, que envolvam não só os protocolos mas também as bases de dados usados. Quando os protocolos ou as bases de dados usados são padronizados de formas diferentes são necessárias adaptações ou conversões. Criar um modelo a mais aqui não seria criar um novo problema ?

Heterogeneidade de Equipamentos

Os equipamentos instalados nas operadoras de centrais telefônicas são dos mais diferentes tipos. A heterogeneidade de equipamentos, usuários, fornecedores e tecnologias conduz ao uso de diferentes tipos de sistemas operacionais e linguagens de programação para construir diferentes tipos de software que possam acessar os diferentes recursos das centrais telefônicas. Hoje esta diversidade de recursos existe de forma isolada, em geral sem qualquer interação ou cooperação mútua. O que se observa é um conjunto de equipamentos diferentes, específicos por fornecedor, por tipo de central e freqüentemente por versão de equipamento. Os centros de controle regionais das operadoras telefônicas convivem com vários problemas advindos destes fatos: falta de interação ou cooperação entre equipamentos diferentes, especialização de operadores para cada tipo de equipamento e para cada tipo de linguagens de comandos homem-máquina (que são específicas para cada tipo de central), problemas de espaço físico, dimensionamento de energia e ar condicionado, etc.

Seria de muita valia para as operadoras utilizar aplicações que pudessem:

- ser executadas de forma independente do hardware e do sistema operacional usado;
- utilizar uma interface única de interação com os operadores que fosse independente do fabricante da central telefônica e mesmo até do equipamento de computação usado;
- obter as informações a partir dos diferentes tipos de equipamentos já existentes - HP, SUN, PCs, IBM, etc. - e principalmente de forma independente do equipamento.

Sem dúvida que essa abordagem seria de grande valia para as operadoras, porque permitiria usar os sistemas existentes de forma econômica, maciça e rápida. Não se trata somente da economia da aquisição de máquinas mais potentes para uma aplicação específica mas principalamente o uso daquelas que já existem de modo imediato por todos e de forma compatível com as diversas soluções já existentes. Mas isso é viável ?

Perfil dos Usuários da Informação

A maioria dos usuários das informações das bases de dados das centrais telefônicas não é necessariamente perita em informática, mas em telefonia. A informática é apenas uma ferramenta para executarem seus objetivos. Disponibilizar as informações para estes usuários de forma amigável pressupõe muitas vezes usar softwares poderosos, orientados a janelas e com recursos gráficos. Mais ainda, é sem dúvida um ponto desejado e importante poder substituir estes softwares por outros que apareçam posteriormente, mantendo o máximo de aproveitamento e de compatibilidade com o que já exista. Isto, no entanto, envolve tecnologias caras que, em geral, só raramente estão presentes nas centrais. Há como prover já de imediato interfaces amigáveis para tal usuário e garantir compatibilidade com a evolução, já que as soluções que virão através de softwares futuros certamente serão mais

poderosas e atuarão nas mais diversas áreas de interação com as bases de dados das centrais telefônicas?

Gerenciamento de Redes TMN Conformes com o ITU-T

Para resolver os problemas anteriormente citados, o ITU-T definiu há já algum tempo uma série de recomendações para gerenciamento de redes de telecomunicações. Dentre elas, várias se aplicam às necessidades do gerenciamento da comutação telefônica. Várias empresas operadoras têm investido maciçamente em sistemas baseados em tal modelo. O fato de o modelo do ITU-T ter sido estruturado há já algum tempo significa que, por um lado é já um modelo bastante depurado - e portanto bastante sólido. No entanto, significa também que não incorpora recursos tecnológicos mais recentes. É um modelo bastante abrangente e complexo e os produtos que o usam são em geral muito bons mas muito caros. O preço os torna nem tanto disponíveis e a complexidade os torna nem tanto flexíveis quanto seria desejável. Seria possível definir uma interface efetivamente aberta, abaixar o preço da solução e tornar a obtenção de soluções novas, que envolvessem desenvolvimentos no sistema, mais rápidas e mais flexíveis ? Quanto custaria tal solução? Quem a faria? Quanto tempo um usuário deveria esperar para ter uma solução eficiente e amigável para um problema ainda não contemplado pela sua plataforma de interação com as bases de dados das centrais telefônicas?

Resumo das Motivações

O que se pretende é propor uma solução alternativa de monitoração e/ou gerenciamento de centrais telefônicas de comutação pública, usando um modelo baseado em recursos mais recentes da informática. Ela não exclui soluções existentes mas tenta aproveitá-las como soluções herdadas. Tal alternativa considera os seguintes pontos:

1) imensas bases de dados distribuídas, estruturadas de forma diferente e com diferentes naturezas de informações;
2) vários níveis de gerenciamentos que precisam manter a compatibilidade entre si para interagirem e cooperarem mutuamente;
3) vários tipos de problemas específicos para cada tipo de gerenciamento;
4) coexistência com um ambiente heterogêneo e herdado;
5) diferentes tipos de centrais telefônicas;
6) vários fabricantes;
7) versões diferentes de produtos de um mesmo fabricante;
8) várias formas diferentes de estruturação e implementação das centrais telefônicas;
9) diferentes tipos de linguagens de comandos homem-máquina com sintaxes e semânticas independentes;
10) usuários peritos em telefonia mas não necessariamente em informática;
11) a maioria das implementações existentes em geral não são orientadas a objetos nem estruturadas para serem compatíveis ou reaproveitadas com componentes softwares melhores desenvolvidos no futuro;
12) existência de um modelo sólido de TMN do ITU-T que ganha adoção progressiva nas operadoras, que foi estruturado há já algum tempo, que é de implantação cara, que exige hardware e software poderosos e específicos, que é complexo e de modificação restrita a poucos especialistas e que ainda somente é acessível a uma minoria devido aos seus custos;

13) existência de uma barreira muito grande entre as necessidades imediatas ou efetivas do dia a dia dos peritos das áreas de telefonia e a disponibilidade de softwares específicos que os atendam de forma eficiente;
14) permitir acessos remotos e locais, discados e via rede, de forma transparente.

O propósito e a forma de condução do tema

O objetivo é propor um modelo que possa atingir um conjunto de usuários mais amplo, mais diretamente ligados com os problemas do dia a dia da operação e planejamento das centrais telefônicas, de forma a melhorar e agilizar a interação entre estes usuários e as bases de dados das centrais telefônicas. As idéias básicas são:

1) usar tecnologias mais recentes;
2) permitir o uso dos recursos de forma mais acessível que a hoje permitida por outras tecnologias;
3) permitir diversidade de hardware e sistemas operacionais;
4) usar uma interface gráfica de comunicação homem-máquina:
    - que seja amigável para o usuário,
    - que possa ser usada de forma independente do hardware e do sistema operacional que a hospede
    - e que possa ser usada independente da tecnologia usada nas centrais telefônicas;
5) agilizar a obtenção das soluções que dependem do acesso às informações disponíveis nas bases de dados das centrais telefônicas, tanto a nível de leitura como a nível de modificação destes dados;
6) tentar aproveitar ao máximo a infra-estrutura de equipamentos já existente nas operadoras.

O modelo foi testado sobre um protótipo simples e é feita uma proposta para sua adoção em outros contextos mais abrangentes.

A abrangência e a complexidade do assunto sugere:

1) um contexto mínimo para o protótipo que seja suficiente para as abordagens;
2) o uso de várias restrições para simplificar a abordagem;
3) um esforço para não particularizar a solução para um caso específico;
4) propor extrapolações para a solução de outros problemas;
5) propor modelos e temas que sirvam de ponto de partida para outros eventuais estudos.

A proposta feita não tem o objetivo de excluir outros padrões tais como aqueles envolvidos com a TMN do ITU-T mas reconhecê-los, aproveitá-los e conviver com eles.

Um grande problema com que se depara é a dificuldade de acesso às informações proprietárias de cada equipamento de cada fabricante. A única forma encontrada para contornar esse problema foi através de simulações e de proposições.

Uma implementação simples foi feita para consolidar conceitos e confirmar algumas idéias iniciais. Para tanto somente foram levados em consideração alguns poucos aspectos do manuseio das redes de sinalização Número 7 que seguem o padrão ITU-T. Qualquer

aplicação poderia ser usada como protótipo já que a idéia é somente testar se o modelo proposto seria de fato útil em um caso real típico.

Estrutura do trabalho

A idéia básica é verificar como viabilizar a criação de um protótipo simples que possa servir de base para o desenvolvimento de outras aplicações que envolvam a interação de um operador com as bases de dados das centrais telefônicas. Para isto se analisa uma aplicação específica simples que use um cliente e um servidor. O objetivo não é detalhar o servidor mas mostrar como criar o cliente e principalmente o que deve ser padronizado na interface entre os dois. Apenas rapidamente é mostrado como o servidor pode ser mapeado nas bases de dados das centrais porque isto envolve informações proprietárias dos fabricantes e também envolve vários tipos de mapeamentos, que precisam levar em conta o tipo de equipamento e o fabricante. Devido à heterogeneidade de usuários é de fundamental importância considerar recursos amigáveis no cliente para a comunicação homem-máquina.
Há muitos tipos de aplicações que poderiam ser consideradas como base para o desenvolvimento deste estudo (conforme abordado anteriormente em "*Alguns Tipos de Aplicações Possíveis*"). Foi escolhida uma aplicação que trata a criação de um enlace de sinalização número 7 em uma central telefônica. Dentro deste contexto é necessário criar um cliente que contenha uma interface gráfica com os recursos necessários para a interação com o operador. Este cliente deve interagir com um servidor remoto que, por sua vez, deve se encarregar de providenciar, junto à base de dados da central telefônica, as ações solicitadas pelo operador através da interface gráfica. O servidor a seguir deve notificar ao cliente sobre o resultado da solicitação feita.
Este modelo que será tratado será aqui denominado de **protótipo** de estudos.
Este protótipo deve ser tal que permita que as bases de dados de diferentes centrais telefônicas, de diferentes fabricantes e de diferentes tecnologias, possam ser acessadas de forma padronizada.
O modelo usado no protótipo deve servir de base para outras aplicações telefônicas. A implementação não pretende ser exaustiva porque só tem por objetivo ser um modelo para servir de base para implementações reais.
É imprescindível que tal solução não esteja restrita a um hardware ou a um software específico. No entanto, como soluções específicas podem ter desempenho melhor que soluções genéricas, são feitas duas implementações de interfaces entre o cliente e o servidor a fim de comparar o desempenho delas. Uma usa uma interface aberta (OMG-IDL) e outra uma interface específica (Java RMI).
O presente trabalho inclui o desenvolvimento de um software simplificado para o protótipo.
O operador inicia o cliente e o servidor remoto. O cliente inclui a interface gráfica de usuário. O operador interage com a interface gráfica do usuário para caracterizar o que deseja que seja executado pelo cliente. O servidor - que é acionado pelo cliente - interage com a base de dados da central telefônica para que esta possa executar a ação solicitada pelo cliente. O servidor relata ao cliente os resultados da ação na base de dados da central telefônica. O cliente relata à interface gráfica do usuário o resultado que lhe foi informado pelo servidor. A interface gráfica informa ao operador o resultado da ação solicitada. O ciclo se repete indefinidamente a critério do operador da interface gráfica.
O cliente interage com o servidor através de uma interface previamente definida. O servidor não atua efetivamente na base de dados de uma central real mas emula tal atuação ecoando o que lhe foi solicitado pelo cliente, dando sempre como resposta que a ação solicitada foi executada com sucesso. O objetivo do trabalho não é detalhar como fazer o mapeamento do

servidor na base de dados mas mostrar o que deve ser objeto de padronizações na interface entre o cliente e o servidor. Por isto, também do lado do cliente, mesmo considerando que a interface gráfica de interação com o operador possua quase todos parâmetros efetivamente necessários para um caso real, somente parte deles é usado para ser enviado de um lado para o outro através da interface entre o cliente e o servidor. A idéia que norteou as implementações feitas foi tão-somente verificar se os protótipos estavam funcionando corretamente, isto é, se as idéias básicas estavam sedimentadas, se estavam sendo passados corretamente os dados necessários entre o cliente e o servidor, independentemente da quantidade de parâmetro escolhidos, etc.
Os dois modelos de implementação são comparados e são feitas algumas medições para ambos. Estas medições têm tão-somente o objetivo de avaliar o tempo de reação do protótipo para os comandos do operador. A intenção é saber que tipos de computadores pessoais comumente existentes nas operadoras poderiam ser usados para tal protótipo. Não se trata de querer restringir a aplicação a máquinas de pequeno porte mas tão-somente tentar viabilizar o uso deste tipo de solução ao maior número de usuários possível. Em todo o desenvolvimento dos protótipos procura-se uma alternativa simples, que não dependa de infra-estrutura cara, de forma a poder ser ampla e rapidamente viável em um ambiente de operação de uma operadora de telecomunicações.

É um ponto importante construir algo que possa servir de base para verificar até que ponto o protótipo desenvolvido é efetivamente útil para uma operadora de telecomunicações.

A geração automática de código a partir de uma especificação formal (por exemplo em UML) não foi objeto de um estudo detalhado. Numa implementação real ela economiza tempo, facilita a interação em uma equipe, melhora a qualidade da solução, facilita todo o ciclo de vida do software, etc.

Embora seja necessário verificar vários aspectos relacionados com a adequação dos protocolos usados, do modelo de informação, da interface gráfica, etc., nem tudo é objeto deste trabalho. Os maiores enfoques do trabalho estão na interface gráfica de interação com usuários e na interface entre o cliente e o servidor. Outros pontos serão tratados sem muita ênfase ou somente serão considerados através de sugestões para trabalhos posteriores.

O enfoque deste trabalho está em como a interface deve encapsular os dados solicitados pelo usuário, através da interface gráfica de comunicação homem-máquina, em um formato intermediário que possa ser usado, com alguma adaptação, por diferentes equipamentos de diferentes fabricantes. Para isto é necessário modelar os objetos das centrais telefônicas para poder representá-los nesta interface. O objetivo deste trabalho é cuidar disto, de uma forma inicial que sirva de ponto de partida para estudos posteriores, porque é um tema muito extenso e que deve ser objeto de padronização. Para este objetivo a TMN do ITU-T já tem uma MIB bastante evoluída que pode ser usada como ponto de referência para levantar as necessidades de definição de objetos que precisam ser usados numa extensão da interface aqui proposta.

Como o tema aborda muitas áreas é preciso apresentar os conceitos envolvidos para se ter idéia do que está sendo abordado. Como as implementações dos protótipos feitas têm por objetivo a criação de ESN7, os conceitos associados a este tema precisam ser apresentados. Como um dos objetivos do trabalho é mostrar que é possível implementar os conceitos da TMN sem depender de uma arquitetura tão cara é necessário mostrar o que é a TMN para se

entender seus prós e contras, bem como os conceitos da TMN que devem ser aproveitados. Os conceitos básicos necessários estão apresentados no capítulo 2.
É preciso conhecer e entender os recursos de desenvolvimento de software atualmente existentes para viabilizar as implementações. Isto tem impacto direto na facilidade ou na dificuldade do desenvolvimento a ser feito. Por isto são apresentadas no Capitulo 3 as plataformas existentes para tal objetivo. O capítulo 4 trata da análise e do projeto do sistema usado como protótipo. Ali são dados detalhes de como foi construído. No capítulo 5 são apresentados os resultados experimentais feitos neste protótipo e no capítulo 6 são apresentadas as conclusões e sugestões associadas.

Capítulo 2 - Conceitos de Telefonia: SN7 e TMN

A criação de enlaces de sinalização número 7 (SN7) é um tema típico de gerenciamento de redes de telecomunicações. Atualmente o que de melhor estruturado existe nesta área, sob o ponto de vista telefônico, é o trabalho desenvolvido pelo grupo responsável pela padronização da TMN do ITU-T. No entanto este é um trabalho que foi desenvolvido com recursos de computação mais antigos e que hoje, sob o ponto de vista da tecnologia de informação que foi usada, apresenta uma série de desvantagens, como se verá a seguir. Assim, torna-se necessário apresentar os conceitos de sinalização número 7 e os conceitos de TMN para se entender a proposta feita na condução desta dissertação.

2.1 Conceitos de Sinalização Número 7

Para a interligação de centrais telefônicas, dentre as diversas soluções possíveis é usado um feixe PCM ("***Pulse Code Modulation***") - modulação por pulsos codificados - de 32 canais. Este feixe tem que conduzir, em instantes diferentes, informações de diversas naturezas, que são necessárias para gerenciar os diferentes estados pelo qual uma chamada telefônica passa: estabelecimento, conversação, tarifação, desligamento, etc. No estabelecimento da chamada é necessário, entre outras coisas, sinalizar a ocupação de um dos vários canais do PCM, enviar o número discado pelo assinante chamador, junto com o número do assinante que originou a chamada e a sua categoria (se, por exemplo é assinante comum ou telefônico público), etc. Quando o assinante chamado atende é preciso notificar isto à central de origem para que ela se coloque em uma condição que permita que os dois assinantes falem. A partir deste instante cabe ao feixe PCM transportar a voz durante a conversação. Para fins de tarifação é preciso conduzir os eventuais pulsos usados para este fim que sejam necessários enquanto a conversação durar. Ao finalizar a chamada é preciso conduzir sinais que identifiquem isto e que permitam liberar os recursos das centrais telefônicas e o canal do feixe PCM para ser usado para uma nova chamada.

Há vários protocolos usados para este objetivo. Por simplificação trataremos aqui somente a **sinalização por canal comum número 7** (SN7). Mesmo a SN7 pode assumir pelo menos dois padrões distintos, conforme tenha sido padronizada pelo ITU-T ou pela ANSI. A SN7 padronizada pelo ITU-T é usada na rede de telefonia **fixa** do Brasil. A padronizada pela ANSI é usada na rede de telefonia **móvel** do Brasil. Aqui somente será considerada a SN7 ITU-T. Para mais detalhes a respeito, dentre outras referências, ver as Ref. [91] a [99].

A sinalização ITU-T número7 foi inicialmente concebida para permitir o escoamento rápido de informações de estabelecimento, liberação e gerenciamento de chamadas telefônicas entre centrais telefônicas, usando um canal de 64Kbps de um feixe PCM de 2 Mbps.

A troca de informações é feita através de mensagens. A comunicação é full duplex e ponto-a-ponto.

A central que origina e a que recebe uma chamada recebem o nome de **ponto de sinalização** (PS ou em inglês, SP - "*Signaling Point*").

Cada PS é identificado por um endereço que recebe o nome de **código de ponto de sinalização** (CPS - ou em inglês, SPC).

O SPC é um número de 14 bits que muitas vezes é representado em binário, em hexadecimal ou em decimal. Nas redes de tráfego internacional estes 14 bits são separados em 3 partes, de 3, 8 e 3 bits, para identificarem a zona, a área / rede e o número do código do ponto de sinalização. Esta divisão é chamada de representação da rede internacional ou **3-8-**

3. O código da rede internacional do Brasil é zona 7, área / rede 048. São previstos 8 códigos de pontos de sinalização nesta rede internacional. Esta padronização está contida na Ref. [97].

Há no entanto a rede nacional brasileira em que o SPC também é separado em 3 partes, mas a parte mais significante, com 4 bits, representa o código nacional da região em que o SPC está associado, os 4 bits seguintes representam o código da sub-região em que o SPC está alocado e os 6 bits menos significantes identificam o número do SP na região. Esta representação é comumente chamada de representação da rede nacional ou **4-4-6**.

O SPC de quem origina a chamada é chamado de **código de ponto de origem** (OPC - "*Origination Point Code*") e o de quem a recebe, **código de ponto de destino** (DPC - "*Destination Point Code*").

O feixe PCM pode conter mais de um enlace de SN7 (ESN7) entre os 2 SPC aos quais está ligado.

Embora tanto a voz como a sinalização possam usar um mesmo feixe PCM isto não é obrigatório. O meio usado para a sinalização pode ser independente do meio usado para a voz. O mais usual é que um único ESN7, contido em um canal de 64 Kbps de um feixe de 2 Mbps transporte as mensagens de vários canais de voz contidos nos demais canais deste próprio feixe PCM bem como de vários outros. Daí o nome sinalização comum (a vários canais de voz). Seja para atender ao tráfego de vários canais de voz ou por motivos de segurança a falhas, em geral sempre é usado mais de um ESN7 entre 2 SPC.

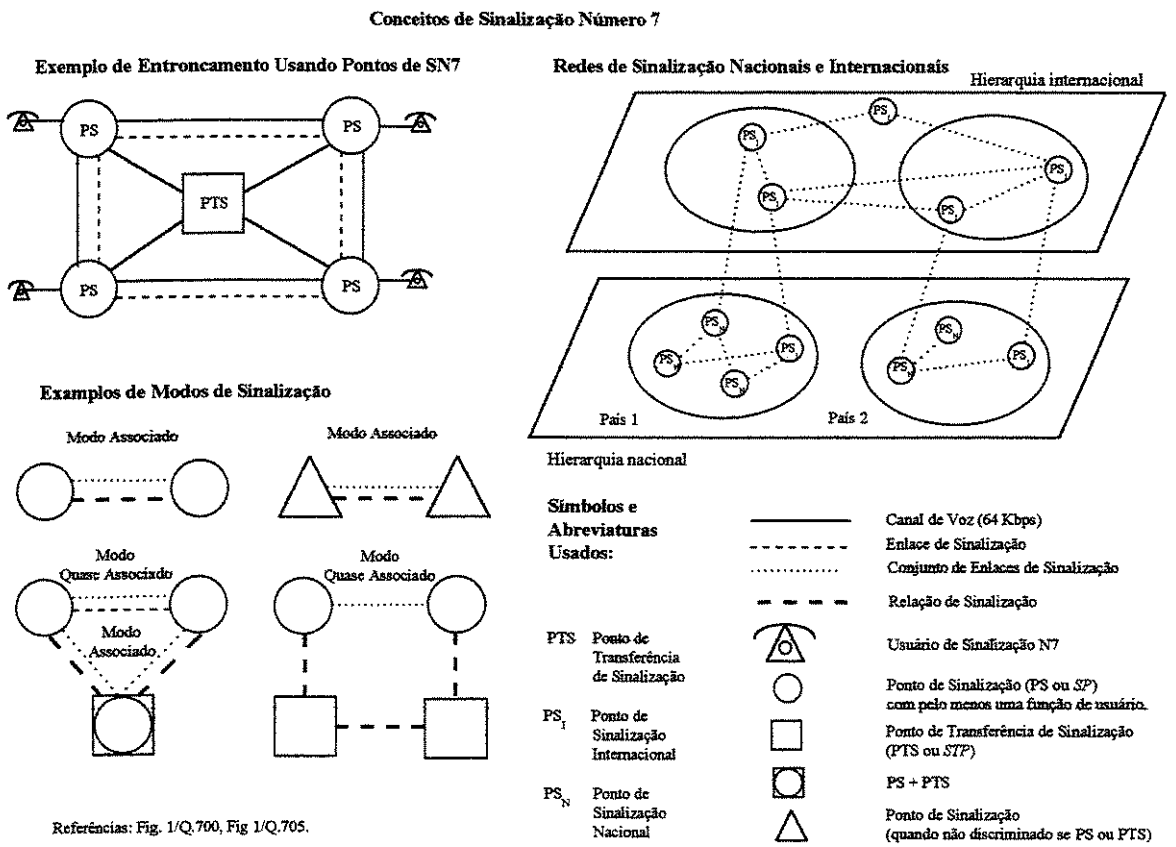


Figura # 2.1 – Conceitos de Sinalização N7

Como existem muitas centrais telefônicas, para evitar que haja um ESN7 entre cada uma delas, freqüentemente a comunicação entre dois SP depende de um ponto intermediário para

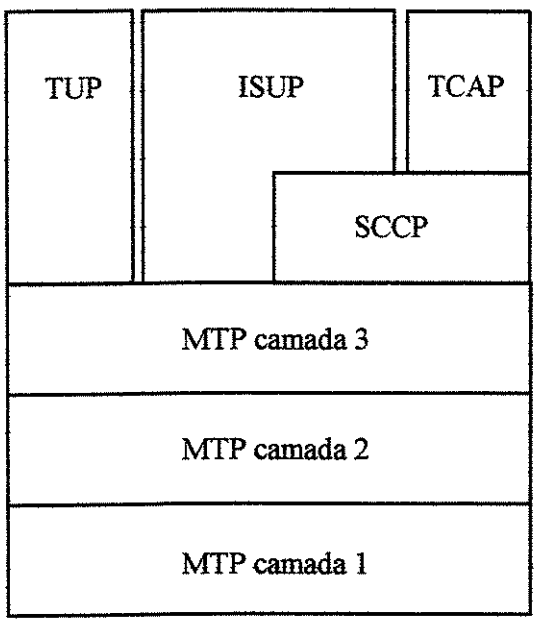
rotear as mensagens entre eles. Este ponto intermediário não origina nem recebe chamadas para si, mas somente desempenha a função de ponto de **transferência de sinalização** (PTS, ou STP em inglês) de um PS para outro PS ou para outro PTS. Entre 2 SPC pode haver até 16 ESN7. Eles são identificados pelo código do ESN7 (em inglês, **SLC**), um número de 4 bits.
A SN7 permite enlaces via satélites ou via terrestre. O tempo de resposta nos enlaces SN7 via satélite é muito maior que nos ESN7 via terrestre. Isto (entre outras coisas) acarreta a necessidade de configurar parâmetros diferentes para cada ESN7. São parâmetros de configuração do ESN7 (chamados **parâmetros da camada de nível 2** do protocolo da SN7). Cobrem aspectos tais como o tempo de espera para iniciar a retransmissão de uma mensagem que não tenha sido confirmada como recebida corretamente pelo seu destino.
Um conjunto de ESN7 que una um mesmo par OPC/DPC e com características (parâmetros) semelhantes forma uma **conjunto de ESN7** ("link set"). Um ou mais conjuntos de enlaces N7 forma uma **rota de sinalização N7** (RSN7).

A SN7 tem suas camadas próprias e segue um modelo diferente do modelo OSI. Ver figura 3.2. Os três primeiros níveis são denominados **MTP (Message Transfer Part)**. Os níveis físico (1) e de enlace (2) têm funções semelhantes aos níveis do modelo OSI. O nível 1 cuida dos aspectos físicos, elétricos e funcionais do enlace de sinalização. O nível 2 assegura a transmissão bidirecional da informação fim a fim em um enlace de sinalização: controle de fluxo, validação de mensagens, CRC. É usado CRC de 16 bits e as mensagens de nível de enlace são confirmadas pelo destino. O nível 3 é responsável pelo roteamento das mensagens, isto é, pelo encaminhamento das mensagens entre os pontos de sinalizações.
Logo acima da MTP estão as partes usuárias destas mensagens.

A **TUP (Telephonic User Part)** somente trata mensagens relativas a chamadas telefônicas.
A **DUP (Data User Part)** foi concebida para somente tratar dados mas não é usada no Brasil.
A **ISUP - "Integrated Service Digital Network (ISDN) User Part"** - trata tanto dados como voz. A tendência é que ela substitua tanto a DUP quanto a TUP.
Na MTP a troca de mensagens pressupõe a existência de um circuito de voz associado às mensagens que estão sendo trocadas, isto é, pressupõe um circuito de voz entre o assinante que origina e aquele que recebe a chamada.
Para prover, sobre a MTP, os serviços com e sem conexão do modelo OSI, foi criada uma camada usuária da MTP chamada **SCCP (Signaling Connection Control Part)**. Enquanto a MTP nível 3 endereça mensagens entre pontos de sinalização (OPC/DPC) a SCCP permite que as mensagens sejam endereçadas, dentro dos pontos de sinalização, a aplicações específicas que são chamadas de **subsistemas**, cada um deles identificados por um **número de subsistema (SSN)**. A partir de um número discado, contido em uma mensagem N7, a SCCP pode associar um DPC e um SSN. Isto é chamado de **tradução de título global (GTT - Global Title Translation).** O SCCP também serve como camada de transporte para a **TCAP (Transaction Capability Application Part)**.
A TCAP permite a troca de poucos dados em serviços sem conexão, em tempo real de processamento da chamada telefônica (serviços sensíveis a atrasos).
A MTP, a SCCP e a TCAP viabilizam a **Rede Inteligente**. São usadas, por exemplo, para encaminhar as chamadas do serviço 0800 de atendimento de clientes de uma certa empresa para o centro de atendimento mais adequado desta empresa, levando em conta a proximidade geográfica entre o chamador e o chamado, o dia da semana, o horário, etc. Quando se disca um número do tipo 0800, o número discado não é efetivamente o número

para o qual a chamada é completada. Há uma "tradução" do número discado para aquele ao qual a chamada efetivamente será completada. Esta tradução não é necessariamente feita na própria central que originou a chamada, mas num outro órgão específico que possui, dentre outras coisas, uma base de dados para estabelecer a correlação entre o número discado e o número ao qual a chamada deve ser encaminhada. Para isto há a necessidade de comunicação entre a central telefônica e o órgão responsável pelos acessos à base de dados da Rede Inteligente, sem que seja estabelecido um canal de voz para conversação entre estes 2 órgãos. Para isto a SCCP e a TCAP precisam ser usadas.

Principais Protocolos de Sinalização N7 - ITUT



Abreviaturas:

MTP - Parte de transferência de Mensagem
(*"Message Transfer Part"*)

TUP - Parte de Usuário Telefônico
(*"Telephonic User Part"*)

ISUP - Parte de Usuário De RDSI (Rede Digital de Serviços Integrados)
(*"ISDN - Integrated services Digital Network User Part"*)

SCCP - Parte de Controle de Conexão de Sinalização
(*"Signaling Connection Control Part"*)

TCAP - Parte de Aplicação de Capacidade de Transação
(*"Transaction Capability Aplication Part"*)

Referência: baseada na Fig. 4/Q.750

Figura # 2.2 – Principais Protocolos de SN7 - ITUT

2.2 *Conceitos de TMN*

2.2.1 Visão Geral

Para padronizar e suprir as necessidades de gerenciamento das redes de telecomunicações, o ITU-T propôs o uso de um conjunto de recomendações que constituem a solução TMN ("Telecommunications Management Network").
Uma visão geral da TMN do ITU-T pode ser vista na Ref. [79].
A TMN utiliza os princípios de gerenciamento de sistemas definidos pelo modelo OSI. Uma boa abordagem do modelo OSI pode ser visto na Ref. [153].
As recomendações abordam as mais variadas áreas de telecomunicações, incluindo arquiteturas, metodologias de especificação, gerenciamento de serviços, gerenciamento de funções, modelos de informações, protocolos de comunicação, terminologias, requisitos de conformidade, etc. Aplicam-se às mais diversas áreas das telecomunicações, incluindo os mais diversos componentes da rede.
O escopo da TMN ITU-T aborda aspectos relacionados a gerenciamentos de objetos, nomes e endereçamentos. Cobre os aspectos relativos aos gerenciamentos de tráfego, manutenção,

aprovisionamento e segurança. Padroniza os protocolos de comunicação, os serviços e as mensagens.
A arquitetura da TMN está descrita na Ref. [80]. Trata a arquitetura física, a arquitetura de fluxo de informação e a arquitetura funcional. Os blocos funcionais podem ser interligados entre si através de pontos de referências aos quais estão associadas interfaces padronizadas. As informações são modeladas com base em um modelo próprio orientado a objetos.
Usa as ferramentas orientadas a objetos desenvolvidas para o gerenciamento de sistemas OSI. O modelo de gerenciamento da informação está coberto pela Ref. [38]. As "diretrizes" para a definição dos objetos gerenciados (GDMO, "Guidelines for the Definition of Managed Objects") estão contidas na Ref. [40]. Para a codificação de PDUs é usada ASN.1 ("Abstract Syntax Notation One", notação de sintaxe abstrata, Ref. [23] e BER - "Basic Encoding Rules for ASN.1", regras de codificação básica para a ASN.1, Ref. [24]).

2.2.2 Principais Protocolos e Serviços Usados na TMN

São utilizados os protocolos de comunicação do modelo OSI, da RDSI e da sinalização N7. Para referências ao modelo OSI ver Ref. [22] e Ref.[25].
O uso do FTAM (ISO 8571) é previsto para a transferência de arquivos e, para o manuseio das informações de gerenciamento, o CMIP ("Common Management Information Protocol", protocolo de informação comum de gerenciamento, Ref. [36]).

A troca de informações entre os agentes e gerentes TMN é feita usando os serviços definidos pelo CMIS ("Common Management Information Service", serviço de informações comuns de gerenciamento, Ref. [35], e o protocolo associado, CMIP ("Common Management Information Protocol", protocolo de informações comuns de gerenciamento, Ref. [36]). Também podem ser usados outros ASEs ("*Application Service Element*", elemento de serviço da aplicação), como por exemplo o FTAM - que é usado para a transferência de arquivos.
Para o estabelecimento de uma associação com o outro lado usuário do CMIS (CMISE, "*Common Management Information Service Element*", elemento de serviço de informação comum de gerenciamento) é usado o serviço de estabelecimento de associação do ACSE (ver Ref.[28], serviço de controle de associação ("Association Control Service"). Aqui somente foi apresentada uma rápida visão dos serviços do CMIS e do protocolo associado, CMIP. Mais detalhes podem ser obtidos nas referências citadas acima.

ACSE e ROSE

A definição dos serviços e a definição do protocolo associado ao ROSE estão cobertas pela Ref. [29] - e pela Ref. [32]. O ROSE, por sua vez, pressupõe alguns serviços da camada de apresentação do modelo OSI que estão definidas na Ref. [27].

A definição dos serviços e a definição do protocolo associado ao ACSE estão cobertas pela Ref. [28] e pela Ref.[31].

Através do recurso de associações providos pelo ACSE é possível, dentre outras coisas:
- correlacionar uma tarefa da camada de aplicação a uma associação;
- correlacionar a associação da camada de aplicação com uma conexão da camada de apresentação e uma conexão da camada de sessão;

- identificar os objetos da camada de aplicação através de nomes (títulos).

Através do ROSE é possível a uma entidade da aplicação interagir de forma organizada com outra entidade localizada em um sistema remoto. As operações solicitadas são feitas através de uma associação estabelecida pelo ACSE. Podem ser solicitadas execuções de operações remotas e várias operações podem ser interligadas. Uma operação solicitada por uma entidade pode ser feita de uma forma síncrona ou assíncrona. Na forma síncrona só após ser recebida uma resposta para a operação solicitada que uma outra operação é solicitada. Na assíncrona podem ser feitas várias operações sem ter que esperar pelas respostas de cada uma. Podem ser notificados os erros e as rejeições ocorridas. É possível escolher entre sempre receber o resultado, só recebê-lo em caso de falha, só em caso de sucesso ou não recebê-lo em nenhum caso.

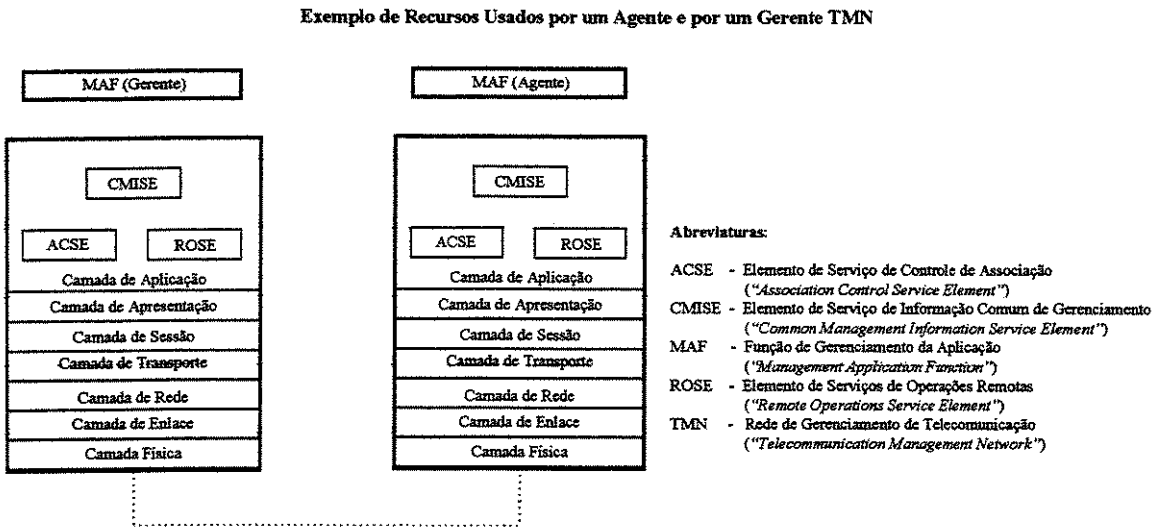


Figura # 2.3 – Exemplos de Recursos Usados por Um Agente e Um Gerente TMN

Mais detalhes sobre todos os níveis do modelo OSI podem ser vistos na Ref. [139] - na qual grande parte deste texto foi baseada.

CMIS ("Common Management Information Service", Serviço de Informações Comuns de Gerenciamento)

A Rec. X.710, Ref. [35], define as primitivas de serviço, os parâmetros e outras informações necessárias para a descrição de cada primitiva de serviço.

Os Serviços do CMIS

Sob um ponto de vista bastante simplificado os serviço de gerenciamento contidos no CMIS são desempenhados pelas seguintes ações elementares:

Ações	Serviço
obter o valor de um ou mais atributos	M_GET
cancelar a obtenção dos valores de um ou mais atributos	M_CANCEL_GET
escrever em um ou mais atributos	M_SET
solicitar a execução de uma ação remota	M_ACTION
criar uma instância de um objeto	M_CREATE
remover a instância de um objeto	M_DELETE
notificar um evento	M_EVENT_REPORT

Tabela # 2.1 – Os Serviços do CMIS

Os Parâmetros Associados aos Serviços do CMIS

Vários parâmetros são definidos para cada serviço, de forma a permitir que cada ação possa ser aperfeiçoada, limitada, especializada, estendida ou particularizada pelo acréscimo de características adicionais que mudem as características básicas da ação elementar. Em função do tipo de serviço, alguns parâmetros são opcionais e outros são obrigatórios.

Cada serviço está associado a **um identificador de invocação** único. É usado para diferenciar entre operações simultâneas do mesmo tipo de serviço.

O parâmetro **modo** permite definir se uma operação deve ou não ser confirmada.

Através do parâmetro **escopo** define-se que objetos (de um conjunto de objetos) devem ser abrangidos pela ação elementar.

Através do parâmetro **filtro** os objetos que serão alvo da ação elementar - selecionados pelo **escopo** - são restringidos a um subconjunto ainda menor, definidos por regras de seleção baseadas em operações lógicas.

O parâmetro "**base**": especifica a classe do objeto gerenciado que será usada como ponto de partida para a seleção dos objetos gerenciados sobre os quais o filtro deverá ser aplicado quando estiver sendo usado.

Como uma única ação elementar pode provocar a emissão de várias respostas, em função da conveniência, as diferentes respostas recebidas podem ter políticas de tratamento diferenciado, podendo - ou não - haver interligação de respostas múltiplas. Para que as respostas múltiplas sejam relacionadas com as ações elementares iniciais é necessário adotar um identificador da ação elementar inicial que provocou as respostas múltiplas.
Uma ação elementar inicial pode desencadear várias ações secundárias. Pode ocorrer que nem todas ações secundárias sejam executadas plenamente. Neste caso é necessário definir a política a ser adotada para a **sincronização** destas ações:

- melhor esforço ou
- ação atômica.

Se a opção é de **melhor esforço**, não haverá controle das ações como um todo. Cada ação secundária é tratada separadamente como se fosse uma ação distinta.

Se a ação é **atômica** ela deve ser tratada de forma única, sem subdivisão. Somente deve ser gerado uma informação de inexistência de erro se todas ações selecionadas pelo filtro, dentro do escopo de ações, forem executadas sem erro. Mesmo que somente uma única ação secundária não possa ser executada deve ser gerada uma informação de erro de execução da ação elementar solicitada.

Também são usados parâmetros para definir a **classe do objeto gerenciado**, a **instância do objeto gerenciado**, a identificação do **instante da geração de um evento**, o **tipo do evento**, **informações relativas ao evento** considerado, **erros**, **lista de atributos**, **ação a ser executada** no objeto, **resposta à ação** executada, etc.

O parâmetro **controle de acesso** é previsto como "entrada para acessar funções de controle"; não são fornecidos mais detalhes.

Em operações que envolvam respostas múltiplas deve ser provido o parâmetro **identificador de "ligação de respostas múltiplas"** ("**linked identifier**"); deve conter o mesmo **identificador de invocação** que estiver associado à operação em execução.

Para o serviço M_SET (escrever em um ou mais atributos) é necessário um parâmetro para definir a **lista de modificação** a ser feita: **identificador do atributo** a ser mudado, **valor do atributo** e o **operador do atributo**. Este operador pode ser: **trocar** o atributo, **adicionar valores** ao atributo, **remover** o atributo e **escrever** um valor no atributo.

Como o uso de respostas múltiplas pode acarretar uma grande quantidade de informação de retorno, o serviço M_CANCEL_GET é usado para cancelar a obtenção dos valores de um ou mais atributos. Não são apresentados outros mecanismos para o controle do fluxo de dados.
Com exceção da ação de cancelamento, as demais ações da tabela anterior estão associadas à unidade funcional "kernel" (núcleo) do CMIS.

As tabelas abaixo mostram os parâmetros associados a cada serviço do CMIS.

Abreviaturas usadas:

M	obrigatório - ("*mandatory*")	**Req**	Requisição
U	opção do usuário	**Rsp**	Resposta
C	pode estar presente ou não em função das condições usadas	**Ind.**	Indicação
I	mesmo valor da coluna da Req/Ind - ("igual")	**Conf**	Confirmação
-	não usado	**Id.**	Identificador

Serviço →	M_GET	
Parâmetros ↓	Req /Ind	Rsp / Conf
id. invocação	M	M
id. ligação	-	C
classe da base	M	-
instância da base	M	-
escopo	U	-
filtro	U	-
controle de acesso	U	-
sincronização	U	-
lista de identificadores de atributo	U	-
classe objeto	-	C
instância objeto	-	C
horário da resposta	-	U
lista de atributos	-	C
erros	-	C

Tabela # 2.2 – Parâmetros do Serviço M_GET do CMIS

Serviço →	M_CANCEL_GET	
Parâmetros ↓	Req / Ind	Rsp / Conf
id. invocação	M	M, I
id. da operação de invocação M_Get associada à M_CANCEL_GET	M	-
erros	-	C

Tabela # 2.3 - Parâmetros do Serviço M_CANCEL_GET do CMIS

Serviço →	M_SET	
Parâmetros ↓	Req /Ind	Rsp / Conf
identificador da invocação	M	M
identificador da ligação	-	C
modo	M	-
classe do objeto base	M	-
instância do objeto base	M	-
escopo	U	-
filtro	U	-
controle de acesso	U	-
sincronização	U	-
classe objeto gerenciado	-	C
instância objeto gerenciado	-	C
lista de modificação	M	-
lista de atributos	-	U
horário	-	U
erros	-	C

Tabela # 2.4 - Parâmetros do Serviço M_SET do CMIS

Serviço →	M_ACTION	
Parâmetros ↓	Req /Ind	Rsp / Conf
identificador da invocação	M	M
identificador da ligação	-	C
modo	M	-
classe do objeto base	M	-
instância do objeto base	M	-
escopo	U	-
filtro	U	-
classe objeto gerenciado	-	C
instância objeto gerenciado	-	C
controle de acesso	U	-
sincronização	U	-
tipo de ação	M	C, I
informação relativa à ação	U	-
horário da resposta	-	U
resposta á ação	-	C
erros	-	C

Tabela # 2.5 - Parâmetros do Serviço M_ACTION do CMIS

Serviço →	M_CREATE	
Parâmetros ↓	Req /Ind	Rsp / Conf
identificador da invocação	M	M, I
classe objeto gerenciado	M	U
instância objeto gerenciado	U	C
instância do objeto superior	U	-
controle de acesso	U	-
instância do objeto de referência	U	-
lista de atributos	U	C
horário da resposta	-	U
erros	-	C

Tabela # 2.6 - Parâmetros do Serviço M_CREATE do CMIS

Serviço →	M_DELETE	
Parâmetros ↓	Req /Ind	Rsp / Conf
identificador da invocação	M	M
identificador da ligação	-	C
classe do objeto base	M	-
instância do objeto base	M	-
escopo	U	-
filtro	U	-
controle de acesso	U	-
sincronização	U	-
classe objeto gerenciado	-	C
instância objeto gerenciado	-	C
horário da resposta	-	U
erros	-	C

Tabela # 2.7 - Parâmetros do Serviço M_DELETE do CMIS

Serviço →	M_EVENT_REPORT	
Parâmetros ↓	**Req /Ind**	**Rsp / Conf**
identificador da invocação	M	M, I
modo	M	-
classe do objeto gerenciado	M	U
instância do objeto gerenciado	M	U
tipo de evento	M	C, I
horário do evento	U	-
informação do evento	U	-
horário da resposta	-	U
resposta ao relatório de evento	-	C
erros	-	C

Tabela # 2.8 - Parâmetros do Serviço M_EVENT_REPORT do CMIS

Note-se que o CMIS embute na lógica dos serviços que serão providos pelo protocolo associado, o CMIP, não só a transferência da informação em si como também todo um rebuscado conjunto de parâmetros para a seleção da natureza da informação a ser transferida entre as partes envolvidas.

CMIP ("Common Management Information Protocol", Protocolo de Informações Comuns de Gerenciamento)

CMIP é o protocolo que suporta os serviços definidos no CMIS. A Ref. [36] define o protocolo CMIP.

Para seu funcionamento o CMIP necessita de alguns serviços que são providos pelo ROSE ("Remote Operations Service Element" - Elemento de Serviço de Operações Remotas) e pelo ACSE ("Association Control Service Element" - Elemento de Serviço de Controle de Associação).

As Operações do CMIP

As operações do CMIP estão associadas às primitivas CMISE conforme tabela abaixo. Esta tabela foi extraída da Tabela 4 da Ref. [36].
Nesta tabela estão também identificados os casos em que é feito uso do parâmetro identificador de "ligação de respostas múltiplas" ("linked identifier").

Primitivas CMIS	Modo	Linked-ID	Operações CMIS
M_CANCEL_GET req/ind	Confirmado	Não se aplica	m-Cancel-Get-Confirmed
M_CANCEL_GET rsp/conf	Não se aplica	Não se aplica	m-Cancel-Get-Confirmed
M_EVENT_REPORT req/ind	Não Confirmado	Não se aplica	m-EventReport
M_EVENT_REPORT req/ind	Confirmado	Não se aplica	m-EventReport-Confirmed
M_EVENT_REPORT rsp/conf	Não se aplica	Não se aplica	m-EventReport-Confirmed
M_GET req/ind	Confirmado	Não se aplica	m-Get
M_GET rsp/conf	Não se aplica	Ausente	m-Get
M_GET rsp/conf	Não se aplica	Presente	m-Linked-Reply
M_SET req/ind	Não Confirmado	Não se aplica	m-Set
M_SET req/ind	Confirmado	Não se aplica	m-Set-Confirmed
M_SET rsp/conf	Não se aplica	Ausente	m-Set-Confirmed
M_SET rsp/conf	Não se aplica	Presente	m-Linked-Reply
M_ACTION req/ind	Não Confirmado	Não se aplica	m-Action
M_ACTION req/ind	Confirmado	Não se aplica	m-Action-confirmed
M_ACTION rsp/conf	Não se aplica	Ausente	m-Action-confirmed
M_ACTION rsp/conf	Não se aplica	Presente	m-Linked-Reply
M_CREATE req/ind	Confirmado	Não se aplica	m-Create
M_CREATE rsp/conf	Não se aplica	Não se aplica	m-Create
M_DELETE req/ind	Confirmado	Não se aplica	m-Delete
M_DELETE rsp/conf	Não se aplica	Ausente	m-Delete
M_DELETE rsp/conf	Não se aplica	Presente	m-Linked-Reply

Ref.: CCITT-X.711 - Tabela 4

Tabela # 2.9 – Operações CMIS

2.2.3 A arquitetura funcional

Os serviços previstos estão discutidos na Ref. [81] e na Ref. [19]. Aqui estão alguns deste serviços: administração de encaminhamento e de análise de dígitos, administração de medidas e análises de tráfego, gerenciamento de tráfego, administração da tarifação, gerenciamento da segurança da TMN, gerenciamento da comutação, administração da qualidade de serviço, administração do desempenho de rede, gerenciamento da rede de sinalização número 7, gerenciamento da rede inteligente, etc.

Os seguintes blocos funcionais são previstos: OSF, NEF, WSF, MF e QAF.

O bloco **OSF** (*"Operations Systems Function"*, função de sistemas de operações) processa as informações relacionadas ao gerenciamento de telecomunicações visando a monitoração, a coordenação e o controle das funções de gerenciamento.

O bloco **NEF** (*"Network Element Function"*, função de gerenciamento da rede) é o bloco que provê as funções da rede de telecomunicações. Não é a TMN mas é o bloco que é gerenciado pela TMN.

O bloco **WSF** (*"Work Station Function"*, função de estação de trabalho) é o bloco que proporciona os meios de interação da TMN com o ser humano.

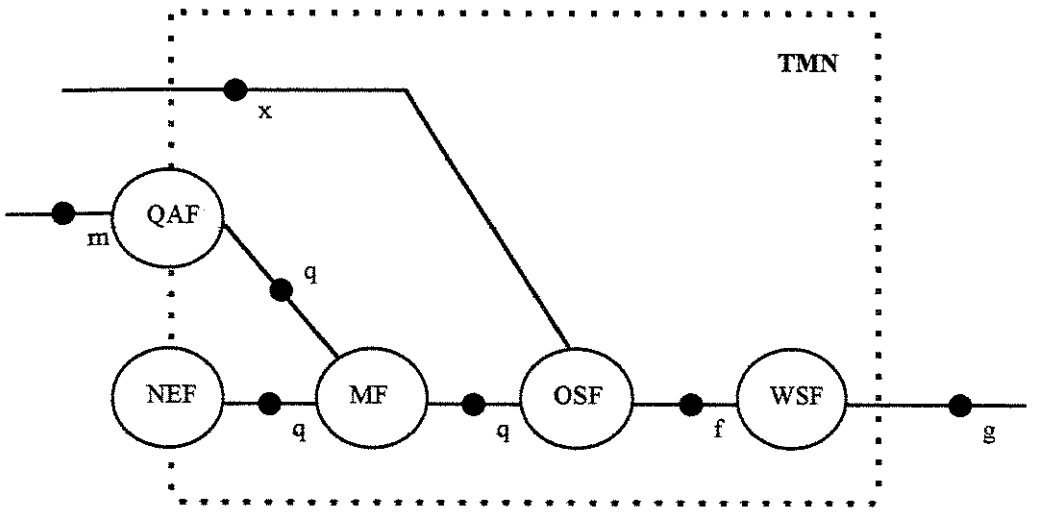
O bloco **MF** (*"Mediation Function"*, função de mediação) é o bloco que garante a uniformidade da representação da informação entre os blocos do ambiente TMN. Para isto, e se necessário, o MF terá que adaptar, filtrar, armazenar e converter as informações de forma a compatibilizar a uniformidade da representação da informação dentre os blocos TMN.

O bloco **QAF** (*"Q Adaptor Function"*, função de adaptação para a interface Q) é o bloco usado para conectar as parte da TMN com outras partes externas à TMN (dos tipos NEF e OSF) e com os quais os blocos da TMN tenham que interagir. A função básica do QAF é prover a adaptação entre o ambiente TMN e o ambiente externo à TMN. A referência à interface "Q" ficará mais clara após a definição dos pontos de referências TMN, a seguir.

A função de comunicação de dados da TMN (**DCF** - *"Data Communication Function"*) é usada para a troca de informação entre os blocos de funções TMN. A DCF provê as camadas 1 a 3 do modelo OSI ou seu equivalente no ITU-T.

Os blocos OSF, NEF, WSF, MF e QAF interagem entre si através de **pontos de referência**. Estes pontos são definidos a partir de 5 **classes de pontos de referência**: q, f, x, g e m. Estas classes podem ser vistas na figura abaixo, baseada na Ref. [80] - Figura 3/M.3010.

Classes de Pontos de Referência na TMN



Referência: Figura 3/M3010

Figura # 2.4 – Classes de Pontos de Referência na TMN

Classes de pontos de referência:

- q: entre OSF e MF, QAF e MF, MF e NEF
- f: entre OSF e uma WSF
- x: entre OSFs de 2 TMNs ou entre OSF de uma TMN e a funcionalidade equivalente à OSF em outra rede
- g: entre uma WSF e os usuários
- m: entre uma QAF e entidades não TMN gerenciadas.

Para permitir a interação entre os diversos tipos de blocos funcionais, são definidas várias interfaces entre estes blocos. Estas interfaces são identificadas através de vários pontos de referências, conforme pode ser visto na Figura 3.9, baseada na Ref. [80] - Figura 6/M.3010. Algumas interfaces estão bastante detalhadas, outras ainda estão em estudos.

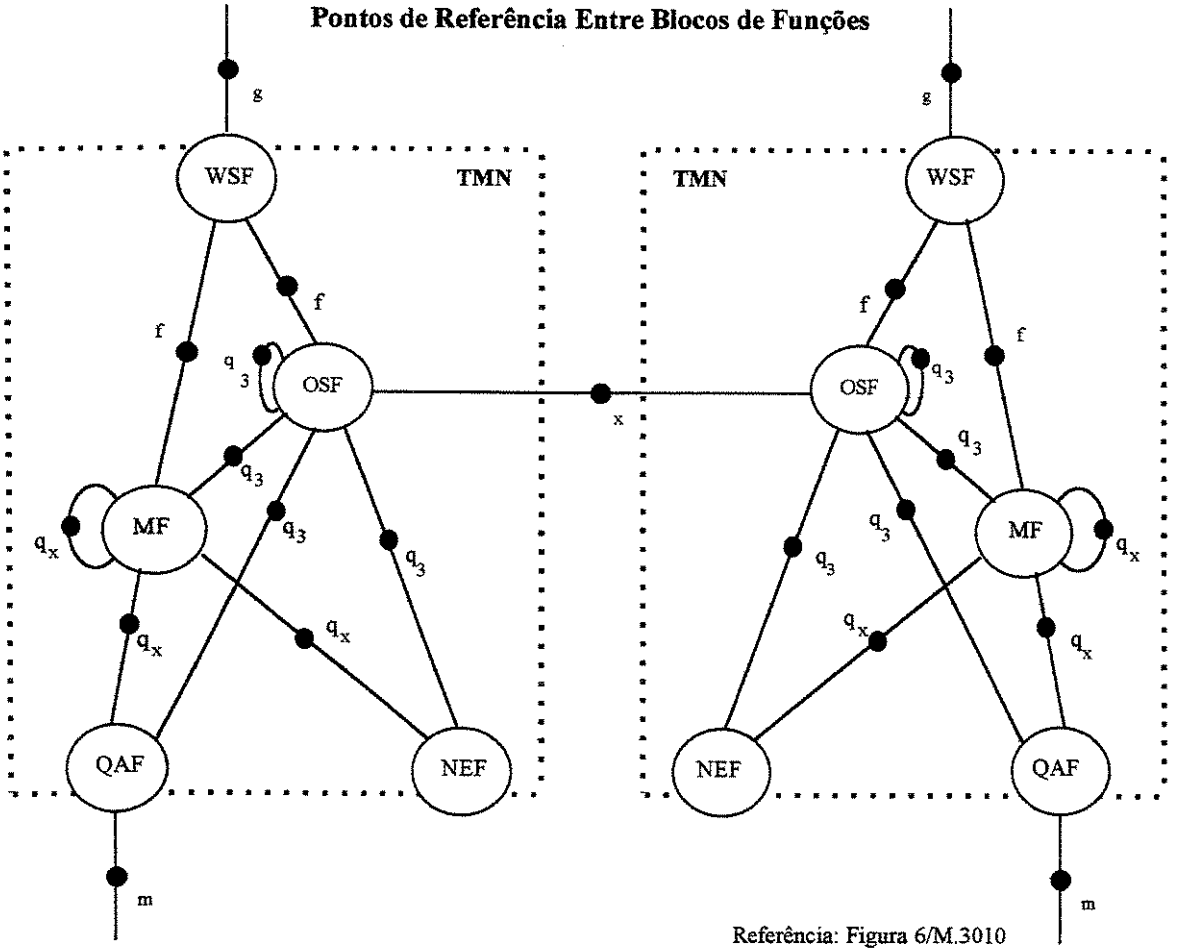


Figura # 2.5 – Pontos de Referência entre Blocos de Funções

NOTA:

A escolha dos protocolos relativos à Qx são deixados para escolha das operadoras. Os protocolos relativos às interfaces f e x não estão definidos. Os protocolos usados na interface Q3 (família **Q3**) estão especificados na Ref. [83] - Q.811, camadas 1 a 3) - e na Ref. [84] - Q.812, camadas 4 a 7). Os requisitos necessários para a interface f são definidos na Ref. [20] - "TMN Management Capabilities Presented at the F Interface", capacidades de gerenciamento TMN apresentadas na interface F.

Os blocos funcionais podem ser formados a partir de **componentes funcionais:** MAF, ICF, WSSF, UISF, MCF, DSF, DAF, SF. Estes componentes funcionais são identificados na Ref. [80].

MAF ("*Management Application Function*", Função de Gerenciamento da Aplicação): é caracterizada pelo tipo de bloco funcional em que está contida: MF, NEF, OSF, NEF e QAF. É parte da funcionalidade de um ou mais serviços de gerenciamento definidos para a TMN. Permite suportar tanto o papel de agente como o de gerente, conforme aplicável.

ICF ("*Information Conversion Function*", Função de Conversão da Aplicação): provê mecanismos de tradução entre modelos de informação. Está presente no MF e no QAF.

WSSF (*"Workstation Support Function"*, Função de Suporte de Estação de Trabalho): inclui suporte para acesso e manipulação de dados, invocação e confirmação de ações, etc.

UISF (*"User Interface Support Function"*, Função de Suporte de Interface de Usuário): traduz o modelo de informação usado pela TMN em um formato de comunicação homem-máquina.

MCF (*"Message Communication Function"*, Função de Comunicação de Mensagem): é composta de uma pilha de protocolos usada para troca de mensagens entre entidades pares do modelo.

DSF (*"Directory System Function"*, Função de Diretório de Sistema): armazena os objetos usados em conjuntos de objetos devidamente identificados e hierarquicamente ordenados.

DAF (*"Directory Access Function"*, Função de Acesso de Diretório): usado para acessar e manter a informação relacionadas à TMN: ler, listar, buscar, adicionar, remover, modificar.

SF (*"Security Function"*, Função de Segurança): provê a política de segurança necessária às diferentes necessidades operacionais dos usuários.

2.2.4 A arquitetura da Informação TMN

Utiliza os conceitos de agentes e de gerentes usado no modelo OSI. (Ref. [34], *"OSI - Systems Management Overview"*, visão geral de gerenciamento de sistemas OSI).
O **papel de gerente** de um sistema distribuído é o da entidade que envia diretivas de operação de gerenciamento e recebe as notificações advindas do outro lado.
O **papel de agente** está associado à parte dos processos de aplicação que gerencia os objetos gerenciados. É responsável por responder às diretivas enviadas pelo gerente, enviar para o gerente uma visão destes objetos e emitir notificações que reflitam o comportamento destes objetos.
Um **gerente** é a parte da aplicação distribuída em que a troca de informação toma o papel de gerente e o **agente** é a parte da aplicação distribuída em que a troca de informação toma o papel de agente.

A X.700, Ref. [33], define o contexto de gerenciamento e a terminologia usada.

Um recurso que esteja sujeito ao gerenciamento é representado (de forma abstrata) por um **objeto gerenciado**. As propriedades de um objeto são denominadas de **atributos**. Um objeto é definido em termos dos **atributos** que ele possui, das **operações** que podem ser desempenhadas por ele, das **notificações** que podem ser enviadas por ele e das **relações** que ele tenha com outros objetos gerenciados.

A informação de gerenciamento pode ser vista como uma coleção de objetos gerenciados, cada um com seus atributos.

O conjunto de objetos gerenciados em um sistema, junto como seus atributos, constitui a **MIB** (*"Management Information Base"*, base de informação de gerenciamento) do sistema.
A figura abaixo, baseada na Ref. [80], Figura 10/M3010, mostra a relação entre objetos e recursos gerenciados para o caso de um elemento de rede (NE).

Relacões entre Obietos e Recursos Gerenciados em um NE

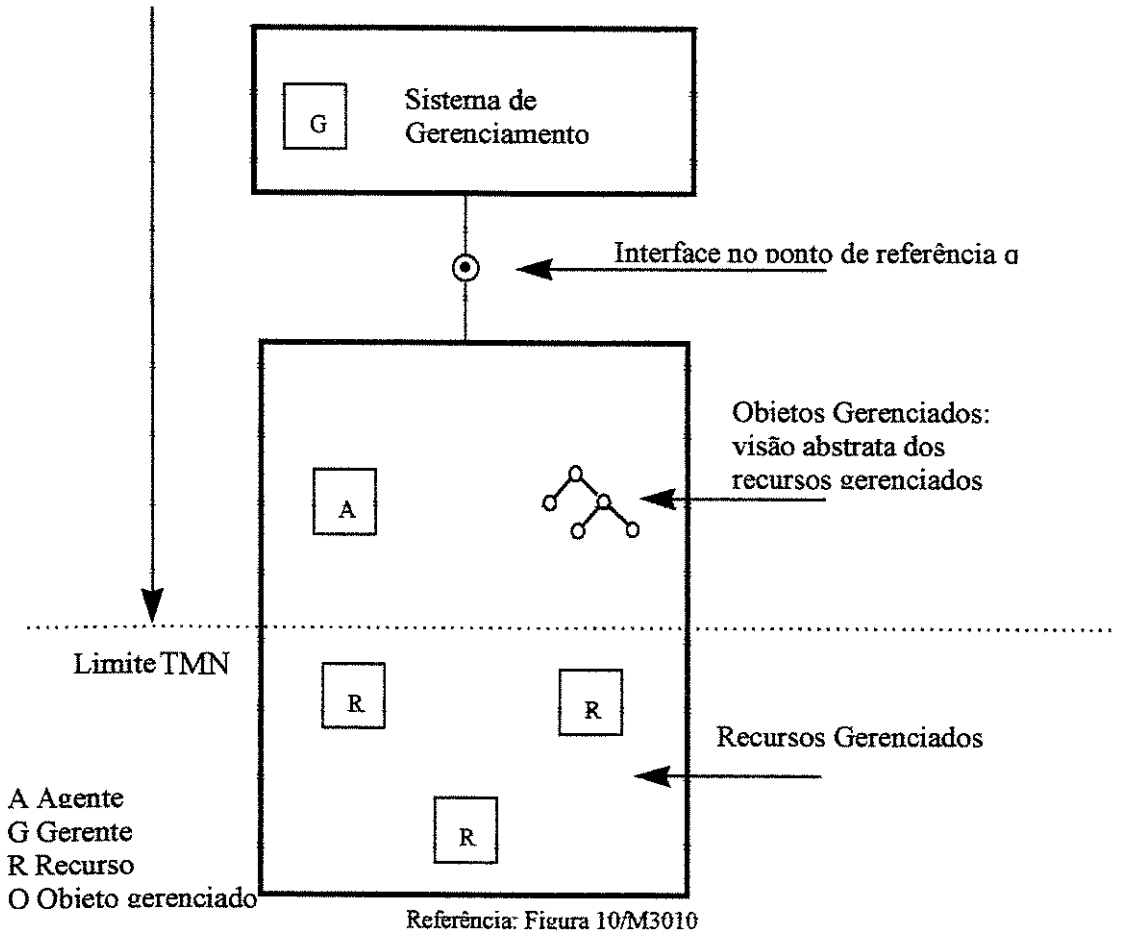


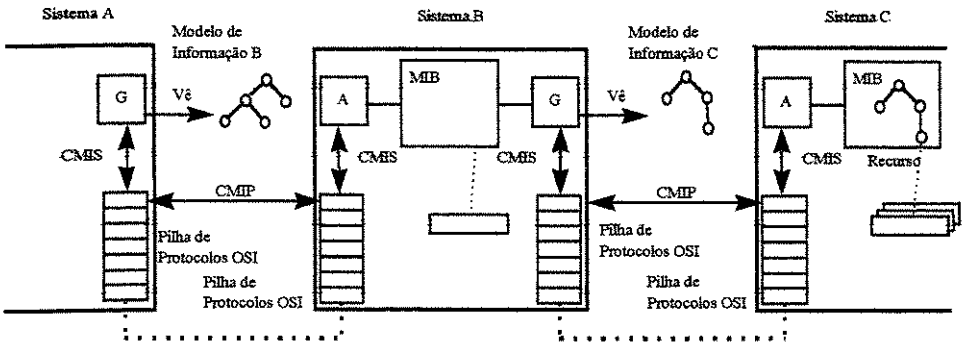
Figura # 2.6 - Relações entre Objetos e Recursos Gerenciados em um NE

Todas as operações de gerenciamento entre um gerente e um agente são feitas usando **CMIS** (Ref. [35] - "*Common Management Information Service*", serviço de informação comum de gerenciamento) e **CMIP** (Ref. [36] - "*Common Management Information Protocol*", protocolo de informação comum de gerenciamento).

A Recomendação M.3100, Ref. [17] trata tópicos relativos às classes dos objetos gerenciados e heranças. Define uma série de classes do modelo de informação de gerenciamento usando GDMO (que está comentado mais a frente). Não trata aspectos específicos da tecnologia usada. É muito abrangente mas não é exaustiva. Muitas classes adicionais precisam ser definidas a parte.

Em uma aplicação TMN é permitido o uso de encadeamento de gerentes e agentes, de forma que uma mesma entidade distribuída pode assumir tanto o papel de agente em algumas situações como o de gerente em outras.
A figura abaixo, baseada na Ref. [80], Figura 11/M3010, mostra um sistema A que gerencia um sistema B, o qual gerencia um sistema C. Note-se que as MIBs de cada um dos 3 sistemas não são iguais.

Relações entre Objetos e Recursos Gerenciados em um NE



CMIP Protocolo de Informação de Gerenciamanto Comum ("Commom Management Information Protocol)
CMIS Sistema de Informação de Gerenciamanto Comum (" Commom Management Information System")
MIB Base de Informação de Gerenciamento ("Management Information Base")
A Agente
G Gerente

Referência: Figura 11/M3010

Figura # 2.7 - Relações entre Objetos e Recursos Gerenciados em um NE

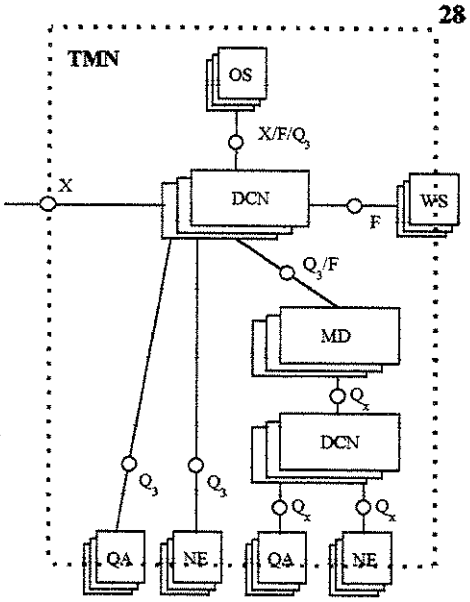
Para a identificação unívoca de um nome em ambiente TMN, é usado um sistema de identificação em subconjuntos que consideram o endereço do ponto de acesso ao serviço de rede e os títulos do sistema (processos e entidades da aplicação). Um serviço de diretório deve ser usado para suportar os requisitos de nomes e endereços. É referenciada a Ref. [87] - OSI - *"The Directory: Overviews of Concepts, Models and Services"*, visões gerais de conceitos, modelos e serviços.

2.2.5 A arquitetura Física da TMN

Uma visão simplificada pode ser vista na **Figura # 2.8**, baseada na Ref. [80], Figura 14/M3010.

Os seguintes blocos são previstos: OS (*"Operations System"*, sistema de operações), NE (*"Network Element"*, elemento de rede), WS (*"Workstation"*, estação de trabalho), MD (*"Mediation Device"*, dispositivo de mediação), QA (*"Q Adaptor"*, adaptador Q) e DCN (*"Data Communication Network"*, rede de comunicação de dados). Implementam, um a um, essencial e respectivamente, as funções OSF, NEF, WSF, MF, QAF e DCF. Em cada um deles, no entanto, pode opcionalmente ser implementada uma função (ou mais de uma) que é classificada como essencial em outro bloco.

Exemplo Simplificado de Arquitetura TMN



DCN	Rede de Comunicação de Dados (*"Data Comunication Network"*)
MD	Dispositivo de Mediação (*"Mediation Device"*)
NE	Elemento de Rede (*"Network Element"*)
OS	Sistema de Operações (*"Operations System"*)
WS	Estação de Trabalho (*"Work Station"*)
QA	Adaptador Q (*"Q Adaptor"*)
—O— F	Interfaces

Referência: Figura 14/M3010

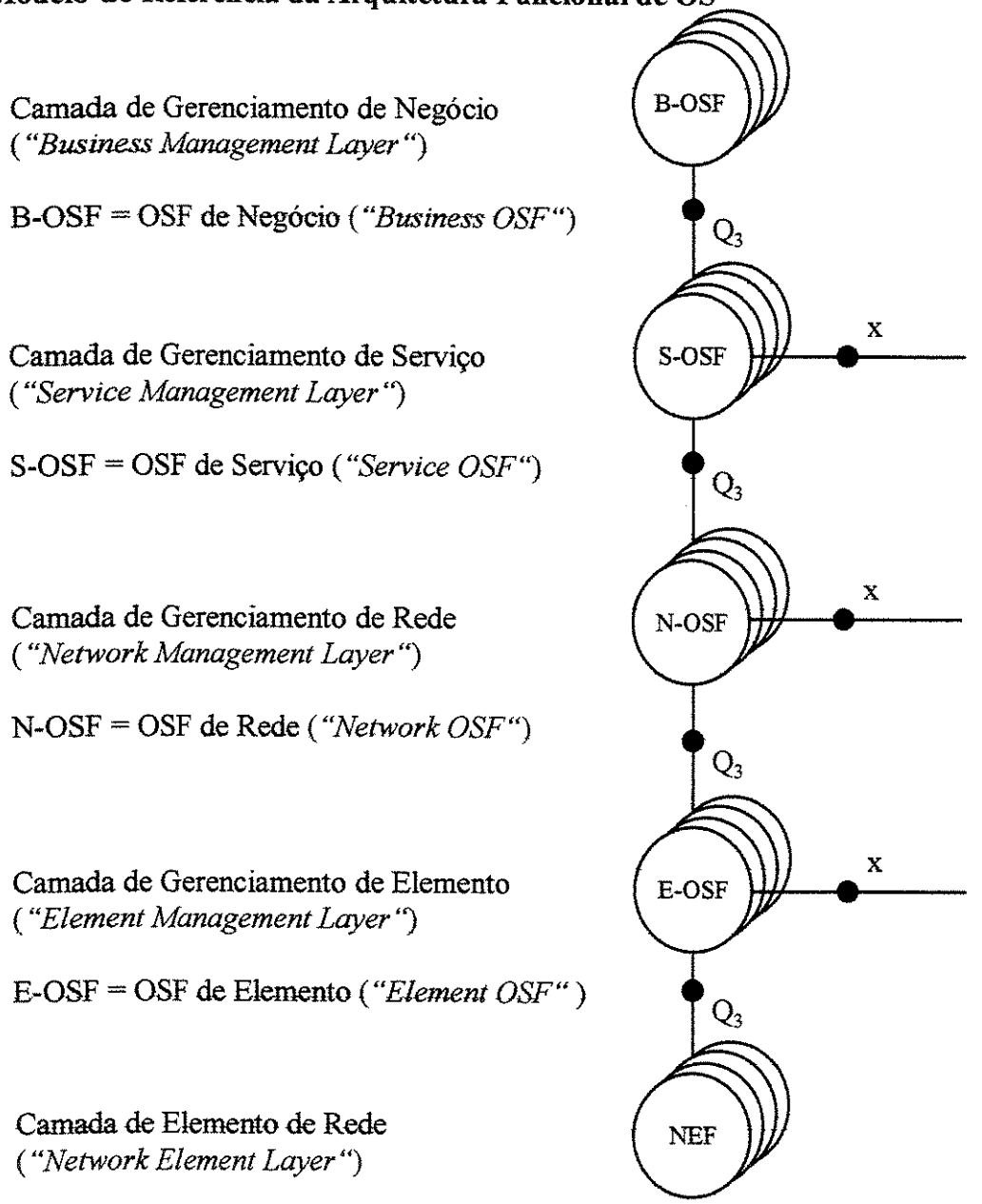
Figura # 2.8 – Exemplo Simplificado de Arquitetura TMN

2.2.6 A Arquitetura Lógica em Camadas da TMN

As OSFs podem ser agrupadas hierarquicamente em diferentes camadas, cada qual responsável por uma parte do gerenciamento TMN. Uma visão simplificada pode ser vista na figura abaixo, baseada na Ref. [80], Figura 18/M3010.

Pelo menos quatro camadas de OSFs estão previstas: gerenciamento do elemento de rede (a de mais baixo nível), gerenciamento da rede, gerenciamento de serviço e gerenciamento do negócio (a de mais alto nível). Cada camada pode tratar diferentes áreas funcionais: falhas, configuração, desempenho, segurança, contabilidade, planejamento, aprovisionamento, etc. A idéia básica é que as OSFs de negócio estejam relacionadas com toda a empresa, isto é, que trate todos os serviços e redes e conduza a coordenação geral dos negócios. As OSFs de serviços são responsáveis pelos serviços de uma ou mais redes. As OSFs de rede gerenciam as redes e as OSFs de elementos gerenciam os elementos.

Modelo de Referência da Arquitetura Funcional de OS



Referência: Figura 18/M3010

Figura # 2.9 - Modelo de Referência da Arquitetura Funcional de OS

2.2.6 Modelo de Informação

As definições associadas podem ser vistas na Ref. [35], Ref. [38], Ref. [7] e Ref. [140].
O modelo de informação trata os objetos gerenciados, define a estrutura da informação de gerenciamento que é usada pelos protocolos de comunicações e modela os aspectos de gerenciamento dos recursos. Os recursos de processamento e de comunicações de dados que podem ser usados para o gerenciamento incluem os protocolos, os modems, as conexões, etc. Os objetos gerenciados são abstrações destes recursos.

Uma **classe** de um objeto é uma coleção de "pacotes" ("***packages***"). Um pacote é formado por um conjunto de atributos (propriedades), operações, notificações e comportamentos; pode ser obrigatório ou condicional. Um **pacote condicional** está presente em um objeto somente se forem atendidas as condições que tenham sido previamente definidas na classe do objeto.

Uma ou mais de uma classe pode(m) dar origem a outra **subclasse** através de uma **especialização**. A classe que foi usada para derivar a outra é chamada de **superclasse**. O mecanismo através do qual os atributos, notificações, operações e comportamentos de uma subclasse são adquiridos de uma superclasse é denominado de **herança**. Quanto uma subclasse é gerada a partir de mais de uma superclasse tem-se uma **herança múltipla**.

Um objeto é uma **instância** de uma classe de um objeto.

Uma **árvore de herança** representa a linhagem de um objeto.

Através do **polimorfismo** se um usuário quer acessar todas as instâncias de classes de objetos hierarquicamente inferiores a uma certa instância de uma classe de objeto que é superior às anteriores, basta acessar a instância da classe de objeto de nível superior para estar acessando as demais inferiores. Por exemplo: se a classe juntores de uma rota incluir juntores de todos tipos de sinalização usados, para visualizar todos os juntores de todas as sinalizações usadas em uma rota, bastaria solicitar a visualização dos juntores da rota.

Para permitir que o solicitante de um serviço possa acessar, através do CMIS e do CMIP, os recursos associados a um objeto, sem os detalhes que lhe sejam irrelevantes a respeito deste objeto, é necessário **esconder** os detalhes de como a classe do objeto gerenciado implementa a funcionalidade modelada. Diz-se que a definição da classe do objeto **encapsula** ou **contém** dentro de si própria tanto os mecanismos quanto os dados necessários para implementar a funcionalidade do recurso modelado. A esta propriedade da classe de um objeto dá-se o nome de **encapsulamento**.

É usado um mecanismo denominado de "**árvore de registro global**" ("***Global Registration Tree - GRT***") para atribuir identificadores únicos aos objetos, atributos, operações e notificações. Cada ramo da GRT recebe um identificador numérico denominado **identificador de objeto** ("***Object identifier" - OID***). Os ramos são identificados por nomes simbólicos e a **raiz** ("***root***") recebe o OID 0 (zero). Os ramos da raiz são administrados pela ISO. Os ramos abaixo da raiz pertencem às várias organizações envolvidas com gerenciamento de rede e cada árvore associada é administrada de forma autônoma por cada uma destas organizações. Isto significa que definir um objeto TMN pressupões antes o registro formal na ISO da entidade que o for definir tais objetos. Portanto, registrar uma classe de objeto pressupõe diferentes níveis de aprovação: na ISO e na empresa específica.

Para descrever a hierarquia das instâncias dos objetos é usado um mecanismo denominado "**abrangência do objeto**" ("***object containment***"). Permite agrupar as instâncias dos objetos gerenciados dentro de outras instâncias de objeto. Isto dá origem a 3 árvores hierárquicas de instâncias de objetos: hierarquia de abrangência (***"containment hierarchy"***), árvore de nomes ("***naming tree***") e árvore de gerenciamento de informação ("***management information tree - MIT***"). A MIT representa todas as instâncias dos objetos dentro de um mesmo domínio de gerenciamento.

A "associação de nome" ("***name binding***") define como uma certa instância de um objeto é nomeada e a possível abrangência das instâncias do objeto.

É necessário distinguir diferentes instâncias de uma mesma classe de objeto. Para isto deve ser usado um atributo na classe do objeto que permita identificar de forma única uma instância do objeto. Por exemplo, para um canal de voz de uma rota de sinalização número 7 - RSN7, o número do código de identificação do circuito de voz - CIC.

Para se atribuir um valor a um atributo de um objeto é usada uma "**atribuição de valor a um atributo**" ("***attribute value assertion - AVA***"). Tal AVA é denominada de "**nome de distinção relativa da instância**" ("***instance relative distinguishing name - RDN***"). O RDN de um objeto é o que diferencia a instância de uma classe de um objeto de outras instâncias da mesma classe dentro da mesma MIT. No entanto, para se diferenciar duas instâncias independentemente da MIT em que se encontra é usado um "**nome de distinção completo**" destas instâncias ("***fully distinguishing name - FDN*** ou somente ***DN***"). O FDN é a concatenação de todos RDNs através da MIT desde o topo até a instância específica do objeto em questão.

Várias classes de objetos foram previamente definidas pelo ITU-T. Classes adicionais são definidas a partir destas classes. Detalhes sobre estas classes e sobre as funções de gerenciamento associadas podem ser encontradas nas referências abaixo:

- gerenciamento de objeto - Ref. [41];
- gerenciamento de estado - Ref. [42];
- gerenciamento de relatório de alarme - Ref. [44];
- gerenciamento de relatório de evento - Ref. [45];
- gerenciamento de controle de "*log*" - Ref. [46];
- gerenciamento de relatório de alarme de segurança - Ref. [47];
- etc.

2.2.7 Uso de ASN.1 e GDMO na TMN

Os objetos da MIB são modelados usando ASN.1 e GDMO. Esta codificação é muito precisa mas cheia de detalhes e de regras que requerem plataformas de desenvolvimento específicas. Estas plataformas precisam ser capazes de permitir independência do sistema operacional e da linguagem de programação que irá hospedar o código desenvolvido. Também apresentam recursos gráficos e de manipulação da MIB para facilitar o manuseio da informação. A seguir são apresentados, resumidamente os principais conceitos de ASN.1 e de GDMO e a seguir são apresentados alguns padrões de APIs usadas em plataformas que estão comercialmente disponíveis no mercado.

ASN.1 ("Abstract Syntax Notation 1") - "Notação de Sintaxe Abstrata 1"

É a sintaxe usada para definir, de uma forma padronizada, os tipos de dados associados às definições das classes dos objetos. Está definida na Ref. [23]. Este é um tema inteiramente relacionado com a camada de apresentação do modelo ISO. Uma boa explicação a respeito da ASN.1 é encontrada na Ref. [139], na Ref. [7] e na Ref. [73].

A ASN.1 inclui regras para a definição dos tipos de dados (**sintaxe abstrata**) e regras para a codificação dos dados a serem transferidos entre duas aplicações diferentes (**sintaxe de transferência**). Para isto prevê vários "**tipos construtores**" ("***constructors types***") tais como inteiro, booleano, número real, caracteres, seqüências, conjunto de bits ("***bit string***"), valores enumerados, valor nulo ("***NULL***"), identificador de objeto.
A sintaxe abstrata somente define a forma usada para a definição das classes. Não há nenhum dado real associado com uma definição de uma classe que use a sintaxe abstrata. Máquinas diferentes podem usar diferentes formas de representações de dados (**sintaxe concreta**) - uma representação codificada dos tipos de dados e dos elementos reais dos

dados. As informações trocadas entre aplicações precisam usar uma sintaxe concreta específica previamente acordada. Para representar os dados codificados pela sintaxe concreta são usadas **regras básicas de codificação** (“***Basic Encoding Rules - BER***”) para definir como a ASN.1 é usada para representar os conjuntos de octetos. A BER contém regras para representar em bytes as instâncias das classes definidas em ASN.1 e pode também ser usada para a sintaxe de transferência.

GDMO

Para especificar uma classe é usado um conjunto de “gabaritos” (“***templates***”) especificados de acordo com a - X.722 (GDMO) [40]. Estes “gabaritos” permitem especificar a posição hierárquica do objeto na sua hierarquia de herança, os pacotes obrigatórios, os pacotes condicionais, as condições associadas, etc. A estrutura dos pacotes inclui os atributos, as operações, os comportamentos e as notificações relativas ao objeto. A GDMO usa a ASN.1 para definir os tipos de dados dos atributos dos objetos. Possui 9 “gabaritos” (“***templates***”): atributo (“***attribute***”), grupo de atributos (“***attribute group***”), ação (“***action***”), notificação (“***notification***”), comportamento (“***behaviour***”), parâmetro (“***parameter***”), pacote (“***package***”), classe de objeto gerenciado (“***managed object class***”) e “associação de nome” (“***name binding***”). O parâmetro é usado para definir erros e mensagens.

Uso de GDMO/ASN.1 em Plataformas de Desenvolvimento

Para ser gerado código executável para uma máquina específica a partir dos gabaritos GDMO é necessário traduzir todas estas formas de representações em código de uma linguagem particular para uma máquina específica. Como em geral tal máquina está associada a um certo sistema operacional, isto acarreta a necessidade de serem usadas plataformas com APIs abertas que permitam tratar tanto as estruturas abstratas dos dados que serão usados na MIB e nos PDUs quanto o envio e a recepção destes dados através dos PDUs do CMIP (para o caso da TMN).

Dentre as entidades que atuam na padronização de APIs podem ser citadas a **POSIX** (“*Portable Operating System Interface for Computer Environment*” - Interface de Sistema Operacional “Portátil” para Ambiente de Computador) e a **X/Open**.

A **X/Open** definiu duas APIs muito usadas nas plataformas: XOM e XMP. A construção de aplicações TMN depende do uso de ASN.1 e GSMO e isto é facilitado pelo uso das APIs XOM/XMP em plataformas comerciais existentes no mercado.

XMP (“*X/Open Management Protocol*”) - protocolo de gerenciamento X/Open - é a API usada para a interface de envio e recepção dos dados do CMIP e que também pode ser usada para o SNMP (“*System Network Management Protocol*”) - protocolo de gerenciamento de rede de sistema - o protocolo de gerenciamento de redes IP da Internet. Consiste num conjunto de funções para suportar o mapeamento das aplicações de forma comum tanto para os serviços do CMIP quanto para os do SNMP.

XOM (“*X/Open Abstract Data Manipulation*”) - manuseio de dados abstratos X/Open - é usada para facilitar o uso dos tipos de dados definidos em ASN.1 e prover a definição dos argumentos a serem usados nas funções do XMP.

Mais detalhes sobre as APIs da X/Open e sobre plataformas que suportam APIs padronizadas podem ser vistos na Ref. [151], na Ref. [7] e na Ref. [64]. Plataformas deste tipo em geral são muito caras e de uso restrito a pessoas muito especializadas.

2.3 *Comentários Sobre O Escopo do Trabalho*

A solução TMN permite atender à maioria das necessidades operacionais de uma empresa telefônica. Muitas destas necessidades já são mesmo propostas em pacotes comerciais de vários fabricantes. Há várias plataformas para o desenvolvimento das aplicações e a compatibilidade entre agentes e gerentes é baseada no protocolo CMIP e na MIB usada. A MIB é padronizada na TMN somente para os níveis hierárquicos mais elevados. Os níveis mais baixos em geral são específicos de cada implementação, de cada fabricante e de cada produto. O acesso a esta MIB é uma informação tida por muitas empresas do ramo como altamente restrita. O desenvolvimento de pequenas aplicações ou de aplicações ainda não existentes em geral é difícil ou morosa porque, além de depender da disponibilidade de plataformas caras e de pessoas muito especializadas, em geral esbarram em negociações específicas entre a operadora interessada e um ou mais fabricantes, cada qual com interesses diferentes.
A dissertação pretende mostrar como criar objetos funcionalmente semelhantes àqueles definidos para a MIB da TMN sem usar as arquiteturas física, funcional, de informação, etc, nem o modelo de informação, nem os protocolos definidos para a TMN. Pretende sugerir que se use o modelo de informação definido pelo ITU-T como ponto de partida para o trabalho. A árvore hieráquica usada para definir os objetos que compõem uma rede ou um elemento de rede pode ser usada com base na que foi definida pelo ITU-T mas principalmente a forma de representar estes objetos é o que muda. Ao invés de se usar a MIB da TMN se propõe o uso de outras metodologias conforme será apresentado a partir do capítulo 3. Assim, a criação de ESN7 apresentada nesta dissertação não usa a estrutura de camadas OSI da TMN nem foi desenvolvida usando as plataformas de desenvolvimento referidas acima. Portanto não se usa ASN.1, GDMO, XOM/XMP, etc., mas são usados recursos mais recentes da tecnologia da informação conforme apresentado nos próximos capítulos. Os conceitos da arquitetura lógica em camadas, da arquitetura da informação e da MIB da TMN, no entanto, podem ser usados como pontos de referências para se definir quais são os aspectos telefônicos relevantes para serem levados em consideração.
O desenvolvimento feito se restringe à criação de ESN7. A criação do ESN7 é uma parte de um conjunto maior necessário para o gerenciamento de configuração de ESN7 (que, por sua vez, inclui outros aspectos como a modificação das características de um ESN7, a visualização dos dados de um ESN7 previamente criado, a criação de circuitos de voz associados ao ESN7, etc.). O gerenciamento de configuração de ESN7 se enquadra funcionalmente no nível 2 da arquitetura lógica de camadas da TMN, isto é, a nível de gerenciamento de elemento de rede.

Para ser mais claro sobre aproveitar o que já existe de cultura telefônica na TMN seguem alguns esclarecimentos adicionais. Parte deles será usada nos capítulos mais a frente que cuidam da implementação do protótipo.

A partir das especificações das árvores de herança da MIB da TMN, das recomendações da série M.3000 do ITU-T e de algum estudo adicional do assunto, é possível dizer que uma certa rede telefônica está composta de várias partes que se caracterizam e se ajuntam de uma forma hierárquica tal como mostrado, de forma simplificada, a seguir.

É importante deixar claro que existem outras formas de apresentar a classificação das partes que compõem este assunto. Há que tomar um certo cuidado para, ao caracterizar cada tópico fazer uma abstração da arquitetura específica de um equipamento específico, seja de um hardware ou de um software. Se isto não for feito a classificação das partes tenderá a ser orientada a uma arquitetura de um sistema específico, à arquitetura adotada por um certo fabricante em seus produtos, etc. Há que buscar as funcionalidades e as ações associadas a elas de uma forma genérica. Sem dúvida que é importante classificar a informação de uma forma hierárquica e o mais completa possível. No entanto cabe ressaltar que o objetivo das listas apresentadas não é apresentar o tema de forma exaustiva mas tão-somente situar o contexto do trabalho em relação a uma central telefônica. Como será visto nos próximos capítulos, as funcionalidades, a título de exemplo, serão mapeadas em parâmetros e as ações em métodos.

- Rede
  - Outras redes
  - Elementos gerenciados
    - Central Pública de Comutação Telefônica
    - Sistema de Energia
    - etc.
  - Conexões entre elementos gerenciados
    - Juntores de Centrais Públicas de Comutação Pública
    - Juntores de Centrais Privadas de Comutação Pública
    - Transmissão e Transporte
    - etc
  - etc.

Considerando apenas uma central pública de comutação telefônica podemos enumerar algumas de suas partes funcionais como listado a seguir.

- Central Pública de Comutação Telefônica
  - Assinantes
  - Juntores
  - Órgãos Complementares
  - Recursos de Tarifação
  - Recursos de Processamento de Chamadas
  - Outros Recursos
  - etc.

Se considerarmos apenas os assinantes, poderíamos caracterizá-los conforme abaixo.

- Assinantes (Ver Nota)
  - Remoto
  - Local
  - Pré-pago
  - Pós-pago
  - Tráfego Irrestrito
  - Restrito Restrito

        - Só faz chamada local
        - Só faz chamada local e regional
        - Só faz chamada local, regional e nacional
        - etc.
        - Só origina
        - Só recebe
    - Telefone público **(*1)**
        - Telefone público local
            - Funcionamento com cartão
            - Funcionamento com ficha
                - Ficha simples
                - Ficha multivalorada
        - Telefone público interurbano
            - Funcionamento com cartão
            - Funcionamento com ficha multivalorada
    - Telefone semi-público **(*1)**
    - PABX
    - Envio envio de pulso de tarifação **(*1)**
    - Envio envio de pulso de atendimento
        - 12 KHz
        - Inversão simples de polaridade
    - Assinante fixo comum (POTS)
    - Assinante RDSI
    - Assinante Celular Fixo
    - Assinante ADSL
    - Assinante Celular Móvil
    - Assinante com Serviço Suplementar
        - Transferência de chamada
            - Incondicional
            - Só quando ocupado
            - etc.
        - Chamada em espera
        - etc.
    - etc.

(*1) Cada um deste tipos de assinantes ainda devem ser classificados pelo tipo de sinalização de linha usada:

- 12 KHz
- Inversão dupla de polaridade

Se considerarmos os juntores, poderíamos caracterizá-los conforme abaixo.

- Juntores
    - **(*2)** Juntores de Central Pública de Comutação Telefônica - **Ver Nota - (*3)**
        - Juntores com sinalização por canal associado
            - Juntores Corrente Contínua (CC)

- Juntores E+M-Pulsado
- Juntores E+M-contínuo
- Juntores R2
- etc.

- Juntores com sinalização por canal comum N7
  - Canais de Voz
  - Conjuntos de Enlaces de SN7
  - Rotas de SN7
  - Enlaces de SN7
- etc.

- Juntores de Centrais Privadas de Comutação Telefônica
  - Idem acima **(*2)**

(*3) - Nota:

Esta é uma visão bastante simplificada. Os juntores, além de serem classificados pelo seu tipo de sinalização, dentre outras coisas devem estar associados a uma rota e precisam ser classificados pelo seu método de captura (cíclica, randômica, etc.).

Os órgãos complementares podem variar de fabricante para fabricante e de central para central. Uma possível classificação seria:

- Órgãos Complementares
  - Tratadores de freqüências MFC (multifrequencial compelido)
    - Enviadores
    - Receptores
  - Tratadores de tons audíveis
    - Enviadores
    - Receptores
  - Tratadores de sinais audíveis
    - Mensagens
    - Tons
  - Circuitos de conferência
  - Canceladores de Eco
  - VOCODERs
  - etc.

Ao considerarmos os recursos de tarifação poderemos ter:

- Recursos de Tarifação
  - Bilhetagem Automática
  - Contadores de Pulsos
  - Memória de massa
    - Fita magnética
    - Fita DAT
    - Disco óptico
    - etc.
  - Enlaces para Transferência de Arquivos

- FTAM sobre pilha OSI
- TCP/IP + FTP
- etc.

- Métodos de cobrança do serviço
  - Por vazão de dados
  - Por tempo de uso
    - Medição Simples
    - Multi-Medição
      - Método de Karlson
        - Karlson Puro
        - Karlson Acrescido
        - Karlson Modificado
      - Trem de Pulsos (*"burst"*)
  - etc.

Igualmente, para os recursos de processamento de chamadas teremos, de uma forma muito simplificada:

- Recursos de Processamento de Chamadas
  - Encaminhamento
    - Número de dígitos que dever esperar receber antes encaminhar
    - Preparação de dígitos
      - Remoção de dígitos
      - Inserção de dígitos
      - Troca de dígitos
    - Tipo de rota de saída
      - Sinalização a usar
      - Tipo de portadora
        - 64 K preferencial
        - Áudio
        - etc.
    - Diferenciação pela rota de entrada
    - etc.
  - Enumeração
    - Natureza e quantidade de dígitos recebidos e a enviar
      - Código Indicativo de chamada a cobrar
      - Código da operadora de longa distância
      - Prefixo nacional
      - Prefixo regional
      - Prefixo da central
      - etc.
  - etc.

Como outros recursos podemos citar:

- Outros Recursos

- Recursos de Operação
- Recursos de Manutenção
- Recursos de Administração
- Recursos de Análise de Desempenho
- etc.

Enfocando somente a uma parte destes conjuntos, para classificar, de uma forma geral, as necessidades de gerenciamento de enlaces de SN7, podemos considerar parte do que está previsto na Ref. [19]. Ali estão abordados pelo menos os seguintes tipos possíveis de **serviços** em uma central telefônica:

1. Planejamento
    - enlaces de sinalização
    - rotas de sinalização
    - configuração
    - dimensionamento
    - planejamento dos sistemas de transmissão

2. Manutenção
    - detecção e correção de falhas
    - verificação de encaminhamento de dados

3. Desempenho
    - medidas de carga
    - medidas de congestionamento
    - controles

4. Aprovisionamento
    - novos enlaces de sinalização
    - pontos de transferência de sinalização

Note-se que, de uma forma genérica, vários **serviços** podem se aplicar a várias funcionalidades diferentes de uma central.

Por outro lado, se considerarmos apenas o gerenciamento de desempenho em mais detalhes veremos que ele pode incluir vários temas conforme listado a seguir.

a) Coleta de dados:
- definir intervalo de coleta de dados;
- suspender e reativar a coleta de dados;
- zerar os contadores de monitoração de desempenho;
- agendar a coleta de dados.

b) Armazenamento de dados:
- definir a duração do registro histórico dos dados de gerenciamento;
- fornecer registro histórico dos dados com base em algum critério de seleção;

- remover o registro histórico dos dados.

c) Relatório de Dados:

- solicitar relatório de dados;
- fornecer relatório de dados;
- habilitar / inibir o fornecimento de relatório de dados agendados;
- fornecer relatório de dados com base em algum critério de seleção.

Se se consideram somente as funcionalidade de aprovisionamento de novos ESN7 podem ser enumeradas pelo menos as seguintes aplicações:

- criação de ESN7
- visualização de características de ESN7
- modificaçao de características de ESN7
- associação de circuitos de voz a ESN7
- etc.

Note-se que vários outros aspectos do gerenciamento de desempenho acima descritos podem ser aplicados para ESN7: supervisão de falhas, coleta de dados de tráfego e de estatística, etc. Cada um destes temas por si só exigem uma abordagem específica. No entanto, o tipo da interface entre o cliente e a BD da central telefônica pode ser essencialmente a mesma para temas afins, se definida adequadamente. Nos capítulos 3 e 4 abordamos como isto pode ser feito.

A idéia básica do trabalho é colocar a complexidade na definição da interface e na implementação do servidor e deixar o cliente simples o máximo para ser flexível e de rápida implementação. Não pretende, no entanto, explorar em detalhes nem o cliente nem o servidor mas apresentar como isto poderia ser feito. Percebe-se facilmente que o assunto é muito amplo. É impossível tratar todo o tema nesta dissertação. O que é tido como importante nesta dissertação é mostrar como as coisas se encaixam e indicar uma forma de conduzir o assunto. Não se ressalta o detalhe, a não ser que ele seja importante para este objetivo. O que se quer construir é um modelo que possa ser completado, um esqueleto a que se possa dar corpo. Além disto, o objetivo é atingir os usuários das BD das centrais telefônicas, que compreendem muito bem esta cultura telefônica mas não conhecem, necessariamente, os aspectos da tecnologia da informação que lhes podem ser úteis para a solução de seus problemas corriqueiros do dia a dia. Por isto esta dissertação se restringirá a somente uma pequena parte deste conjunto de temas de telefonia. Sendo mais específico, a uma pequena parte do aprovisionamento de novos enlaces de sinalização N7, a criação de um ESN7.

No capítulo 3 são apresentados os recursos que se pretende usar para este objetivo.

No capítulo 4 é dado um exemplo de como isto pode ser feito para um caso em que se procurou simplificar ao máximo o tema para ir ao cerne da questão, ao invés de se ater a detalhes.

Capítulo 3 – Recursos: CORBA, JAVA e UML

A interação com as BDs das centrais telefônicas pode ser caracterizada como uma aplicação típica de sistemas distribuídos. Há muitos recursos e abordagens atuais para a solução de problemas de ambientes distribuídos. Este capítulo aborda dois temas específicos ligados a este assunto e que foram escolhidos para fazerem parte da solução proposta para a interação com as BDs das centrais telefônicas. CORBA e JAVA foram escolhidos por permitirem o uso de soluções padronizadas, independentes do hardware, do sistema operacional e da linguagem de programação usada na implementação do protótipo e por serem recursos de fácil acesso.
UML não foi usado devido à simplicidade da implementação feita. O propósito de apresentar tal assunto aqui é realçar sua importância como um recurso que deve ser utilizado em desenvolvimentos reais de complexidade maior.

3.1. CORBA

Esta descrição está baseada nas seguintes referências, dentre outras: Ref. [127], Ref. [171], Ref. [108], Ref. [50], outras referências encontradas na Ref. [1] e na principalmente na página "web" da OMG. Tais referências devem ser consultadas para maiores detalhes.

CORBA (*"Common Object Request Broker Architecture"*) – Arquitetura Comum de "Agenciador" para Solicitação de Objeto – é uma arquitetura padronizada pela **OMG** (*"Object Management Group"*) – Grupo de Gerenciamento de Objeto – para permitir a interação entre objetos heterogêneos de sistemas distribuídos, independentemente das plataformas em que estejam residentes, da linguagem usada para defini-los, da localização destes objetos na rede e do tipo de plataforma usada (software e hardware).
A figura abaixo dá uma visão geral da estrutura das interfaces CORBA que serão descritas nos itens a seguir.

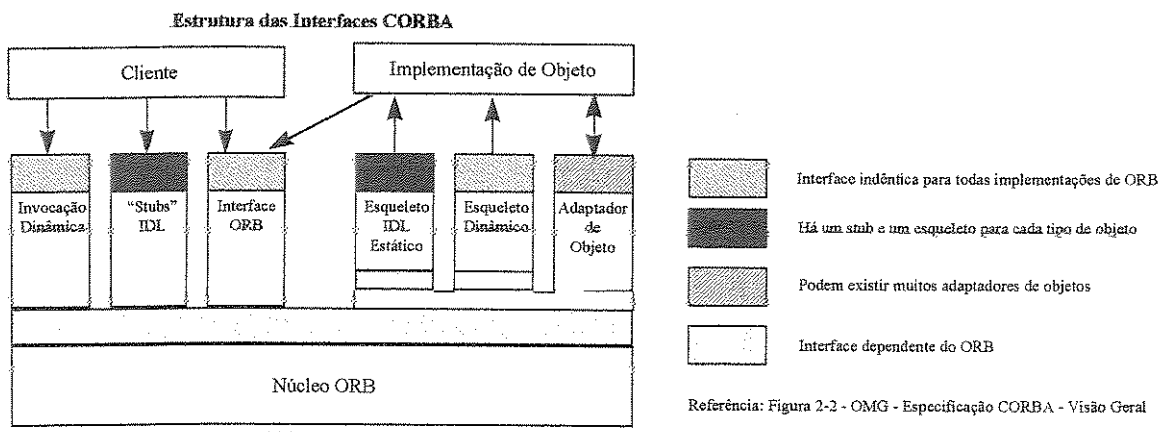


Figura # 3.1 - Estrutura das Interfaces CORBA

O **cliente** é a entidade que deseja executar uma operação no objeto e a **implementação do objeto** é constituída pelo código e pelos dados que efetivamente implementam o objeto. Denomina-se "**fábrica de objetos**" ao objeto que permite criar outros objetos distribuídos.

Todo o modelo CORBA está associado com a solicitação de serviços a objetos. Para isto a OMG definiu, entre outros, uma linguagem de definição de interfaces (IDL), um "agenciador" para solicitação dos objetos (ORB), um protocolo para interação destes ORB via Internet (IIOP), um conjunto de serviços que podem ser usados (COS) e adaptadores de objeto (OA) para interação entre os ORB.

3.1.1 **ORB** (*"Object Request Broker"*)

As solicitações são enviadas de um cliente para um servidor através de um "Agenciador" para Solicitação de Objeto denominado **ORB** – (*"Object Request Broker"*). O ORB fica no núcleo da CORBA. Ele é responsável por localizar a implementação do objeto na rede distribuída, preparar a implementação para receber a solicitação, encaminhar a solicitação feita no cliente ao objeto implementado no servidor remoto, esperar pelo resultado da solicitação e retornar os eventuais resultados ao cliente. O ORB possibilita o tratamento dos objetos remotos e locais de forma transparente para o cliente, isto é, ele provê **transparência de localização**. Ele também permite que o cliente que está fazendo a solicitação tenha sido implementado em uma linguagem de programação diferente daquela em que foi implementado o objeto no servidor ou seja, provê **transparência de linguagem de programação**. Para que isto seja possível a OMG define uma linguagem de definição de interfaces denominada **IDL** – (*"Interface Definition Language"*). O ORB é responsável pelas "traduções" necessárias para tais interações entre as linguagens que já foram padronizadas para tratar a IDL.

3.1.2 **COS** (*"CORBA Services"*)

A OMG definiu um conjunto de serviços para a integração e interoperação dos objetos distribuídos. Eles são denominados objetos de serviços CORBA – **COS** (*"CORBA Services"*). São definidos como objetos CORBA, usam interfaces IDL e operam através do ORB.
Para que tais serviços sejam viabilizados são definidas as interfaces IDL que devem ser respeitadas. Alguns destes serviços estão citados a seguir.

- **Serviço relativo ao "ciclo de vida do objeto"**: define como os objetos são criados, removidos, movidos e copiados.
- **Serviço de nomes**: define como os objetos CORBA podem ser identificados de forma amigável.
- **Serviço de Eventos**: cuida da comunicação de eventos entre os objetos distribuídos.
- **Serviço "relações"** (*"Relationships"*): providencia suporte para relações arbitrárias entre objetos CORBA: criar uma relação, definir os papéis e as regras, excluir, navegar e gerenciar as relações entre os objetos.
- **Serviço de "controle para acesso externo"** (*"Externalization"*): transformação de objetos CORBA de e para meios externos.
- **Serviço de transações**: coordena o acesso a um ou mais objetos CORBA através de uma ou mais operações.
- **Serviço de controle de concorrência** (*"Concurrency Control"*): provê um serviço de "restrição de acesso" (*"locking"*) para garantir acesso serial ao objetos CORBA.
- **Serviço de "negociações"** (*"trader"*): localização de objetos CORBA com base na descrição dos serviços ofertados pelo objeto.
- **Serviço de consulta** (*"query"*): suporta consulta a objetos.

3.1.3 Facilidades Comuns

Com base nos serviços anteriores são definidas duas categorias de "**facilidades**" CORBA que são muito importantes para permitir compatibilidade de interoperação entre produtos de fabricantes diferentes. Uma 1ª. categoria é composta de 4 funções de alto nível que são comuns à maioria das áreas de aplicações e que por isto mesmo são denominadas de facilidades comuns *("Common Facilities")*. Uma 2ª. categoria de funções é formada por funções que são específicas para cada domínio de aplicação possível: medicina, telecomunicações, etc. As funções da 1ª. categoria são conhecidas como **funções horizontais** e as da 2ª. categoria como **funções verticais**.

Para as funções **horizontais** há 4 conjuntos definidos: - **interface de usuário** *("User Interface")*, **gerenciamento de informação** *(" Information Management")*, **gerenciamento de sistemas** *(" Systems Management")* e **gerenciamento de tarefas** *("Task Management")*.

Várias entidades trabalham em cooperação com a OMG para a definição das funções **verticais** – que são específicas para cada área de aplicação real. Para a área de **telecomunicações** há vários trabalhos já feitos e vários outros em andamento. Dentre eles, é de especial interesse para o presente trabalho o mapeamento de IDL em ASN.1 e GDMO. Mais detalhes podem ser vistos nas Ref. [117], [119], [120], [118], [123], [1] e [124].

3.1.4 IDL *("Interface Definition Language")*

É a linguagem definida pela OMG para a especificação das interfaces dos objetos. Ela somente define as interfaces, isto é, as operações suportadas pelo objeto. Não há recursos na IDL para definir como um objeto é implementado. Para isto é necessário usar uma linguagem qualquer de implementação tal como Java, C++, etc. A interface apenas define as "regras" que devem ser respeitadas; é um "contrato" que, sendo seguido, permite a interação entre o código que usa e o que implementa o objeto. Já existem padronizações para a IDL ser usada para C, C++, Java, Ada, COBOL, Smalltalk, C orientado a objetos e Lisp.
Na IDL as interfaces dos objetos são definidas em **módulos**. As **operações** e os **atributos** do objeto são listadas. As **exceções** que possam ocorrer em uma operação devem estar definidas. Os **tipos** de dados de uma operação com retorno de valor, seus **parâmetros** e os **atributos** associados ao objeto devem estar definidos.

Os tipos de dados usados na IDL estão listados a seguir.

- Tipos de dados **básicos** como inteiro longo, inteiro "curto", *string*, ponto flutuante, etc. (*"long, short, string, float, etc."*)
- Tipos de dados "**construídos**" a partir dos tipos de dados básicos (*"constructed"*): estrutura, união, enumeração, seqüência enumerada (*"struct, union, enum, sequence"*).
- "**Referências** a objetos 'tipificados' "
- Tipo de dado **definido dinamicamente** (*"any"*)

Os **parâmetros** para as operações suportadas por um objeto são identificados por uma palavra chave conforme sejam usados para passagem de dados de entrada ("*in*"), de saída ("*out*") ou tanto para entrada quanto saída de dados ("*inout*").

As declarações em IDL são compiladas por um compilador IDL e convertidas para as representações associadas às linguagens de programação usadas para a codificação do cliente e/ou do servidor de acordo com os "mapeamentos" ("*binding*") feitos entre a IDL e a linguagem específica.

Uma **referência de objeto** identifica o objeto distribuído de forma que um cliente possa emitir uma solicitação a este objeto e este objeto possa receber tal solicitação. Para emitir uma solicitação a um objeto, a referência de objeto é usada no cliente dentro do código da linguagem de programação específica que tiver sido usada para implementar o cliente.
Um compilador IDL para uma certa linguagem de programação gera "*adaptadores*" ("*stubs*") do lado do cliente que são usados para representar o objeto CORBA para o cliente na linguagem de programação usada no cliente. Todas as interfaces e tipos de dados usados em IDL são também representados na linguagem de programação usada do lado do cliente ou seja, o código gerado para o cliente depende não só das definições de interfaces IDL como também do código gerado pela linguagem de programação utilizada no cliente.
É feito um "mapeamento" ("*binding*") dos vários "construtores" da IDL nos "construtores" associados de cada linguagem.

Por exemplo, para mapear os "construtores" IDL nos "construtores" Java:

IDL	Java
module	pacote ("package")
interface	interface
operação	método
atributo	par de métodos
exceção	exceção

Tabela # 3.1 – Exemplo de mapeamento entre IDL e Java

Igualmente para mapear os tipos de dados IDL nos tipos de dados Java:

IDL	Java
boolean	boolean
char /wchar	char
octet	byte
short / unsigned short	short
long / unsigned long	int
long long / unsigned long long	long
float	float
double	double
string / wstring	String

Tabela # 3.2 – Mapeanto dos Tipos de Dados IDL nos Tipos de Dados Java

Há 2 tipos de exceções em IDL: de sistema e de usuário. As exceções de sistemas são aquelas geradas pelo sistema quando ocorre algo errado.
Exemplos: um método remoto que não existe foi invocado; ocorrência de erro de comunicação; o ORB não foi iniciado corretamente; etc.
Podem estar associados às exceções de sistema uns códigos específicos do fabricante da plataforma que fornecem mais dados sobre o que ocorreu.
As exceções do usuário são geradas quando ocorre algum erro na execução remota dentro do próprio método. São declaradas dentro da definição do objeto em IDL e geradas automaticamente pelo compilador de IDL para a linguagem específica usada (Java, C, etc.).

É previsto o mapeamento da IDL no GDMO /ASN.1. Embora este mapeamentos tenha muitas "perdas" inerentes às adaptações, permite o "diálogo" entre arquiteturas baseadas em CMIP ou SNMP e CORBA.
Nota:
O **SNMP** – "*System Network Management Protocol*"- protocolo de gerenciamento de rede de sistema – é usado para gerenciamento de redes IP –"*Internet Protocol*" – protocolo da Internet. É encontrado em várias aplicações de gerenciamento em centrais telefônicas mas não será analisado neste trabalho. Para detalhes ver Ref. [135], Ref. [9] e Ref. [77].

3.1.5 Protocolos GIOP, ESIOP, IIOP

Um dos objetivos da CORBA é permitir independência de plataforma de diferentes fabricantes, isto é, **portabilidade**. A portabilidade a nível de clientes tem sido alcançada pelos fabricantes mas as implementações dos objetos, na prática, ainda tem exigido adaptações.
A OMG definiu um protocolo geral para a interação de diferentes ORB denominado de **GIOP** – ("*General Inter-ORB Protocol*") – protocolo geral entre ORB. Ele é usado para definir outros protocolos. O ESIOP – ("*Environment-Specific Inter-ORB Protocol*") - protocolo entre ORB específico do ambiente – é usado em redes particulares.
O protocolo que foi definido pela OMG para padronizar as interações entre ORB de diferentes fabricantes através da Internet é denominado **IIOP** – "*(Internet Inter-ORB Protocol*"). O IIOP usa TCP/IP.
Associados a estes protocolos está o conceito de "pontes" – "*(bridge*") entre ORB diferentes. Quando dois ORB estão em um mesmo domínio eles podem se interconectar diretamente. Quando uma invocação é feita em um domínio diferente é necessária uma ponte para que ela se complete. É também possível usar uma "meia ponte" *("half bridge*") para IIOP em cada domínio e interconectar os ORB através do IIOP.

Para que os dados sejam transmitidos de uma origem para o seu destino na rede usando o IIOP, por exemplo, é necessário que sejam convertidos na origem para um formato independente da linguagem de programação usada. Igualmente, no destino, devem ser convertidos de volta para o formato específico da linguagem de programação usada no destino. A conversão dos tipos de dados IDL para a linguagem de programação (Java, no exemplo) é chamado de "***marshaling***" *("empacotamento")* e o processo reverso de "***unmarshaling***" *("desempacotamento")*. Isto é feito pelo ORB.

É também previsto o mapeamento para possibilitar a "*interoperabilidade*" de CORBA com COM – *("Component Object Model*") – modelo de objetos componentes – usado pela Microsoft.

3.1.6 Cliente

Para requisitar uma solicitação em um objeto o cliente precisa antes conhecer a **referência do objeto**, isto é, a informação necessária para especificar um objeto dentro do ORB. Esta representação pode ser diferente entre diferentes implementações de ORB. No entanto, todos ORB devem mapear a referência do objeto de forma igual para uma certa linguagem de programação. Uma referência de um objeto é criada usando uma operação específica para tal. Quando isto ocorre o objeto é identificado no sistema e é associado a uma interface e a uma implementação. A partir daí o objeto pode ser ativado, desativado ou destruído.

Um cliente pode utilizar duas formas de invocação de objeto: estática e dinâmica. A invocação **estática** é mais simples e mais eficiente mas a interface com o ORB depende do "adaptador" ("*stub*") gerado a partir da definição da interface feita em IDL. Para isto, previamente, as operações que podem ser executadas e os parâmetros associados devem ser definidos na interface IDL. O cliente inicia a solicitação chamando as rotinas do "adaptador" ("*stub*") que forem específicas para o objeto a ser invocado.

A invocação **dinâmica** permite que um cliente seja compilado sem que haja conhecimento prévio da definição em IDL da interface do objeto a ser usada pelo cliente. Para que as operações e os parâmetros dos objetos sejam descobertos quando um objeto for invocado é usado um serviço "repositório" ("*Repository service*"). Ele representa os componentes da interface como objetos e permite acessos aos componentes em tempo de execução. O cliente inicia a solicitação invocando o repositório para descobrir a definição da interface do objeto. A seguir o cliente constrói a invocação de acordo com os dados recebidos do repositório e só então faz a solicitação.

As seguintes possibilidades de comunicação podem ser usadas numa invocação:
- síncrona,
- síncrona diferida (também chamada de assíncrona) e
- unidirecional.

A comunicação **síncrona** é bloqueante, isto é, impede que o ORB faça qualquer outra coisa até que a resposta seja obtida.
A comunicação síncrona diferida (**assíncrona**) não é bloqueante mas exige que seja feito uma varredura para conhecer se a solicitação feita já foi atendida.
A escolha da invocação **estática** exclui a comunicação **síncrona diferida**.
Com a comunicação **unidirecional** não há qualquer realimentação a respeito da solicitação feita; um vez enviada, nem virá resposta do servidor nem o cliente irá questionar o resultado. No entanto, em caso de erro podem acontecer as exceções padrões previstas no sistema.

3.1.7 O Servidor

As **implementações de objetos** podem ser feitas em uma linguagem totalmente diferente daquela que foi usada para implementar o cliente. O que é preciso respeitar é o mesmo "contrato" definido na interface IDL.
O compilador IDL para uma certa linguagem de programação além de gerar os "adaptadores" ("*stubs*") para o lado do cliente também gera uma classe "**esqueleto**" ("*skeleton*") que é usada como um "ponto de entrada" para a implementação do objeto CORBA. O "esqueleto" ("*skeleton*") faz a conversão ("*unmarshaling*") dos dados de

entrada em IDL para a linguagem específica de programação usada no servidor, chama o método que implementa a operação solicitada, providencia a "conversão" ("*marshaling*") dos dados a serem retornados para IDL e os retorna. O método que implementa a operação solicitada precisa ser codificado em uma linguagem que permita o "mapeamento" na IDL.
A implementação do objeto recebe uma solicitação através do "esqueleto" gerado para a invocação estática ou através do esqueleto gerado para a invocação dinâmica.
Um cliente que envie uma solicitação para um servidor não entra no mérito de como o servidor funciona nem considera o que estiver ocorrendo com o servidor no instante em que a solicitação é feita. Para permitir isto CORBA prevê um adaptador de objeto **OA** – ("*Object Adapter*") cujas funções são:

- geração e interpretação das referências de objeto;
- invocação de método;
- interações de segurança;
- ativação e desativação de objeto e implementação;
- mapeamento das referências de objeto nas implementações correspondentes do objeto;
- registro das implementações.

3.2 *Java*

As descrições aqui contidas são baseadas nas referências listadas a seguir – que devem ser consultados para mais detalhes: Ref. [6], Ref. [165], [2], Ref. [56], Ref. [107], Ref. [149], etc.

Java é uma linguagem de programação compilada e a seguir interpretada. Permite transparência para vários tipos de hardware e sistemas operacionais porque possui uma máquina virtual que permite que o mesmo código compilado possa ser executado em vários sistemas operacionais. Para viabilizar isto os programas são inicialmente compilados para gerar um código intermediário denominado de "***Java bytecode***" que é independente da máquina que o hospede. Este "*Java bytecode*" pode ser transportado para outra máquina para a seguir ser interpretado e gerar um código executável específico para a máquina em que o programa deve ser executado.

Java possui um conjunto de recursos muito grande, dentre outros:

- comunicação entre entidades de um sistema distribuído,
- acessos a base de dados,
- programação concorrente,
- confecção de **GUI** ("*Graphical User Interface*") – interface gráfica de usuário (**IGU**),
- definição de políticas de segurança para acessos aos recursos do sistema,
- compatibilidade com a interface IDL da CORBA,
- interação com "browsers" para confecção de páginas de uma intranet ou da Internet,
- etc.

3.2.1 Conceitos Básicos da Linguagem Java

Java é uma linguagem orientada a objetos. O **objeto** é o principal "*bloco*" usado em um programa de uma linguagem orientada a objetos. É uma abstração de programação para agrupar os dados com o código que opera nesses dados. O objeto consiste de dados e de

procedimentos (funcionalidades) que descrevem o recurso usado. Estes procedimentos são denominadas de **métodos**. Os dados dos objetos são denominados de **variáveis de estado**.

Classe é um modelo (gabarito) para um conjunto de implementações de objeto. Um objeto é caracterizado pela sua classe. Uma classe define a implementação de um certo tipo de objeto. Uma classe pode ser "particularizada" dando origem a uma subclasse. Diz-se que há **herança** entre tais classes porque a subclasse herda todas as características da classe superior da qual foi gerada. Java só permite herança simples de classes. Classes relacionadas podem ser agrupadas em um "**pacote**" ("*Package*"). Quando uma classe é gerada a partir de outra diz-se que a subclasse "*estende*" ("*extends*") a classe de nível superior.

Uma **instância** é um objeto de uma classe particular. Uma classe possui um nome que a identifica e consiste de um conjunto de campos de dados e métodos. Quando uma classe é definida não é alocado espaço de memória para ela porque ela apenas define as características de um certo objeto. Para ser alocado espaço de memória é necessário criar uma instância desta classe. O "**construtor**" é um método especial da classe que inicia o objeto quando ele é "instanciado", isto é, quando é alocado espaço de memória para ele e antes de ele ser usado.

Um conjunto de métodos (que podem ser implementados por várias classes) define a **interface** de um objeto. Uma interface em si não contém o código da implementação mas apenas a declaração dos métodos. Uma interface pode implementar ("*implements*") várias outras interfaces, ou seja, as interfaces podem ser herdadas de outras. Um objeto é conhecido externamente pela sua interface com os demais objetos.

Os erros são tratado através de **exceções**. A exceção permite tratar o erro, no local onde possa ocorrer, sem a necessidade de escrever um código complicado, alheio ao contexto principal do tema que esteja sendo tratado.

Para suportar programação concorrente são usadas "linhas de execução" ("***threads***"). Elas podem ter diferentes prioridades de ocupação dos recursos do equipamento. Para permitir o acesso ordenado aos recursos comuns do sistema, Java permite o uso de monitores e sincronização de "linhas".

Através de palavras-chave em um texto HTML é possível incluir um código escrito em Java que possa ser transportado através de uma intranet ou da Internet e ser executado em outra máquina com a ajuda de um "***browser***". Este código escrito em Java é denominado "***applet***". Uma applet possui um ciclo de vida que inclui carregá-la na máquina hospedeira, iniciar sua execução, parar a execução e destrui-la (destruindo ou saindo da página HTML).

Uma característica importante para o nosso contexto que é permitida pela linguagem Java é a capacidade de permitir o mapeamento de um código escrito em Java em um código escrito em outras linguagens. Faz-se um "invólucro" ("*wrapper*") escrito em Java que acesa o código escrito em outra linguagem de programação. Isto permite aproveitar códigos já existentes e, por exemplo, mapeá-los em um servidor Java. Atualmente este mapeamento é possível para "A*ssembly*", C e C++. Mais detalhes podem ser vistos em JNI ("*Java Native Interface*" – Interface Java para códigos "nativos") na Ref. [6].

Uma visão rápida, simplificada e parcial é dada aqui. Detalhes podem ser vistos na Ref. [6], na Ref. [56], na Ref. [107], na Ref. [149], etc.

As classes fundamentais Java – **JFC** (*"Java Foundation Class"*) são um conjunto de recursos para a construção de **IGU** – interface gráfica de usuário (***"graphical user interface"*** – **GUI**): **AWT** (*"Abstract Window Toolkit"*) – caixa de ferramentas de janelas abstratas, componentes Swing, suporte para configuração de "aparência" (*"look and feel"*), etc. Aqui nos centralizaremos no uso dos componentes **AWT** e **Swing** que foram necessários à construção das telas do protótipo: painéis, botões, menus, caixas de escolha de opções (*ComboBox*), etc.

A configuração de ***"look and feel"*** personaliza a aparência da IGU usada, já que ela é dependente do ambiente em que o programa for executado. Há 3 possibilidades previstas: Windows, CDE/Motif (Solaris) ou uma que é independente da plataforma e que foi definida como específica do ambiente Java.

Para o tratamento de uma IGU a linguagem Java permite um conjunto básico de recursos que incluem vários **componentes** software que podem ser ajuntados e/ou adaptados para atender as necessidades específicas do usuário.

Inicialmente foram definidas as classes dos componentes AWT e a seguir as classes dos componentes Swing. Todos estão organizados de uma forma hierárquica. O pacote de componentes denominado ***Swing*** foi incluído em Java para ampliar os componentes GUI que os componentes AWT permitem e para tornar os desenvolvimentos independentes das plataformas. O componentes *Swing* possuem funções e nomes semelhantes aos componentes AWT mas os nomes começam pela letra **J**: ***Jcomponent, JscrollBar, JscrollPane, JmenuBar, JmenuItem, Jmenu, JradioButton, Jlist, Jlabel, JcomboBox, Jbutton,*** etc.

Uma hierarquia parcial dos componentes Swing é listada na **Tabela # 3.3** – uma árvore com as principais classes para o nosso contexto.

A classe ***"Object"*** é em Java a classe de mais alto nível hierárquico. Toda classe tem esta classe como sua superclasse.

Um componente (instância da classe ***Component***) é um objeto que possa ser desenhado na tela através de uma representação gráfica, que possa interagir com o usuário e que tenha tamanho e posição.

Um ***"containner"*** é um componente que pode conter outros componentes. Java Swing contém 3 tipos de **container de alto nível**:
Jframe, Jdialog e Japplet.

Hierarquia de algumas classes Java usadas para interfaces gráficas de usuários

java.lang.**Object** ⇨ java.awt.**Component** ⇨ java.awt.**Containner**

... java.awt.**Container** ⇨ java.awt.**Window** ⇨ java.awt.**Frame** ⇨ javax.swing.**Jframe**
... java.awt.**Container** ⇨ java.awt.**Window** ⇨ java.awt.**Dialog** ⇨ javax.swing.**Jdialog**
... java.awt.**Container** ⇨ java.awt.**Window** ⇨ javax.swing.**Jwindow**
... java.awt.**Container** ⇨ java.awt.**Panel** ⇨ java.applet.**Applet**
... java.awt.**Container** ⇨ javax.swing.**Jcomponent** ...

... javax.swing.**Jcomponent** ⇨ javax.swing.**AbstractButton**⇨ javax.swing.**Jbutton**
... javax.swing.**Jcomponent** ⇨ javax.swing.**AbstractButton**⇨ javax.swing.**JmenuItem** ⇨ javax.swing.**Jmenu**
... javax.swing.**Jcomponent** ⇨ javax.swing.**AbstractButton**⇨ javax.swing.**JmenuItem** ⇨ javax.swing.**JradioButtonMenuItem**
... javax.swing.**Jcomponent** ⇨ javax.swing.**AbstractButton**⇨ javax.swing.**JtoggleButton** ⇨ javax.swing.**JcheckBox**
... javax.swing.**Jcomponent** ⇨ javax.swing.**AbstractButton**⇨ javax.swing.**JtoggleButton** ⇨ javax.swing.**JradioButton**
... javax.swing.**Jcomponent** ⇨ javax.swing.**JcomboBox**
... javax.swing.**Jcomponent** ⇨ javax.swing.**JinternalFrame**
... javax.swing.**Jcomponent** ⇨ javax.swing.**Jlabel**
... javax.swing.**Jcomponent** ⇨ javax.swing.**JlayeredPane** ⇨ javax.swing.**JdesktopPane**
... javax.swing.**Jcomponent** ⇨ javax.swing.**JmenuBar**
... javax.swing.**Jcomponent** ⇨ javax.swing.**Jpanel**
... javax.swing.**Jcomponent** ⇨ javax.swing.**JtabbedPane**
... javax.swing.**Jcomponent** ⇨ javax.swing.text.**JtextComponent** ⇨ javax.swing.**JtextArea**
... javax.swing.**Jcomponent** ⇨ javax.swing.text.**JtextComponent** ⇨ javax.swing.**JtextField**
... javax.swing.**Jcomponent** ⇨ javax.swing.**JtoolBar**

Tabela # 3.3 - Hierarquia de algumas classes Java usadas para interfaces gráficas de usuários

Java Swing possui também vários **containers intermediários**, dentre eles:
Jpanel, JscrollPane, JtabbedPane, JinternalFrame, etc.

Os programas que usem Swing devem conter pelo menos um "**container**" Swing **de alto nível** para que os componentes de uma GUI possam aparecer na tela. O container permite que os componentes tenham uma barra que delimite o contorno em que estão, possuam ter uma barra de menus, etc.

Um **container de alto nível** pode ser uma instância de ***Jframe, Jdialog, Jwindow*** ou ***Japplet***. Ver a árvore abaixo. Eles fornecem o "***framework***" ("quadro" ou "padrão") em que os demais componentes existirão. Cuidam do tratamento de eventos, tratam o desenho dos componentes na tela ("*painting*") e usam **"gerentes de formato" ("*layout managers*")**. Estes servem para controlar o tamanho e a posição do componente dentro de um containner, de forma que os objetos sejam posicionados dentro do container de uma forma definida. O "*layout manager*" determina os tamanhos máximos, mínimos e preferenciais, os alinhamentos, as posições dos componentes no container, o espaçamento entre os componentes que estiverem contidos no container, etc.

Há vários "*layout manager*" predefinidos em Java (*BorderLayout, FlowLayout, CardLayout, GridLayout, GridBagLayout*). Também é possível criar um "*layout manager*" específico para as necessidades do usuário.

A título de exemplo, um mesmo conjunto de botões pode ser disposto de formas diferentes conforme o gerente de formato que estiver sendo usado. Um gerente de formato do tipo fluxo ("*FlowLayout*") disporá os botões lado a lado. Um gerente de formato do tipo grade (*GridLayout*), disporá os botões dentro de uma tabela de n colunas por m linhas.

Entre os componentes AWT há alguns que é necessário citar para facilitar o entendimento dos componentes Swing.

Uma janela de alto nível sem bordas é chamada de "***window***". Permite gerar eventos que assinalem que a janela foi aberta, fechada, minimizada, maximizada, etc.

Um "***Frame***" é uma janela de alto nível que possui um título, bordas, uma barra de escolha de opções ("***menubar***").
Se a janela permite a entrada de dados do usuário é denominada de "***dialog***".
Para a escolha de um arquivo é usado um componente AWT denominado "caixa de diálogos de arquivo" ("***FileDialog***").
Um **painel** ("***Panel***") permite a criação de "*layouts*" mais elaborados.
Um botão ("***Button***") permite que uma ação seja definida e executada quando o botão for "*clicado*" pelo mouse.
Para a escolha de uma entre duas opções é possível usar uma "***caixa de seleção***" ("***Checkbox***").
Quando se seleciona uma opção a outra fica automaticamente excluída. Para selecionar entre várias opções pode ser usada uma lista rolante de caracteres ("***List***").
Uma área de texto que pode ser editada é denominada área de texto ("***TextArea***").
Um texto com uma só linha é denominado campo de texto ("***TextField***").
Uma barra de rolagem ("***Scrollbar***") é usada para rolar uma imagem horizontal ou verticalmente na tela.

Além das classes já citadas acima as demais classes mostradas na Tabela 4.3 e descritas a seguir são todas classes Swing usadas para permitir adicionar componentes aos containers.

Com exceção dos **containers de alto nível**, todos componentes que começam com J herdam as características da classe ***Jcomponent***, que permite:

- combinar componentes para criar outros componentes;
- usar "dicas" ("***tool tips***") – uns textos que explicam a função do componente e que aparecem quando se para o mouse perto de um componente;
- usar teclas para manipular as opções das telas sem o uso do *mouse*;
- definir bordas que definem o conteúdo do componente;
- ter a aparência das telas ("***look and feel***") configurável;
- definir os tamanhos mínimo, máximo e preferido para um componente; etc.

Jframe é uma extensão de *Frame* que permite adicionar ao seu conteúdo um novo objeto "filho" ("*child*") e gerenciar a sobreposição dos objetos. Cada objeto ***Jframe*** implementa uma única janela principal de alto nível. É um lugar em que outros componentes Swing podem ser desenhados. Possui uma borda, um título e botões para fechar e minimizar o "*frame*". Quando um usuário fecha um "*frame*" a aplicação é terminada.

Um objeto ***Window*** é uma janela de nível alto, que permite gerar eventos que assinalem a criação ou o fechamento da janela. Não tem bordas nem barra de menu. Precisa , obrigatoriamente, ter definida como seu usuário ou um "frame", ou um diálogo ou outra janela.

Jdialog implementa uma janela secundária para interação com o usuário: envio de mensagens de advertência, solicitação de senha *("password")* ou solicitação de entrada de dados. Estas podem ser feitas via texto, caixas de seleção do tipo "caixa de seleção" ("*Check Box*"), caixa de combinações ("*Combo Box*") ou botões. É usado para construir uma janela que seja dependente de outra, isto é, que desapareça quando a outra janela é minimizada.

Japplet é usada quando se usa uma *applet* em um *"browser"*. Estaremos usando somente aplicações Java e *Japplet* não será considerada aqui.

Para permitir que os componentes de um container possam se sobrepor em várias camadas, uns sobre os outros, é usada a classe ***JlayeredPane.***

JinternalFrame é um tipo de container intermediário onde se adiciona os componentes desejados. Através dela é possível colocar uma janela dentro de outra. Permite que um frame possa se tornar um ícone, ser arrastado, aberto, mudar de tamanho, ter um título e suportar uma barra de menu. Em geral se criam instâncias de objetos que são adicionadas a ***JdesktopPane***.

JdesktopPane estende ***JlayeredPane*** e permite gerenciar as sobreposições internas. Os objetos *JinternalFrame* são criados e adicionados a *JdesktopPane* usando primitivas do tipo adicionar ("*add*").

Jpanel é um painel que faz o papel de um **container intermediário**. É usado junto com um "*layout manager*" para dividir uma área em partes menores que sejam controladas por "*layout managers*" específicos. É usado para agrupar componentes que estejam relacionados entre si ou que, por serem agrupados tornem o tratamento do layout mais fácil. Nele podem, por exemplo, estar contidos um *Jbutton* e um *Jlabel*, de forma que o *Jpanel* seja usado para simplificar o posicionamento do botão e do *label* entre si. Um painel com vários componentes pode ser atribuído como o conteúdo de um *frame*.

JtabbedPane também é um *container* intermediário. Ele permite a seleção entre vários componentes que "disputam" o mesmo espaço na tela. Esta seleção é feita através de "*etiquetas*" ("***tabs***") que identificam a escolha feita e mostra somente um componente por vez.

Outros *containers* intermediários (***JscrollPane***, ***JsplitPane***, etc.) podem ser vistos na Ref. [6], na Ref. [107] e na Ref. [149].

Jlabel é uma linha única que pode conter um ícone e um texto ("rótulo") fixos, isto é, que não podem ser modificados. Pode ser usado para várias aplicações, desde a identificação de qualquer componente até o envio de uma mensagem de advertência ao usuário.

Jbutton permite associar a um botão: um texto, um mnemônico que o usuário pode digitar para escolher o botão e o código que será usado como um tratador de eventos associado ao

botão. Por exemplo, se o mnemônico é ***b***, digitando ***Alt-b***, o botão é selecionado como se tivesse sido pressionado pelo mouse e um código construído em um tratador de evento é acessado para executar a ação associada ao botão.

Uma "***caixa de seleção***" do *Swing* ("***Jcheckbox***") permite a escolha de uma entre duas opções e é semelhante a uma "***caixa de seleção***" do AWT ("***Checkbox***").

A caixa de combinações do Swing ("***JcomboBox***") permite a escolha entre um conjunto de opções de uma lista de textos e permite adicionalmente, digitar um texto que será usado como a opção escolhida. Quando o usuário digita a primeira letra de um texto que exista como uma opção de escolha na caixa de combinações a opção aparece em destaque, pronta para ser selecionada pelo "*click*" do mouse, facilitando a entrada de dados.

Um "***conjunto de botões de rádio***" do *Swing* ("***JradioButton***") permite selecionar uma entre várias opções, "*clicando*" com o mouse na opção desejada.

Vários componentes Swing podem ser adicionados aos containers através de primitivas de adição ("*add*").

Qualquer objeto pode ser notificado sobre a ocorrência de um ***evento***. Para isto, tal objeto deve implementar a interface apropriada e estar registrado na fonte responsável pela geração do evento ("***event source***") como um "*escutador do evento*" ("***event listener***").
Vários tipos de eventos podem ser gerados pelos componentes Swing. Por exemplo, quando um botão é selecionado, um "*event listener*" do tipo "***ActionListener***" deve ser usado. Cada evento é representado por um objeto que dá informação sobre o evento e identifica a fonte do evento. Uma mesma fonte de eventos pode ter vários "*event listeners*" associados a ela. Os tratadores de eventos podem ser instâncias de quaisquer classes.

3.2.2 Conceitos Essenciais de HTML e CGI Usados em Java RMI

Para entender melhor alguns detalhes da RMI é necessário definir antes alguns conceitos básicos de HTML e CGI.
Para que um browser possa entender e tratar um texto HTML ("*Hypertext Markup Language*") são usadas "palavras-chaves" denominadas de "elementos" e seqüência de caracteres que não ocorrem em um texto comum, embutidas juntas ao texto ASCII comum ("útil") a ser exibido pelo browser. Também usam uma sintaxe específica da HTML para serem diferenciadas do texto a ser mostrado pelo browser.

Quando se usa um formulário em uma página HTML, um "*script*" CGI é usado para tratar as informações fornecidas através deste formulário para, por exemplo, providenciar a leitura ou a escrita em uma base de dados, o envio de dados por *e-mail*, etc.

Para a entrada de dados em um formulário HTML é usado um elemento denominado FORM que é responsável por definir todas as ações do formulário. A seguinte sintaxe é usada:

<FORM ACTION= ...(*1)... METHOD=...(*2)... TARGET=...(*3)... > ... </FORM>

onde:

(*1) = endereço do programa CGI que receberá os dados do formulário; o endereço deve estar no formato URL, isto é, por exemplo:

http://www.xxx.yyy/zzz/kkk/yyy/jjj/programa.cgi

(*2) = parâmetro que define como os dados serão transmitidos para o programa CGI que os tratará; há duas possibilidades, conforme a forma de empacotamento dos dados: GET e POST;

(*3) = especifica o *"frame"* em que uma resposta será colocada, quando tal frame é diferente daquele em que estiver o formulário.

Esta explicação está baseada na Ref. [134] – que pode ser consultada para mais detalhes sobre HTML.

3.2.3 Invocação Remota de Método (Java RMI)

Java suporta tanto "***sockets***", quanto chamada de procedimento remoto (***RPC***), bem como invocação remota de método (***RMI***).
"***Socket***" é um "*ponto lógico*" ("*end point*") para a comunicação entre duas máquinas.
"Socket" exige o detalhamento da máquina de estados do protocolo a ser usado bem como a codificação e a decodificação das mensagens envolvidas.
RPC também não é eficiente para invocar diferentes objetos que residam em diferentes endereços. Para tal necessidade é mais eficiente usar ***RMI***.
O protocolo IIOP é usado como o protocolo de transporte para Java RMI – ver Ref. [108] e Ref. [50]. Java também permite definir um protocolo específico para ser usado sobre o protocolo IP com RMI. Mais detalhes sobre isto podem ser vistos na Ref. [147] na abordagem ali feita sobre criação de "*fábrica de socket*" RMI que pode ser adaptada para as necessidades específicas do usuário ("*RMI Socket Factories*").
Algumas características da RMI são descritas a seguir. As Ref. [147] e [6] – Java Tutorial – devem ser consultadas para uma descrição completa.

A RMI provê um mecanismo para comunicação entre programas escritos em Java. A linguagem de definição de interfaces (IDL da CORBA) deve ser usada quando os programas estão escritos em linguagens diferentes. No entanto, é possível usar RMI para linguagens diferentes, desde que sejam usados "adaptadores" ("***wrappers***") entre os programas escritos na outra linguagem e os escritos em Java.
RMI pressupõe o uso de um mecanismo padrão também usado nos sistemas RPC para a comunicação com os objetos remotos: objetos locais ("***stubs***") encarregados de gerenciarem a invocação dos objetos remotos. Em versões anteriores ao JDK1.2 cada stub está associado a um objeto remoto ("***skeleton***") que é responsável para "encaminhar" a chamada à implementação efetiva do objeto remoto. No JDK1.2 é usado um código genérico para desempenhar as funções do skeleton.
RMI permite chamadas de retorno ("***callbacks***") de um servidor para uma applet.
As aplicações RMI em geral são compostas de 2 programas separados: um servidor e um cliente.
Um servidor típico, cria alguns objetos remotos, torna as referências a estes objetos acessíveis e espera que os clientes invoquem os métodos destes objetos remotos.
Uma aplicação pode passar e retornar referências de objetos ou usar um recurso de registro de nomes aos objetos ("***rmiregistry***").
Um objeto pode ser associado a um nome usando os métodos "***bind***" ou "***rebind***".

O método *"bind"* não associa um nome a um objeto se o nome já estiver associado a um objeto. O método *"rebind"*, no entanto, faz uma nova associação, mesmo que uma outra já exista.
Um cliente pode obter a referência de um objeto remoto (***"lookup"***) e invocá-lo usando seu nome.
Uma aplicação de cliente típico obtém uma ou mais referências para objetos remotos no servidor e invoca métodos neles.
A RMI provê os mecanismos para o cliente e o servidor comunicarem entre si e passarem informações nos dois sentidos. Tal aplicação é às vezes chamada de aplicação de objeto distribuído. Para isto os objetos são transformados, de forma transparente para o programador, para um formato adequado para a sua transferência em série entre cliente e servidor. Para mais detalhe, ver na Ref. [6], *"Serialização"* de objetos em Java, classe ***"java.io.Serializable"***.
Os detalhes da comunicação entre os objetos remotos são tratados pela RMI de uma forma transparente para o programador. Para este, a comunicação remota se apresenta como uma invocação padrão de um método da Java.
A RMI provê os mecanismos para um cliente passar os objetos Java para um servidor remoto, transmitir os dados e carregar os códigos do objeto.
A camada de transporte da RMI normalmente tenta abrir sockets diretas com as máquinas hospedeiras na Internet. Quando são usados ***"firewalls"*** isto não é permitido. Alternativamente é possível usar dois mecanismos baseados em HTTP para permitir que um cliente atrás de um *firewall* invoque um método em um objeto remoto que resida for a do *firewall*. Os dados da chamada RMI são enviados para for a no corpo de uma solicitação HTTP POST e a informação de resposta é envida de volta no corpo de uma resposta HTTP.
A solicitação POST é feita de duas formas, conforme a máquina do *firewall* (***"firewall proxy"***) encaminhe a solicitação HTTP para uma porta arbitrária ou para uma porta específica na máquina hospedeira. Se for usada porta arbitrária, a solicitação HTTP é encaminhada diretamente para uma porta em que um servidor RMI está à escuta usando uma *socket* capaz de entender e decodificar chamadas RMI dentro de solicitações POST.
Se for usada porta específica, a solicitação HTTP é encaminhada ao servidor HTTP que está à escuta na porta 80. Um ***"script"*** CGI será executado para encaminhar a chamada para a porta do servidor RMI na mesma máquina.
As chamadas transmitidas via solicitações HTTP são, no entanto, bem mais lentas que as feitas via *sockets* diretas.
A invocação de um método em um objeto remoto segue a mesma sintaxe da invocação de um método em um objeto local.
A RMI permite especificar um URL (***"codebase"***) a partir de onde as classes a serem usadas devem ser carregadas (***"downloaded"***).
A localização de um servidor é feita através de seu endereço IP (***"hostname"***).
É possível especificar um gerente de segurança (***"RMISecurityManager"***) para definir uma política de segurança (ver detalhes no item específico mais adiante).
O protocolo de multiplexação da RMI usa uma conexão TCP. A Ref. [147] deve ser consultada para detalhes a respeito.
A RMI torna a programação mais simples e menos passível de erros para o programador porque oculta detalhes do protocolo efetivamente usado para interações com os objetos remotos e proporciona as demais facilidades inerentes à plataforma Java.

Visão Geral da RMI

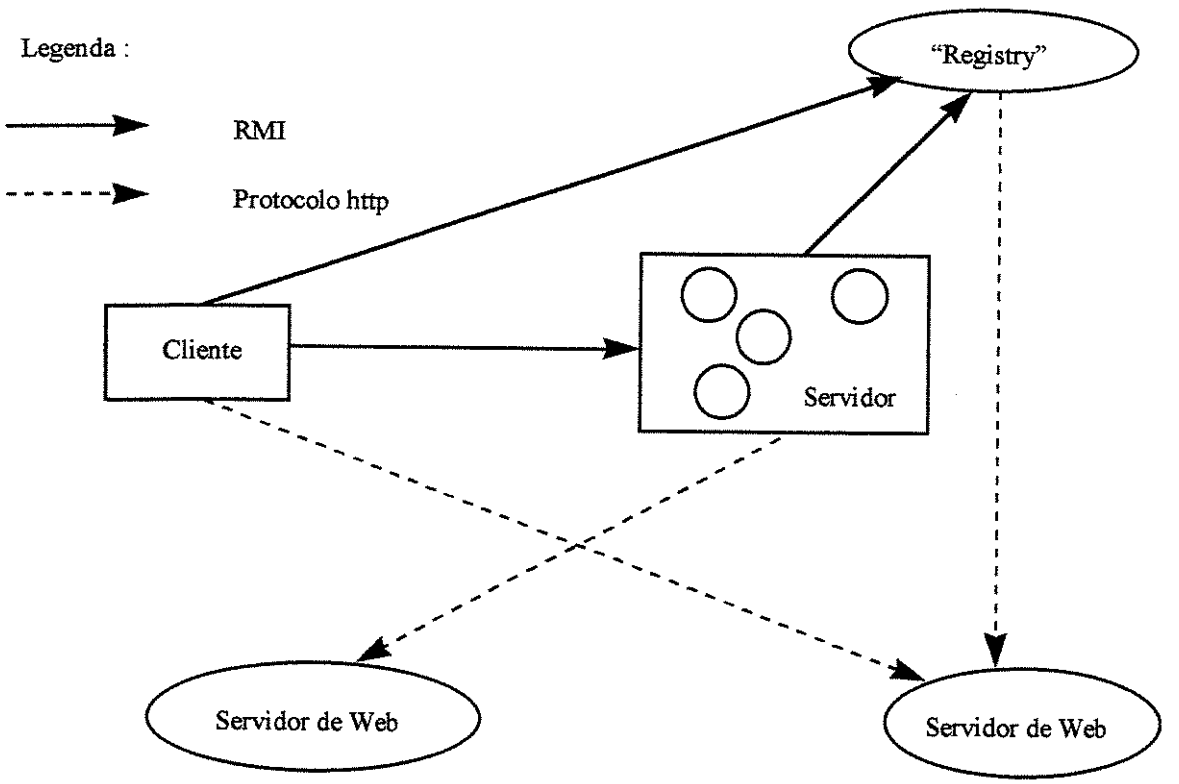


Figura Extraída da Referência [L.6]

Figura # 3.2 - Visão Geral da RMI

A **Figura # 3.2** mostra uma visão geral do funcionamento da RMI.

Um servidor de web é usado para carregar "bytecodes" de classes Java do servidor para o cliente e do cliente para o servidor usando qualquer protocolo URL: HTTP, FTP, file, etc.

Mais detalhes podem ser encontrados na Ref. [147] e na Ref. [6] – em que a presente descrição está baseada.

"Feixes" de dados ("*Streams*")

A linguagem Java provê uma API para a leitura e a escrita de dados em série de e para qualquer lugar (arquivo, disco, memória, rede, outro programa) em "feixes" de dados (que podem ser tanto bytes quanto caracteres ***Unicode*** de 16 bits). Isto será usado de forma transparente pela RMI para a comunicação entre o cliente e o servidor. Os caracteres Unicode são representados em 16 bits, estão definidos pela ISO 10646 e são úteis para suportar os mais variados idiomas possíveis. Isto é interessante para aproveitar o suporte que Java provê para os diferentes idiomas.

A comunicação entre o cliente e o servidor

Os objetos remotos são essencialmente passados por referência. Uma referência remota de um objeto é realmente um *"stub"*, isto é, um *"proxy"* do lado do cliente que implementa um conjunto completo de interfaces remotas que o objeto remoto implementa.
Qualquer mudança feita no estado do objeto através de chamadas remotas de métodos são refletidas no objeto remoto original.
Os objetos locais são passados por cópia usando o mecanismo de "**serialização**" de objetos da Java.
A classe "**java. Rmi. Server.UnicastRemoteObject** " é uma classe que a Java RMI provê para poder ser estendida pelas implementações remotas para facilitar a criação dos objetos remotos. Os "construtores" da classe permitem criar e exportar um objeto novo usando uma porta anônima ou uma porta específica. Esta classe define objetos remotos únicos que suportem comunicação ***"unicast"*** (comunicação ponto-a-ponto), que podem ser referenciados enquanto o processo no servidor remoto estiver ativo e que usem o mecanismo default da RMI (baseado em sockets). Ela provê todo o suporte para as referências ponto-a-ponto entre objetos ativos. As invocações, os parâmetros e os resultados usam um protocolo TCP de "feixes" de dados para as comunicações entre o cliente e o servidor. Este mecanismo é usado para o transporte de toda a comunicação e permite aceitar solicitações de chamadas dos clientes em uma porta qualquer.

Tratamento de Exceções

Para atender os casos em que ocorra uma exceção durante uma invocação remota de método é usada a classe **java.rmi.RemoteException.** Tal exceção atua quando a invocação remota de método falha por algum motivo como , por exemplo:

- falha de comunicação:
    - o servidor remoto não pode ser encontrado;
    - o servidor remoto não está aceitando conexões;
    - a conexão foi encerrada pelo servidor;
- falhas durante o manuseio dos valores dos parâmetros e resultados ("**marshaling**" / "**unmarshaling**")
- erros de protocolo;
- quando não hajam recursos de comunicação disponíveis ou a classe "stub" adequada não for encontrada, etc.

A política de segurança

Uma classe ***java.rmi.RMISecurityManager*** provê a política de segurança default para as aplicações que usem RMI. O "***security manager***" provê a proteção contra acessos indevidos aos recursos do sistema. Define a que recursos o novo código que foi "carregado" ("*downloaded*") terá acesso (sistema local de arquivos, propriedades do sistema, aceitar conexões, etc.) e que tipo de operação terá direito de executar em tal sistema (leitura, escrita ou nenhuma operação).

A obtenção de referências de nomes para os objetos remotos

É feita pela classe ***java.rmi.Naming***.

É baseada na sintaxe de **URL** (*"Uniform Resource Locator"*).

O URL de um objeto remoto é especificado usando a sintaxe

//host:porta/nome

onde:

- *"host"* é o nome do equipamento (local ou remoto) onde se faz o registro do nome (*"registry"*);
- **porta** é o número da porta (*"port"*) em que o registro aceita chamadas;
- **nome** é o nome do objeto remoto no registro.

Dentre os métodos existentes (*bind, rebind, unbind, lookup e list*) usaremos os listados abaixo.

- ***rebind*** (nome, objeto): "registra" (*"bind"*), no registro remoto, o nome do novo objeto sobrepondo-o sobre qualquer outro nome eventualmente já existente.
- ***lookup*** (nome): retorna, desde o registro remoto até o URL do cliente, o nome do objeto remoto que foi previamente "registrado".

O ***rebind*** é necessário para que um cliente possa obter a referência de um objeto remoto através de seu nome. Para isto o servidor faz o registro prévio do nome do objeto remoto através de um serviço remoto de nomes de objetos (*"RMI registry"*). Uma vez que o objeto remoto tenha sido registrado através do "registro RMI" no equipamento local ou remoto (*"host"*), um cliente de qualquer outro equipamento (*"host"*) pode obter a referência do objeto remoto (*"lookup"*) através de seu nome e então invocar o método remoto deste objeto.

O nome do objeto inclui o nome do equipamento (*"host"*) no qual o registro (*"registry"*) está sendo executado e o nome do objeto remoto que foi previamente registrado no *"registry"*. Quando o nome é adicionado ao *"registry"* através de um *"rebind"*, o primeiro parâmetro deste método contém um *"string"* com o nome formatado em URL do objeto remoto no *"registry"* e o segundo parâmetro contém uma referência para o objeto remoto (um *"stub"*) que está sendo registrado.

3.3 UML

3.3.1 Conceitos Gerais

Para o desenvolvimento de um software é necessário prever todos os estágios do ciclo de vida do software, o que inclui garantir a uniformidade da informação entre diferentes pessoas em diferentes intervalos de tempos. Há muitos outros aspectos que devem ser considerados: custo, funcionalidade, compatibilidade com outros desenvolvimentos, segurança a falha, desempenho, duração do produto desenvolvido, tamanho e localização da equipe de desenvolvimento, etc.

As técnicas de orientação a objetos reúnem um conjunto de boas práticas para serem seguidas. A **UML** (*"Unified Modeling Language"*) – Linguagem Unificada de Modelagem – é uma linguagem padronizada pela OMG e usada para especificação, visualização, desenvolvimento e documentação de software.

Alguns conceitos da UML estão rapidamente apresentados a seguir. Mais detalhes podem ser obtidos na Ref. [68], na Ref. [133], na Ref. [162], etc. em que este texto está baseado. Uma descrição completa pode ser vista na Ref. [122].

A UML é uma linguagem aberta que suporta todo o ciclo de desenvolvimento de software em diversas áreas de aplicações. A partir de uma especificação em UML é possível gerar, automaticamente, o código associado em uma certa linguagem de programação. Várias ferramentas a suportam. É baseada na experiência e nas necessidades de diversos tipos de usuários. Ela acumula os resultados de vários estudos na área de orientação a objetos. Em Novembro de 1997 a OMG publicou a sua 1ª. versão oficial – a UML 1.3.

Atores (“***Actors***”) representam alguma coisa ou alguém que pode interagir ou trocar informação com um sistema. Para o nosso protótipo podemos identificar um ator que irá representar as interações humanas com o protótipo, isto é, o **operador**. Podemos também identificar um outro ator que irá interagir com um sistema externo, isto é, a **base de dados** da central telefônica.

Um sistema deve atender todas necessidades de seus usuários. Estas necessidades são representadas como **casos de uso** (“***use cases***”) – um padrão de comportamento que o sistema possua. Cada um é uma seqüência de transações feitas por um ator e pelo sistema em um diálogo. O conjunto de casos de uso especificam todas as formas de uso do sistema. Para um protótipo que pretenda representar as necessidades de um operador para fins de comunicação homem-máquina em centrais públicas de comutação telefônica poderíamos identificar vários casos de uso, em várias áreas de atuação. Alguns exemplos de casos de uso dentre as várias opções possíveis são:

- tratar aspectos de tarifação:
  - obter contadores de assinantes,
  - obter registros de chamadas de longa distância,
  - definir a periodicidade de obtenção dos dados de tarifação,
- tratar sinalização número 7 (SN7):
  - tratar enlace de SN7,
    - criar enlace de SN7,
    - remover enlace de SN7,
    - ver dados de enlace de SN7,
  - tratar enumeração de CICs,
  - tratar rota de SN7,
  - tratar ponto de SN7,
  - tratar contadores estatísticos de SN7,
- tratar dados de tráfego,
- tratar encaminhamento,
- tratar outros dados.

Cada um dos exemplos acima é um caso de uso. A título de exemplo, para o nosso protótipo abordaremos com mais detalhes somente o caso de uso relacionado à criação do enlace de SN7.

Podem ser adicionadas **relações** (“***relationships***”) em um diagrama para mostrar as interações entre os atores e os casos de uso. Os casos de uso podem usar recursos de outros casos de uso ou podem ser estendidos por outros casos de uso. Relações especiais entre

casos de uso podem ser de inclusão ("***include***") ou de extensão ("***extend***"). Estas relações são exemplos de **estereótipos** ("***stereotypes***") – propriedades que podem ser adicionadas a módulos de elementos simples para lhes dar um significado especializado.

Para viabilizar um caso de uso os atores interagem com os objetos do modelo.

Os **diagramas de colaboração** ("***collaboration diagrams***") são representações gráficas que podem ser usadas, dentre outras aplicações, para mostrar as interações entre as instâncias dos objetos. Há duas formas: diagramas de seqüências e diagramas de colaboração (propriamente ditos). Estes últimos são também denominados em algumas bibliografias de **diagramas de interação**.

Um **diagrama de seqüência** ("***sequence diagram***") mostra as interações de objetos através de uma seqüência temporal. Os retângulos representam **objetos**. As linhas pontilhadas verticais abaixo dos retângulos representa **tempo** e o conjunto de eventos ou solicitações feitas entre os objetos. As setas representam **mensagens** entre os objetos e mostram as funcionalidades dos casos de uso. As interações de objetos necessárias para a criação de um novo enlace de SN7 pode ser vista no diagrama abaixo.

UML - Diagrama de Seqüência (parcial)

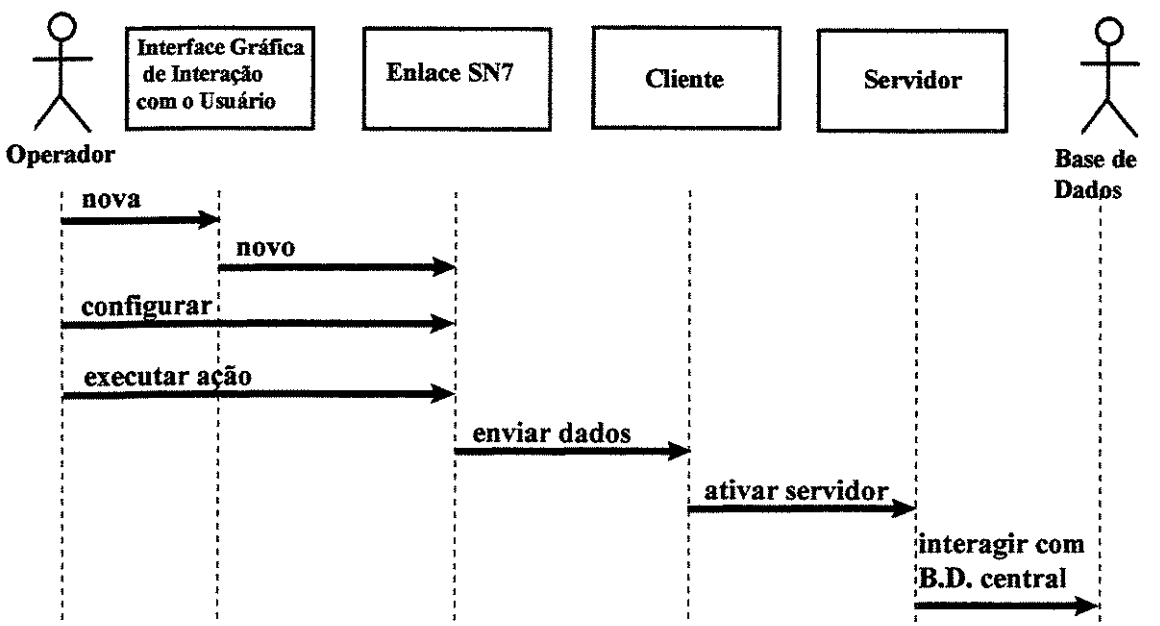


Figura # 4.1 - UML – Diagrama de Seqüência (parcial)

Os objetos do nosso diagrama de seqüência para a criação de um ESN7 são usados para criar os seguintes objetos: interface gráfica do usuário (IGU), enlace de sinalização N7 (ESN7), cliente e servidor.

Os **diagramas de interação** não mostram as relações temporais entre os objetos mas mostram como são feitas as interações entre os objetos e as interligações entre eles. Nos diagramas de colaboração os objetos são mostrados como ícones. As mensagens enviadas

entre os objetos do caso de uso representado são representadas por setas (tal como no diagrama de seqüência).

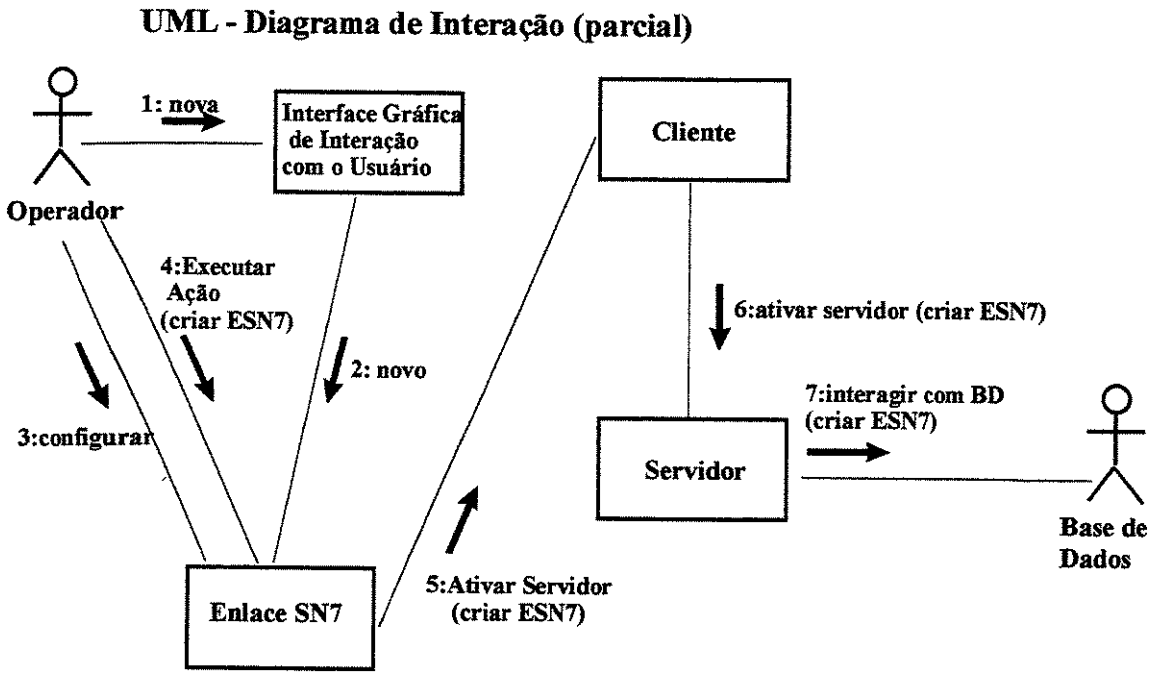


Figura # 4.2 - UML - Diagrama de Interação (parcial)

As relações ("***relationships***") fornecem caminhos de interações entre os objetos que são esboçados como linhas entre as classes. Um tipo de relação é a **associação** ("***association***"). As associações entre classes são modeladas como relações bidirecionais entre módulos. Os diagramas de seqüência e de colaboração são examinados para determinar que ligações entre objetos deve existir para possibilitar o comportamento desejado. Se dois objetos precisam interagir deve haver uma ligação entre eles.

Um dos objetivos dos diagramas de classes é a comunicação. Um tipo de elaboração de comunicação é denominado de papel ("***role***"): identifica o propósito ou a capacidade de uma classe ser associada a outra classe. O nome do papel é colocado ao longo da linha de associação junto com a classe que ele modifica.

Uma **agregação** ("***aggregation***") é uma forma especial de associação. Ilustra a relação entre um papel e sua parte. No nosso exemplo, o ESN7 é parte da IGU. Esta associação é especificada através de uma relação de agregação.

Uma vez estabelecidas as relações de associação e de **agregação** é necessário definir quantos objetos irão participar nas relações. **Multiplicidade** ("***multiplicity***") é o número de instâncias de uma classe relacionada com uma instância de uma outra classe. Para cada associação e agregação há duas decisões de multiplicidade, uma para cada lado da relação.

Embora a agregação e a associação sejam bidirecionais, quando é necessário restringir a navegação em somente uma direção uma seta é usada para explicitar isto. Por exemplo, a

IGU fala com o ESN7 mas o ESN7 não fala com a IGU. O servidor e o cliente falam um com o outro.

Os **diagramas de classes** (“***class diagrams***”) também representam a **estrutura** e o comportamento de cada classe. A estrutura de uma classe é descrita por um conjunto de atributos. Um **atributo** é uma definição de dados mantida pelas instâncias das classes. Os atributos são mostrados no 2° compartimento separado do ícone que representa a classe no diagrama da classe. O **comportamento** (“***behaviour***”) da classe é representado por um conjunto de operações. Uma **operação** é um serviço que pode ser solicitado por um objeto que possua tal comportamento. As operações são mostradas no 3°. compartimento do ícone que representa a classe no diagrama da classe.

Um outro tipo de relação é a relação de **herança**. A herança define as relações com classes onde uma classe compartilha a estrutura e o comportamento de uma ou mais classes. Define uma hierarquia de abstrações onde uma subclasse herda as características de uma ou mais super classes.

O comportamento dinâmico de um objeto pode ser representado por **estados**. Uma **transição** é a mudança de um certo estado para outro como resultado de algum estímulo. Uma transição pode ser automática ou pode ser rotulada por um evento, condição e/ou ação. No caso de uma transição rotulada por um evento só haverá mudança de estado se ocorrer o evento e se a condição de guarda for atendida.
Existem dois estados especiais em um **diagrama de transição de estados** (“***state chart diagram***”): o estado inicial e o estado de final ou de parada. Só há um estado inicial mas podem existir vários estados finais.
Os estados podem ser encadeados . O encadeamento de classes simplifica os diagramas complexos. Não há limites para o número de estados encadeados em um diagrama.
Uma atividade é desempenhada pelo comportamento de um objeto quando o objeto está em um certo estado. Uma atividade pode ser uma ação interna ou um evento enviado para um objeto externo.

Os **componentes de código fonte** mostram dependências entre arquivos. Este tipo de dependência é dependente da linguagem.
Os componentes de tempo de execução mostram o mapeamento das classes para as bibliotecas de tempo de execução tais como applets Java, componentes Active-X e bibliotecas dinâmicas.
Os **componentes executáveis** mostram as interfaces e dependências de chamadas entre componentes executáveis.
Os **elementos de processamento tempo de execução** (“***run time processing elements***”) são representados como nós. Os nós são conectados por associações que indicam caminhos de comunicação entre eles. Os processos de software são ilustrados como textos anexados aos nós ou grupos de nós
Um **pacote** (“***package***”) é um container de elementos de modelados. Cada pacote pode conter um conjunto de casos de uso, classes ou componentes. Linhas pontilhadas são usadas para mostrar dependências entre pacotes.
Os **estereótipos** (“***stereotypes***”) podem ser usados para aumentar o vocabulário da UML. Eles permitem a criação de novos elementos de “modelagem” criados a partir dos já existentes. Os rótulos “***includes***” (**inclui**) e “***extends***” (**estende**) podem ser usados para refinar as associações entre casos de usos. Outros estereótipos:

- "***boundary***" (**extremo** ou **limite**): uma classe que representa a interface com o mundo externo;
- "***control***" (**controle**): uma classe responsável por informações de sequenciamento;
- "***entity***" (**entidade**): uma classe que é uma abstração de alguma classe do mundo real;
- "***exception***" (**exceção**): classe que representa uma condição excepcional, um erro sério;
- "***interface***" (**interface**): usado para especificar operações que podem ser vistas externamente.

O **diagrama de atividade** ("***activity diagram***") representa o comportamento dinâmico do sistema. São fluxogramas que mostram o fluxo de controle de uma atividade para outra onde cada atividade representa uma operação sobre alguma classe no sistema. As transições são representadas como setas direcionadas que mostram a passagem do fluxo de controle de atividade para atividade. Certas atividades podem ocorrer em paralelo. Uma barra de sincronização – que é uma linha grossa horizontal ou vertical – é usada para indicar a bifurcação ("***forking***") e o ajuntamento de fluxos de controle paralelos. Ramos de decisão são representados por losango e são usados quando estiver sendo feita a modelagem de fluxos de controle de um sistema onde o fluxo de controle flui em função de um ponto de decisão.

Mais detalhes sobre a UML podem ser obtidos na Ref. [122].

3.3.2 Rose

O Rose é uma plataforma desenvolvida pela Rational que conta com a parceria de várias empresas e é largamente usada no mundo para atender diferentes complexidades de projeto. É uma ferramenta de modelagem visual, usa UML e permite o desenvolvimento interativo do software, conduzido por uma equipe de tamanho variável e localizada em diferentes lugares. Permite atender às necessidades do projeto em todos seus estágios, inclusive em aspectos de "reengenharia". As especificações são feitas de forma visual, gráfica e interativa com a plataforma Rose, através de UML. A seguir são convertidas automaticamente pela Rose em código de uma linguagem de programação específica. Atualmente já é possível fazer tal tipo de desenvolvimentos para as seguintes linguagens de programação: Visual Basic, Visual C++, Java, Forte, PowerBuider e Ada. No que diz respeito tanto à manutenção quanto à "reengenharia" de um projeto de software, apresenta facilidades muito úteis como, por exemplo, um código já escrito ou modificado em uma das linguagens de programação atendidas pelo Rose pode tanto gerar quanto atualizar automaticamente as documentações de especificação. Detalhes podem ser vistos na referência Ref. [133] e na página "web" da Rational. Segundo a Ref. [133] já há mais de 50.000 usuários desta plataforma no mundo, nas mais diferentes áreas de aplicação; muitas das grandes empresas de telecomunicações têm usado esta plataforma.

3.4 Comentários

CORBA e JAVA foram escolhidas porque o enfoque da implementação está na implementação do cliente e principalmente na definição da interface entre o cliente e o servidor. Sob o ponto de vista do tipo de implementação a adotar para a interação com a BD das centrais telefônicas, CORBA e JAVA possuem características divergentes, isto é, ou se opta por uma implementação que use CORBA e IDL ou por outra que use RMI. Sem dúvida que são duas coisas diferentes, mas ambas mostram que há mais de uma forma de

resolver a interação com as BDs das centrais telefônicas, cada uma com vantagens e desvantagens, como será abordado no capítulos seguintes .
É também importante observar que CORBA e JAVA apresentam características que podem ser complementares. IDL permite definir uma interface padrão, com independência de hardware, de sistema operacional e de linguagem de programação. Como IDL somente define a interface, para haver uma solução efetiva para o problema é necessário uma linguagem de implementação que use IDL. Java permite isto, provê recursos vantajosos para a construção da IGU e permite independência de platataforma. No entanto, JAVA apresenta o inconveniente da ineficiência do código gerado, de forma que é importante ressaltar que outros recursos teriam que ser considerados para tratar de forma mais adequada o servidor. Assim, para tal objetivo, deveriam ser consideradas outras linguagens de programação mais eficientes e eventualmente, linguagens específicas de interação com BDs tal como SQL.
UML é uma ferramenta eficiente para permitir desenvolvimentos de software real, profissional. Nesta dissertação é um item que não pode deixar de ser citado devido à sua importância, mas nada mais que isto será feito porque a complexidade do protótipo é pequena o suficiente para permitir prescindir de UML.

Capítulo 4 – Análise e Projeto do Sistema Protótipo

4.1 Escolhas Feitas para os Recursos de Implementação

Foram feitas duas implementações. A interface gráfica de usuário é comum a ambas. A interface entre o cliente e o servidor é específica para cada uma. Somente a criação de novos elaces de snalização N7 é abordada.
Um dilema com que se deparou no desenvolvimento de cada um dos protótipos foi como compatibilizar as necessidades de sua baixa complexidade técnica e de seu pequeno grau de necessidade de gerenciamento, advindas das várias simplificações feitas para agilizar seu desenvolvimento, com as necessidades de um projeto real de alta complexidade técnica e de alta complexidade de gerenciamento. De fato, o grande problema do desenvolvimento do software é a complexidade com que se depara. Há que tomar cuidado para que uma complexidade maior não invalide a solução proposta. O uso de UML em uma plataforma do tipo da plataforma ROSE permite minimizar o problema da complexidade de um desenvolvimento real mas o protótipo a ser construído é muito simples. Além disto o maior objetivo deste protótipo é concatenar as idéias para mostrar, através de uma implementação simples, a importância de se padronizar interfaces para a interação com as BD das centrais telefônicas. Assim, optou-se por chamar a atenção para a importância do uso de UML numa implementação real mas optou-se por não utilizá-la na implementação do protótipo.

4.1.1 Linguagem de Programação Escolhida

A linguagem Java tem várias vantagens para a implementação do cliente e por isto foi escolhida para tal. Como o servidor é emulado quase toda a implementação está relacionada com o cliente. Por isto a linguagem Java foi escolhida para toda a implementação. Um dos motivos desta escolha é porque ela permite transparência de sistemas operacionais e de equipamentos. Isto é importante para atender a diversidade de equipamentos que é conveniente aproveitar dentre os existentes nas operadoras de telefonia.

Java permite usar uma solução específica (RMI) e outra padronizada (IDL). Isto facilita a implementação e a comparação.

O fato de RMI e IDL possibilitarem interações com clientes e servidores em sistemas distribuídos é fundamental para a natureza de problemas a atender, onde vários usuários precisam acessar várias bases de dados. Além disto, o uso de transparência de protocolos e de localização facilita o trabalho de programação.

Java também é interessante por permitir bons recursos de confecção de interfaces gráficas amigáveis – o que é fundamental não só para o usuário final como também para o programador responsável pela confecção da solução.

O servidor, por estar sendo emulado, também foi implementado em Java por ser muito simples. Numa implementação real, no entanto, deveria ser usada uma outra linguagem de melhor eficiência tal como C.

A possibilidade de usar IDL fornece um recurso interessante para prover compatibilidade entre CORBA, ASN.1 e GDMO. Já há propostas e estudos a respeito na comissão de telecomunicações da OMG. Ver Ref. [118] a [120], [123] e a página "web"da OMG.

Embora JNI permita mapear em Java muitas soluções já existentes nas operadoras isto não foi objeto de estudo por estar ligado à implementação do servidor.

Recursos de acesso a bases de dados via SQL também poderiam ser muito úteis nos servidores mas, conforme já dito, o servidor não é objeto de maiores estudos neste protótipo.

O fato de o protocolo IIOP funcionar sobre TCP/IP é interessante não só para permitir portabilidade futura como também para aproveitar eventual infra-estrutura atual de rede local (o que é muito comum nas operadoras). Pode também permitir também acesso discado para servidores remotos localizados junto às bases de dados das centrais telefônicas.

Sob o ponto de vista de eficiência do código gerado pelo compilador, há outras linguagens mais eficientes que Java. No entanto, o conjunto de recursos associados à linguagem Java foram suficientes para que ela fosse escolhida como a linguagem a ser usada para todo o desenvolvimento, considerando as simplificações feitas principalmente no servidor.

4.1.2 Sistema Operacional e Outras Ferramentas Software

Como pressuposto do trabalho foi totalmente descartada a opção de usar plataformas que permitissem soluções via ASN.1 e GDMO: dependem de plataformas caras, de acesso restrito; geram códigos que prescindem de uma pilha de camadas OSI – que além de cara não é de acesso fácil no contexto de sistema operacional e equipamento que se queria adotar para o desenvolvimento (conforme descrito mais abaixo).

Como a linguagem escolhida – Java – permite independência de plataforma, o desenvolvimento poderia ter sido feito, por exemplo, tanto em UNIX quanto em Windows. LINUX também é uma opção interessante. Optou-se pelo Windows por um motivo simples: já estava disponível facilmente e é o recurso mais comum nas operadoras brasileiras.

Sendo a complexidade do sistema como um todo muito simples, após algumas experiências foi descartado o uso de ferramentas de ajuda para geração de código em Java porque sobrecarregavam muito a máquina em que se pretendia testar se era possível viabilizar o desenvolvimento. Também por motivos de aprendizado se optou que todo código feito fosse editado com um editor de texto simples. Uma implementação profissional deve, no entanto, usar UML e uma plataforma do tipo da plataforma ROSE, conforme já citado anteriormente. Os recursos de software para prover a implementação em Java foram obtidos usando versões experimentais da página "web" da Sun, o que causou muita perda de tempo devido às constantes modificações entre versões sucessivas.

Algumas tentativas de usar uma outra implementação de ORB (ver Ref. [150]), foi descartada porque sobrecarregava muito a máquina em que se pretendia testar se era possível viabilizar o desenvolvimento (um Pentium 133 com apenas 16 Mb de RAM).

4.1.3 Contexto Mínimo para o Protótipo

A implementação escolhida só abrange alguns poucos aspectos da criação de um ESN7 conforme os padrões ITU-T.
Tratar um ESN7 pressupõe, pelo menos, criar um ESN7, remover um ESN7, modificar um ESN7, ver as características de um ESN7, criar RSN7, adicionar circuitos de voz à RSN7 criada, etc. Optou-se por tratar de uma forma simplificada, somente os principais parâmetros da criação do ESN7 (ITU-T). Nada é abordado a respeito de ESN7 ANSI.

Não é feita qualquer restrição na implementação do protótipo que invalide a possibilidade de se tratar quaisquer outros temas, mesmo que totalmente distintos (como por exemplo a coleta de dados de tarifação). Por isto, nas telas que foram criadas para o protótipo de ESN7 são deixados alguns exemplos de outros temas que poderiam ser acrescentados na IGU.

Principais Parâmetros para Criação de um ESN7

A partir de uma análise telefônica do assunto (conforme exposto no item anterior específico sobre os conceitos do ESN7) pelo menos os seguintes parâmetros devem ser usados:

1) - OPC,
2) - DPC,
3) - tipo de representação adotada para os códigos de ponto de sinalização (OPC e DPC),
4) - número do canal do ESN7 no feixe PCM,
5) - SLC,
6) - identificação do ESN7,
7) - identificação do conjunto de ESN7,
8) - identificação da rota de SN7,
9) - identificação do hardware do terminal de dados e
10) - identificação do terminal de ESN7.

Em grande parte dos casos práticos, os parâmetros de nível 2 do ESN7 podem assumir valores default previamente definidos. Se necessário mudá-los faz-se isto antes ou após ter sido criado o ESN7 em uma tela específica para este objetivo. Isto simplifica a atividade inicial de criação do ESN7, simplifica o protótipo e não sobrecarrega a janela de interações com o operador com muitas informações em uma mesma área. Além disto, é uma alternativa usada em produtos reais que operam nas redes das operadoras mundiais. Por isto os parâmetros de nível 2 do ESN7 não serão objeto de detalhamento no protótipo.

4.2 A Solicitação do Operador

A solicitação é composta das partes e das regras listadas a seguir. Parte destas sugestões serão implementadas no protótipo, conforme detalhado mais a frente.

4.2.1 Identificador das Versões Usadas

Deve ser composto das seguintes partes:

- versão do cliente,
- versão do servidor,
- versão da interface.

A negociação da versão a ser usada por um servidor quando receber uma solicitação de um cliente deve ser baseada nas seguintes premissas:

- sempre deverá haver compatibilidade entre uma versão mais nova e uma mais velha;
- o servidor deve responder com a versão mais nova que exista nele e que não seja superior à usada pelo cliente;
- todos clientes e servidores devem manter a implementação de uma versão comum a todos eles e a supressão desta versão só pode ser feita quando todos clientes e servidores evoluírem para uma versão mais nova (que necessariamente não precisa ser a última em todos clientes e servidores).

4.2.2 Identificador da Solicitação

É um identificador único da solicitação feita pelo cliente. Serve para diferenciar a solicitação de criação de um ESN7 da solicitação concorrente de criar outro ESN7. Deve ser usado um identificador composto de:

- IP do cliente que fez a solicitação,
- data, hora, minuto e segundo da solicitação
- e um número seqüencial de 16 bits usado para identificar a solicitação.

A codificação pode ser feita usando caracteres ASCII ou Unicode, devidamente indicados por um identificador de código usado.
A seguir pode vir uma lista de endereços IP de servidores aos quais a solicitação é dirigida. Esta lista deve ser precedida do número de endereços de servidores contidos na lista.
O código do padrão usado deve ser a 1ª informação enviada.

Detalhamento da Solicitação

Usado para identificar o que está sendo solicitado: o que é, qual cliente solicita, os detalhes da solicitação e a que servidor (ou a que servidores) ela vai dirigida. Um conjunto de números deve ser usado para detalhar a solicitação. Estes números devem estar associados a um texto que pode ser traduzido para diferentes idiomas. O texto em si, na medida do possível, não é enviado na solicitação mas só o número associado ao texto. Os parâmetros são dependentes da escolha específica feita pelo usuário da interface gráfica e de como foi feita esta entrada de dados e por isto terão que ser dependentes do idioma e poderão ser enviados para o servidor tal como selecionados pelo usuário da IGU. Deverão ser compostos de um "*string*" de caracteres ASCII ou Unicode.
A tabela abaixo sugere uma forma de fazer isto. Não pretende ser exaustiva. Apenas pretende cobrir o que é necessário para o protótipo.

Ações		Parâmetros	
No.	Texto	No.	Texto
1	Criar ESN7	1	Hardware do Terminal de Dados
1	Criar ESN7	2	Hardware do Terminal de ESN7
1	Criar ESN7	3	Representação do SPC
1	Criar ESN7	4	OPC
1	Criar ESN7	5	DPC
1	Criar ESN7	6	Número do canal do ESN7
1	Criar ESN7	7	SLC
1	Criar ESN7	8	Identificação do ESN7
1	Criar ESN7	9	Identificação do Conjunto de ESN7
1	Criar ESN7	10	Identificação da Rota de SN7
1	Criar ESN7	11	Endereço IP
2	Remover ESN7	...	...
3	Modificar ESN7	...	...
4	Ver ESN7	...	...
...	...	...	...

Tabela # 4.1 – Solicitação do Usuário

4.3 A Resposta

A resposta é composta de partes semelhantes às da solicitação e, adicionalmente, do identificador único da resposta, ou seja:

1) Um identificador das **versões dos softwares** usados.
2) Um identificador único da **solicitação**.
3) **Detalhamento da solicitação**.
4) Um identificador único da **resposta**.
   Mesmas premissas do identificador único da solicitação: tipo de codificação usada, IP do servidor, data, hora, minuto, segundo e número seqüencial de 16 bits. É necessário para eventualmente permitir o uso de mais de uma resposta associada a uma mesma solicitação.
5) **Detalhamento da resposta**.
   As respostas podem ou não vir acompanhadas de detalhes. Os detalhes, quando usados, devem vir em ASCII ou Unicode, conforme as necessidades de padronização para diferentes idiomas, endereço IP do servidor que a envia, etc. Não são objeto de detalhamento neste trabalho.
6) **Resultado** básico da operação
   Consta de um byte com um número que está associado a um significado do resultado básico da operação. Deve vir antes do campo de detalhamento da resposta. Pode, opcionalmente, associar à resposta básica outras informações complementares, conforme aplicável. Quando houver necessidade de uso deste detalhamento, será especificado conforme mostrado na tabela abaixo. As opções de respostas são igualmente identificadas por números (que podem estar associadas a textos independentes do idioma) e a uma seqüência opcional de informações de detalhamento da resposta. A tabela abaixo apresenta uma 1ª. idéia de como padronizar isto.

Resultado Básico da Operação	
No.	**Texto**
1	Resultado sem detalhes, bem sucedido, execução integral e correta.
2	Resultado sem detalhes, não executado.
3	Resultado sem detalhes, execução parcial.
4	Resultado com detalhes, bem sucedido, execução integral e correta
5	Resultado com detalhes, não executado.
6	Resultado com detalhes, execução parcial.
...	...
...	...

Tabela # 4.2 – Resultado da Operação Solicitada pelo Usuário

Detalhamento da Resposta	
Uso	**Tamanho em caracteres**
Tipo de caracteres usados	1 (*)
Detalhamento da resposta	até 2^{24} caracteres ASCII ou Unicode

Tabela # 4.3 - Detalhamento da Resposta

Nota:

(*) Identificação do padrão usado: 1 = Unicode; 2= ASCII.

4.4 Estrutura da Base de Dados

A partir das tabelas anteriores, sobretudo a partir daquela em que foi mostrado o detalhamento da solicitação é possível obter uma referência para uma padronização da base de dados necessária. A lista dos parâmetros e das ações necessárias para criar um ESN7 mostra quais são os objetos para a criação de um ESN7 que precisam ser padronizados na Base de Dados da central e quais as ações relevantes que devem ser consideradas. Para outras aplicações, conforme dito anteriormente, o trabalho feito na definição da MIB da TMN do ITU-T pode ser usado como referência para levantar os objetos necessários para as diferentes necessidades de interações com as bases de dados das centrais telefônicas.

Para atender às necessidades de criação de um ESN7 tal como apresentado anteriormente podem ser definidas interfaces que incluam como métodos, as ações listadas nas tabelas anteriores e cujos parâmetros de entrada e de saída também já estão listados nestas mesmas tabelas.

É necessário fazer uma representação mais completa e exaustiva não só destes objetos desta base de dados como também da requisição e da resposta que foram rapidamente apresentadas anteriormente. Devem ser usadas interfaces e classes para representar as requisições e as respostas contidos nos textos e tabelas anteriores.

Nos itens que se seguem uma pequena parte destas idéias – que foi implementada – é detalhada. Isto foi feito parcialmente para simplificar a implementação e testar, em protótipo, tão-somente o cerne da questão.

4.5 Criação do ESN7 no Lado do Usuário

Sob um ponto de vista operacional, estão listados abaixo, os principais pontos que são necessários, no lado do usuário, para criar um ESN7.

- Ter um software que possa ser executado em diferentes sistemas operacionais (UNIX, Windows32, etc.).
- Selecionar, a partir de janelas gráficas:
  - o tipo de representação para os SPC (OPC/DPC);
  - OPC e DPC em que o ESN7 deve ser criado;
  - o número do canal do ESN7;
  - o código do ESN7 (SLC);
  - a identificação do ESN7;
  - a identificação da rota de sinalização associada ao ESN7;
  - a identificação do conjunto de sinalização do ESN7;
  - hardware do feixe PCM de 2Mbps ("**terminal de dados**");
  - hardware do canal de 64 Kbps do ESN7 ("**terminal de sinalização**").
- Usar vários recursos gráficos no terminal de vídeo para selecionar a operação específica (criar um ESN7) entre outras existentes (visualizar, remover, modificar) e controlar a execução da criação do ESN7.
- Usar, para as seleções, recursos gráficos de seleção que já mostrem (através de uma lista) as possibilidades existentes para cada parâmetro, de forma a tornar o sistema amigável, robusto, menos susceptível a erros.
- Receber e tratar os informes de respostas e os informes espontâneos de erros.

4.6 Criação do ESN7 no Lado da Central Telefônica

Dentre outros pontos está listado abaixo o que é necessário prover no lado da central telefônica para criar um ESN7.

- Interpretar a ação solicitada pelo usuário e mapeá-la diretamente nas funções da central telefônica específica ou em comandos CHM desta central ou em qualquer sistema herdado que interaja com a BD da central.
- Receber o resultado da ação e enviá-lo para o usuário.
- Notificar qualquer irregularidade observada.

4.7 Interface Gráfica do Usuário (IGU ou "GUI")

É também chamada de Interface para Comunicação Homem – Máquina (ICHM).

Numa primeira janela foi colocado um menu com uma barra com algumas opções básicas possíveis. A intenção não é detalhar todas as opções apresentadas mas somente sugerir o que poderia ser feito.

A área de texto em branco na **Figura # 4.3** é uma área que foi prevista para ser utilizada pelo cliente para realimentar o operador sobre o andamento de suas solicitações no servidor. Uma barra de rolagem permitiria ao usuário verificar o andamento de suas últimas solicitações e os resultados obtidos. No entanto, para simplificação da implementação, foi deixado que a realimentação fosse dada pela própria janela do sistema operacional.

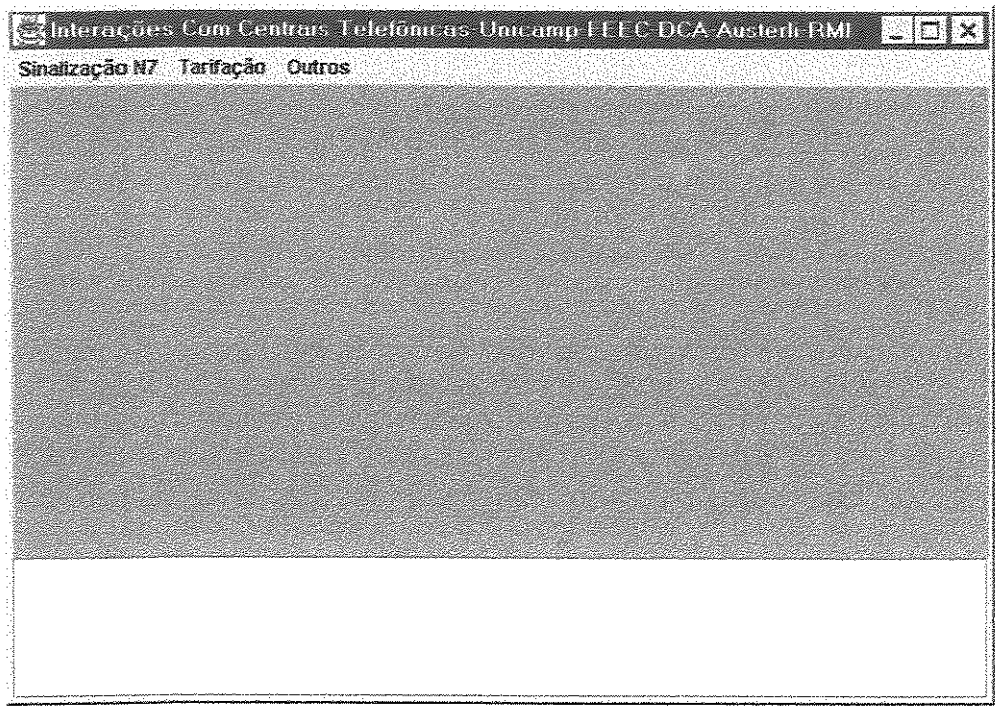


Figura # 4.3 – 1a. Janela da IGU

Uma 2ª janela apresenta detalhes do tratamento possível para SN7. Por simplificação não foram incluídas opções de "adicionar circuitos de voz", modificar ESN7, etc.

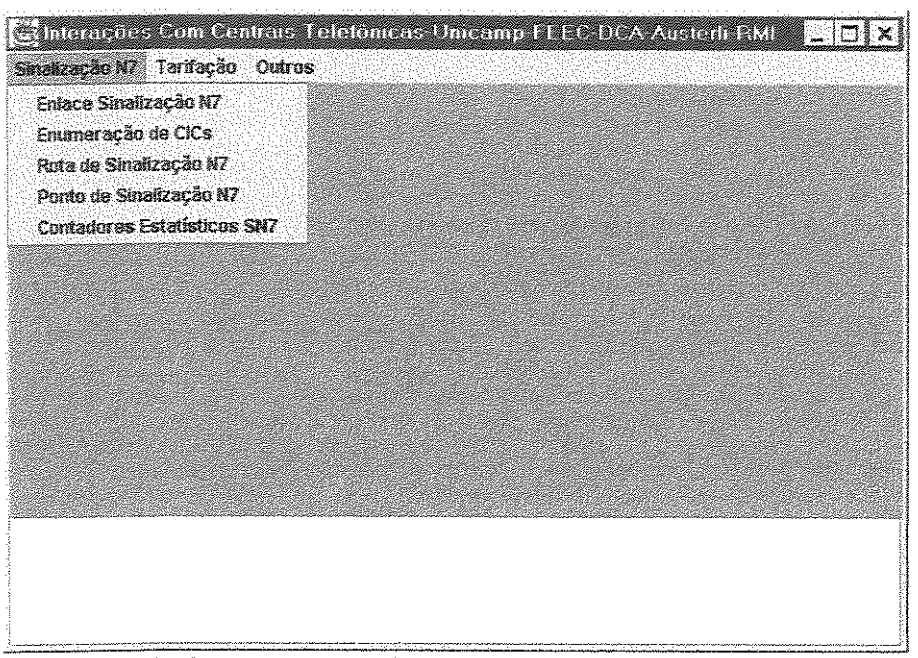


Figura # 4.4 - 2a. Janela da IGU

A 3ª. janela apresenta detalhes específicos da criação do ESN7. É a única opção que foi mais detalhada. Podem ser criadas várias janelas destas, cada uma para criar um ESN7. Isto também poderia se aplicar às demais opções da 2ª janela mas somente a opção de criação do ESN7 foi detalhada.

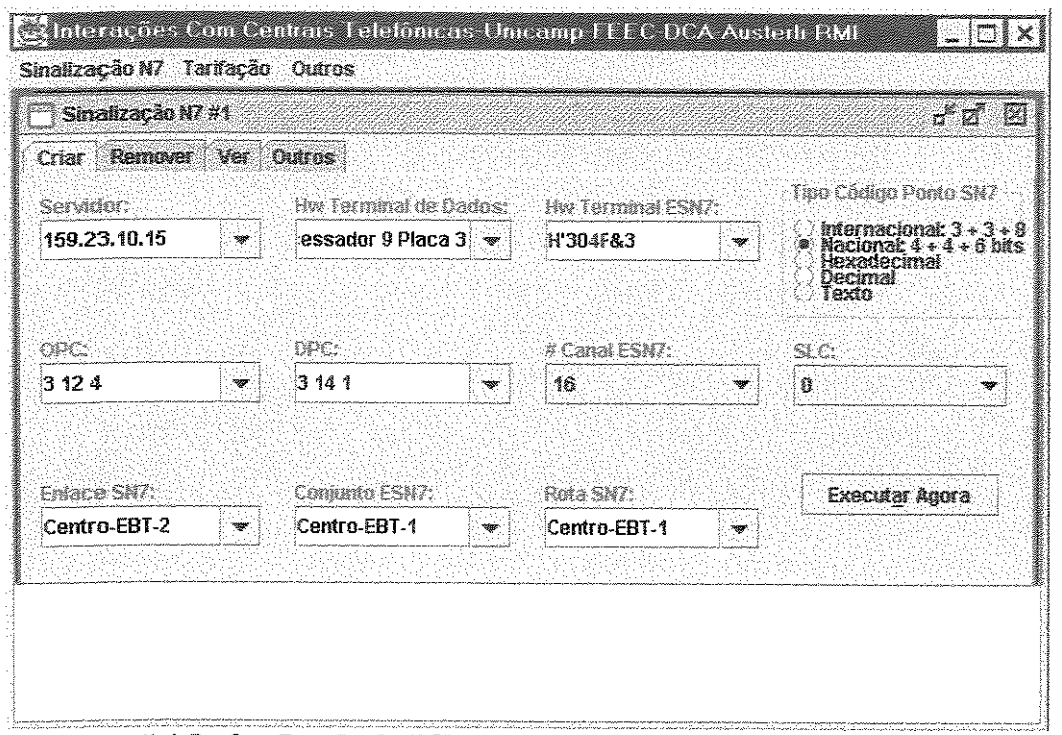


Figura # 4.5 - 3a. Janela da IGU

Outras telas, como a abaixo, não foram detalhadas.

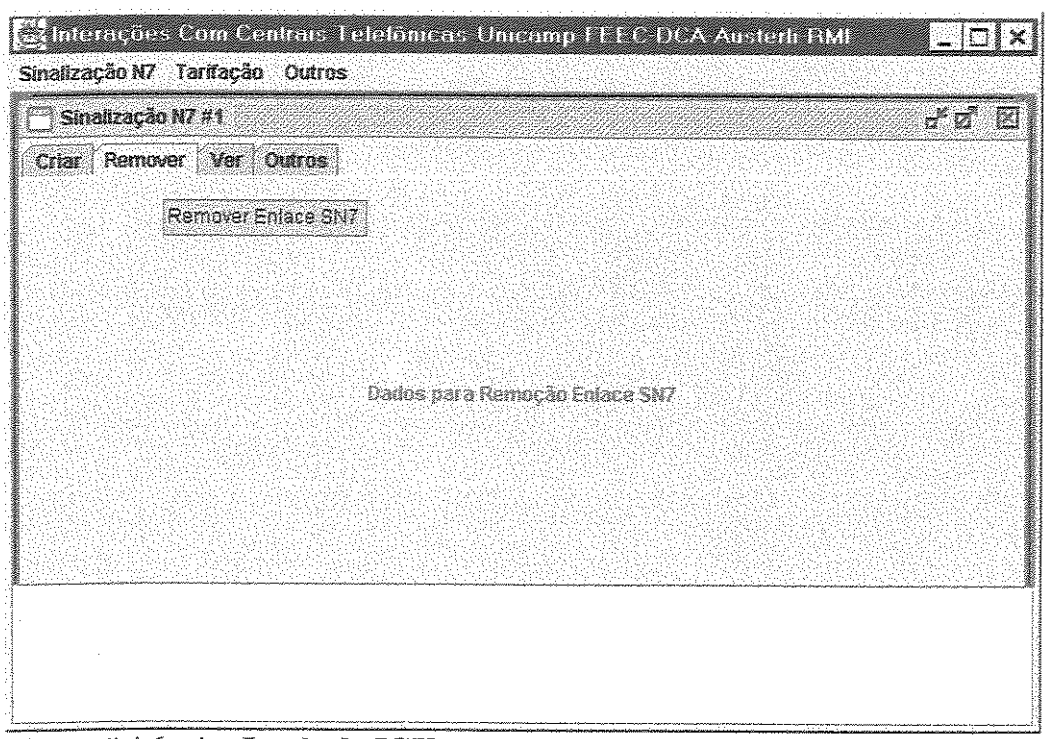


Figura # 4.6 - 4a. Janela da IGU

4.7.1 Estrutura de Classes Usada na Implementação da IGU

Em todo o protótipo, em geral, em muitas classes usadas há métodos com o mesmo nome da classe. Devido ao fato de o próprio contexto das frases poder ser suficiente para diferenciar uma referência da outra e para não tornar o texto ainda mais carregado de detalhes, nem sempre está explicitado se a referência feita no texto se refere à classe ou ao método. O objetivo de apresentar isto aqui é mostrar um exemplo de como as classes foram agrupadas para montar uma IGU de aplicação concreta e também mostrar como foi feito o mapeamento dos dados fornecidos na IGU com os parâmetros da interface usada entre o cliente e o servidor.

4.7.2 A Construção da 1ª e 2ª Janelas da IGU

Java incorpora em suas primitivas muitos recursos que facilitam a construção da IGU. Neste protótipo foram usados vários componentes Swing da linguagem Java. Os programas feitos foram adaptados a partir de vários exemplos contidos nas Ref. [6].

Para receber os parâmetros de criação do ESN7 é necessário ter uma janela que possa ser iconizada, ampliada, reduzida, etc. e que possa conter os vários componentes que permitam a seleção das opções que o usuário irá usar.

A primeira janela foi feita usando uma instância da classe que foi denominada ***InternalFrameTst***.

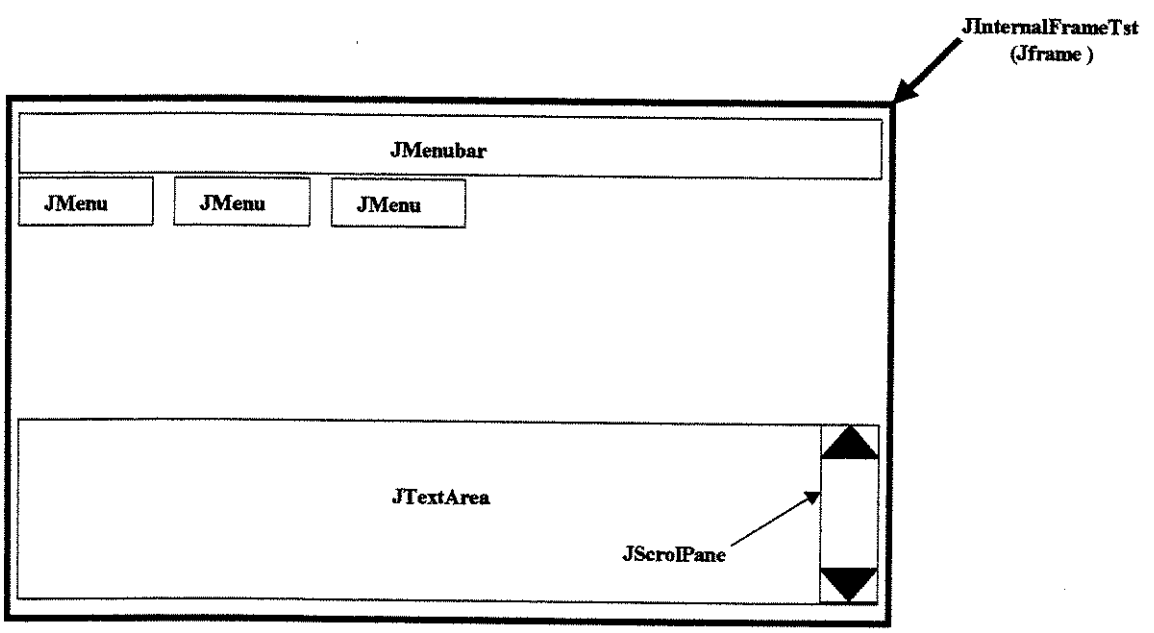


Figura # 4.7 - Estrutura de Classes Usadas na Implementação da 1a. Tela da IGU

A classe ***InternalFrameTst*** estende **Jframe** – um container de alto nível que permite adicionar ao seu conteúdo um novo objeto e gerenciar a sobreposição dos objetos. Note que esta janela possui uma borda, uma barra de escolha de opções ("***menubar***"), um título e botões para fechar e minimizar que garantem que se esta janela é fechada a aplicação é

terminada. Esta classe coloca o título na 1ª. janela, dimensiona o tamanho inicial da janela, cria uma barra de menus no topo da janela e uma área de texto na parte inferior da 1ª. janela. A área de texto usa as classes **JtextArea** e está reservada para receber o texto de resposta que venha do servidor. Para rolar o texto na área de texto é usada a classe **JscrollPane.**

As opções básicas que aparecem no menu da 1ª. tela são configuradas a partir de instâncias da classe **JMenu**, sendo elas:

- "Sinalização N7"
- "Tarifação
- "Outros"

Estas opções de menu são adicionadas à barra de menu **JMenuBar** através de primitivas do tipo "*add*". Somente a instância do **JMenu** "Sinalização N7" é detalhada em uma 2ª. tela. Para este "detalhamento", um método utiliza várias instâncias da classe **JMenuItem** para incluir os seguintes itens como opções de escolhas válidas para a instância do **JMenu** "Sinalização N7" do menu da 1ª janela:

- "Enlace Sinalização N7"
- "Enumeração de CICs"
- "Rota de Sinalização N7"
- "Ponto de Sinalização N7"
- "Contadores Estatísticos SN7"

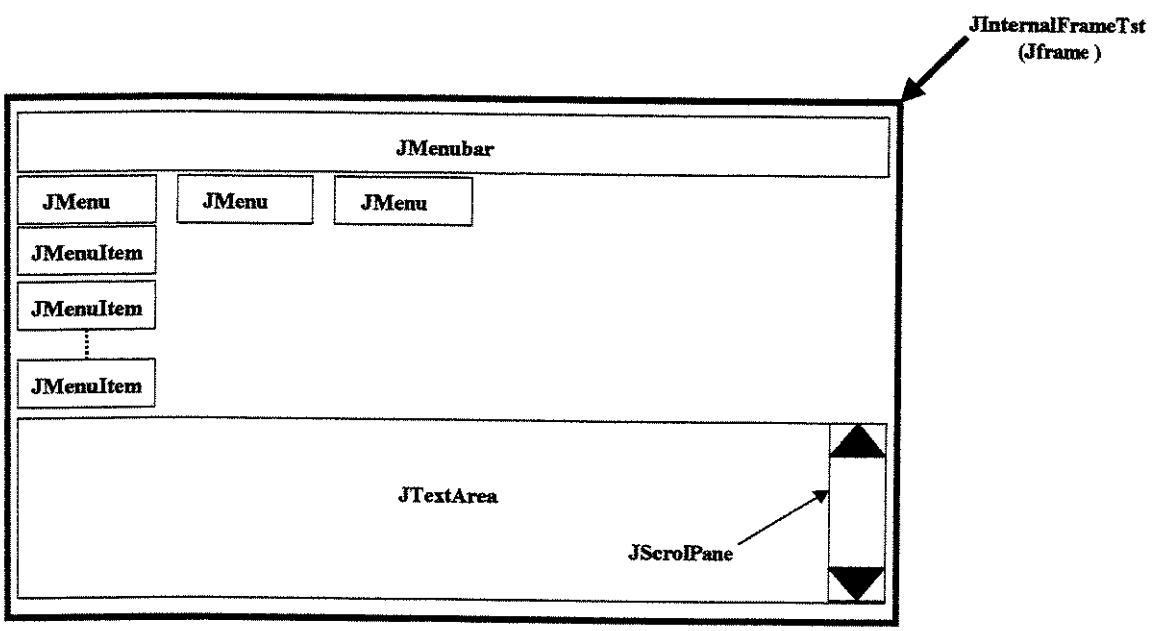


Figura # 4.8 - Estrutura de Classes Usadas na Implementação da 2a. Tela da IGU

Para cada uma destas instâncias da classe **JmenuItem** é associado um tratador de eventos (instâncias de **ActionListener**) que executa um método para criar um "frame" específico que é função da escolha feita.

Escolha feita no Menu (JmenuItem)	Método associado à Escolha ("actionPerformed")
"Enlace Sinalização N7"	createFrameESN7
"Enumeração de CICs"	createFrameCIC
"Rota de Sinalização N7"	createFrameRSN7
"Ponto de Sinalização N7"	createFramePSN7
"Contadores Estatísticos SN7"	createFrameCESN7

Tabela # 4.4 – Tratadores de Eventos

Somente o método de criação do ESN7 foi detalhado. Ele usa uma instância de **InternalFrameESN7** para montar a janela que tratará o ESN7. A classe **InternalFrameESN7** estende a classe **JinternalFrame** de forma a permitir que outras "janelas" sejam criadas dentro da "janela" para criar um ou mais ESN7. Note-se que foi usado o termo "janela" somente para simplificar a compreensão. Na verdade, a que foi chamada de "janela" principal, conforme dito antes, é uma extensão de **Jframe** e, igualmente, a "janela" interna é uma extensão da classe **JinternalFrame**.

4.7.3 A Construção da 3ª Janela da IGU

Para adicionar os vários objetos que permitem ao operador fazer sua escolha na hora da criação do ESN7 foi definida a classe **InternalFrameESN7** como uma extensão da classe **JinternalFrame.** Ela define as dimensões default que a nova janela pode ter, indica onde ela deve aparecer quando for criada, preestabelece que seu tamanho pode ser mudado e também que esta janela pode ser fechada, maximizada e minimizada. Adiciona a janela ao conteúdo do container e cria uma instância da classe **TabbedESN7** para tratar do conteúdo desta nova janela (a 3ª).

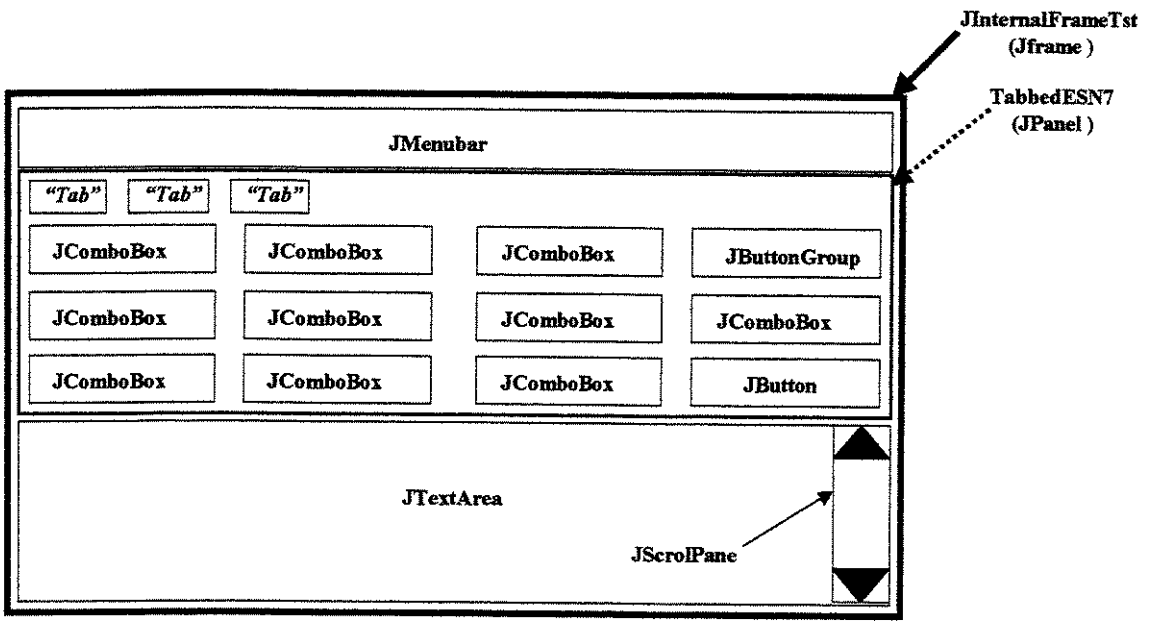


Figura # 4.9 - Estrutura de Classes Usadas na Implementação da 3a. Tela da IGU

A classe **TabbedESN7** estende a classe **JPanel** e contém um método que usa internamente a classe **JTabbedPane** para permitir criar 4 etiquetas ("*tabs*"), identificadas com os nomes das opções de escolha previstas para o ESN7, conforme abaixo:

- "Criar"
- "Remover"
- "Ver"
- "Outros"

A escolha inicial quando se entra na tela foi preestabelecida para ser "**Criar**".

Cada uma destas etiquetas dá acesso a uma tela específica, que deve estar associada a um método para detalhar o tratamento possível para cada um dos tipos de ações previstas de serem feitas em ESN7.
Quando se para sobre cada uma destas etiquetas, é dada uma mensagem com a função ("dica" ou em inglês, "*tip*") do que está contido em cada etiqueta.

Ação (Etiqueta)	Texto Associado à Etiqueta ("dica" ou "tip")
"Criar"	**"Criar Enlace SN7"**
"Remover"	**"Remover Enlace SN7"**
"Ver"	**"Visualizar Dados de Enlace SN7"**
"Outros"	**"Reserva para Uso Futuro"**

Tabela # 4.5 – Textos Associados às Etiquetas da 3a Janela da IGU

A etiqueta associada à opção "Criar" dá acesso a uma janela que detalha os parâmetros necessários para a criação de um ESN7. Na verdade, somente a opção "**Criar**" foi detalhada (através do método **PainelCriarESN7**). As demais opções não estão detalhadas e somente estão dando acesso a janelas em que somente está escrito um texto orientativo específico, conforme tabela abaixo.

Ação (Etiqueta)	Método	Texto dentro da Janela Específica
"Criar"	**PainelCriarESN7**	
"Remover"	**makeTextPanel**	"Dados para Remoção Enlace SN7"
"Ver"	**makeTextPanel**	"Dados para Visualização Enlace SN7"
"Outros"	**makeTextPanel**	"Outros Dados de Enlace SN7"

Tabela # 4.6 - Texto Dentro da Janela da IGU da 3a. Janela da IGU

Embora exista previsão de métodos para atender cada escolha feita pelo usuário, somente uma delas – a criação do ESN7 – é detalhada através do método **PainelCriarESN7**. Este método cria a interface gráfica para a entrada de dados de criação de um ESN7. Usa uma instância da classe **JPanel** para montar o painel a ser usado para receber os componentes necessários na opção "Criar" ESN7 da 3ª janela. Para simplificar a implementação, para quase todos parâmetros definidos como necessários para a criação de um ESN7 é criada uma classe específica estendida a partir da classe JcomboBox. Estas classes específicas estão listadas na tabela abaixo. Os diversos componentes são adicionados ao objeto Jpanel através de primitivas do tipo "add".

Para prever a interação com diferentes servidores remotos foi incluída uma caixa de combinações ("ComboBox") especial para dar o endereço IP do servidor (central telefônica ou dispositivo intermediário) associado à solicitação do cliente (IGU).
Para selecionar o tipo de representação usada para o SPC (OPC/DPC) é usado um conjunto de "botões de rádio" ("radio buttons") criados a partir de uma classe definida especialmente para isto (RadioButtonCPS). Ela é uma extensão da classe JPanel – que por sua vez, como "containner" intermediário, agrupa em um mesmo grupo de botões (classe ButtonGroup) vários "botões de rádio" (instâncias da classe JRadioButton). Estes "radio buttons" são agrupados em um único grupo coeso que forma um painel de uma única coluna dentro da 3a janela. Para isto é usada uma instância da classe JPanel e um gerenciador de layout do tipo GridLayout.

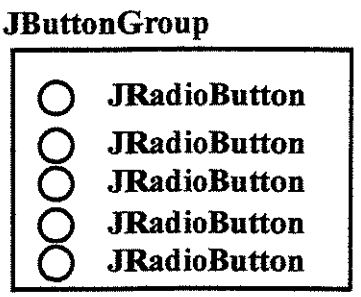


Figura # 4.10 - Composição da Classe JButtonGroup na 3a. Tela da IGU

Para disparar a criação do ESN7 é usado um botão para Executar a escolha feita, criado a partir de uma instância da classe ButtonExecutar – que é uma extensão da classe Jbutton. Quando este botão é acionado a interação do cliente com o servidor remoto é acionada. Esta interação é detalhada mais a frente no item específico do cliente (TrataIGU).

As classes específicas que constam no método PainelCriarESN7 da classe TabbedESN7 estão listadas abaixo.

Parâmetro	Classe
Hardware do Terminal de Dados	**ComboBoxDTM**
Hardware do Terminal de ESN7	**ComboBoxSLT**
Representação do SPC	**RadioButtonCPS**
OPC	**ComboBoxOPC**
DPC	**ComboBoxDPC**
Número do canal do ESN7	**ComboBoxCanalESN7**
SLC	**ComboBoxSLC**
Identificação do ESN7	**ComboBoxIdentidadeESN7**
Identificação do Conjunto de ESN7	**ComboBoxIdentidadeCESN7**
Identificação da Rota de SN7	**ComboBoxIdentidadeRSN7**
Botão para Executar a Escolha Feita	**ButtonExecutar**
Endereço IP	**ComboBoxIP**

Tabela # 4.7 - Classes do método PainelCriarESN7 da classe TabbedESN7

Como a janela que trata a criação do ESN7 está composta de várias instâncias de classes estendidas a partir da classe JcomboBox, é usado um gerente de layout da classe

GridLayout, de forma a adicionar uniformemente em um mesmo Jpanel, em 3 linhas e 4 colunas, as instâncias das diversas classes necessárias para compor a 3ª janela.

A caixa de combinações (Combo Box) para escolha do endereço IP do servidor que aparece na janela de criação do ESN7 é construído a partir de uma instância da classe **ComboBoxIP** e ocupa a 1ª posição (linha 1, coluna 1) da grade 3X4 do gerente de layout da classe **GridLayout.**

As demais caixas de combinações, "botões de rádio" e o botão de execução ocupam as posições seguintes. Veja tabela acima. Todos são adicionadas ao painel Jpanel da 3ª janela através de primitivas "*add*".

As opções exibidas na caixa de combinações de escolha do SLC são previsíveis (faixa de 0 a 15). Por isto – e também para evitar erros na entrada dos dados – o método **ComboBoxSLC** existente dentro da classe **ComboBoxSLC** não permite que os valores dentro desta caixa de combinações seja editável pelo usuário. Isto é, os valores podem ser escolhidos mas não podem ser outros senão os da lista apresentada.
O mesmo ocorre para a caixa de combinações de escolha do número do canal de SN7. A única diferença aqui é que a faixa permitida de intervalos de tempos de canais do feixe PCM é de 1 a 31.
Para as demais caixas de combinações os parâmetros devem ser escolhidos pelo usuário livremente e por isto foram configurados para serem editáveis. A idéia é apresentar os valores que efetivamente existam em uma aplicação real e que seriam obtidos a partir de iniciação dos valores das caixas de combinações a partir dos valores reais existentes. No entanto, para fins deste protótipo, foram usados valores hipotéticos quaisquer. Numa aplicação real estes dados deveriam ser iniciados a partir de condições reais.
Outra simplificação feita diz respeito a verificar a consistência entre os valores escolhidos para OPC e para DPC e o tipo de representação escolhida para eles. Embora numa aplicação real isto seja importante (ou mesmo imprescindível, de forma a evitar que tal crítica seja feita pelo servidor) para os objetivos deste protótipo isto não foi considerado.

4.8 1ª Implementação da Interação Cliente / Servidor: Uso de RMI

4.8.1 Objeto Enviado pela ICHM à Central Telefônica

A idéia básica é que um usuário que queira resolver um problema específico seja capaz de desenvolver um objeto que possa de ser enviado por um equipamento que funcione como cliente para ser executado em um outro equipamento que funcione como servidor remoto e que, adicionalmente, após executar as ações necessárias no servidor, envie as respostas de volta ao cliente.
À medida em que o operador atualiza os dados via IGU, estes dados são atribuídos aos objetos específicos que lhes são associados para serem enviados ao servidor. Para isto, em cada classe que trata cada parâmetro ao qual o usuário da IGU tem acesso, quando a escolha é feita é acionado um evento que, através de um método **(actionPerformed)** atribui a escolha que o usuário fez na caixa de combinação (na maioria dos casos) ao objeto a ser enviado ao servidor.
Para o caso mais completo de a IGU permitir mais de um tipo de ação no servidor (por exemplo, criar, modificar ou ver um ESN7), antes de ser ativada a ação no servidor, a natureza da a ação a ser executada deve ser escrita no objeto (ESN7) que será enviado ao

servidor remoto. No caso do protótipo isto ficou restrito a criar ESN7 (que por isto não é discriminada).
Para facilitar a integração com outras aplicações este objeto deve ser composto de um conjunto de "*strings*". Isto permite, dentre outras coisas, por exemplo, construir ferramentas para editar e enviar os comandos para o servidor usando um arquivo simples de "***batch***" conforme comentado no item 4.10.1.

4.8.2 O Servidor Remoto

Está baseado numa proposta apresentada na Ref. [6]. Nela um cliente solicita o cálculo do valor de "Pi" (¶) em um servidor remoto com maior capacidade de processamento que a do cliente.
O princípio do mecanismo do servidor é o mesmo, tanto para o cálculo de "Pi" (¶) quanto para o caso descrito neste protótipo. Para o caso deste protótipo, mudam os clientes e as tarefas associadas a eles. Para o protótipo foi também adicionada a interface gráfica do usuário que não existe no cálculo de "Pi" (¶).
O servidor desenvolvido na Ref. [6] é um programa simples que executa a tarefa que lhe for entregue, independentemente de qual seja esta tarefa. O que o protótipo faz é adaptar tal mecanismo para o caso específico dos acessos às bases de dados das centrais telefônicas.
Trata-se de um mecanismo genérico para execução de tarefas remotas, inicialmente desenvolvido por uma pessoa e deixado disponível para uso de qualquer outra em um certo diretório preestabelecido de um equipamento de uma rede. Permite que uma tarefa genérica desenvolvida por um cliente, seja "***serializada***", enviada para um servidor remoto, executada no servidor remoto e que a resposta seja enviada de volta ao cliente. Usa Java RMI para este propósito. As tarefas são executadas no equipamento em que o servidor estiver rodando. Isto exige, portanto, que o objeto remoto do servidor seja previamente carregado no equipamento onde deve ser executado. O servidor, no entanto, não entra no mérito da tarefa que o cliente deseja que seja executada no servidor. Cabe ao cliente detalhar toda a tarefa que será executada no servidor.

A idéia principal é que um objeto remoto no servidor ("**ComputeEngine**") receba uma tarefa de um cliente remoto, execute-a e retorne o resultado ao cliente de forma que quaisquer tipos de tarefas possam ser definidas a qualquer instante após a definição do objeto do servidor remoto e serem entregues a ele para serem tratadas sem depender de modificação na definição do objeto remoto do servidor. Tudo que se exige da tarefa a ser executada é que implemente a interface previamente definida no objeto do servidor remoto. Para isto o servidor é constituído de duas interfaces e de uma classe. As interfaces definem os métodos que podem ser invocados pelo cliente. A classe provê a implementação.

4.8.3 As interfaces Remotas contidas no "Package" compute

Funcionam como um mecanismo genérico para execução de tarefas remotas.
Residem em um diretório qualquer da rede, aqui denominado de "**d1**" (diretório de definição das interfaces) em um "*package*" denominado "**engine**". Seus códigos fonte podem ser encontrados no "tutorial" de RMI da Ref. [6]. Definem o protocolo que permite que as tarefas sejam submetidas ao objeto no servidor remoto e que, após serem executadas, que os resultados sejam enviados de volta ao cliente. Incluem duas interfaces que estão definidas em dois arquivos (**Task.java** e **Compute.java**). Está claro que a omissão de algumas partes do código compromete o entendimento do assunto. Para evitar dúvidas na compreensão do

tema e considerando a simplicidade e a eficiência dos códigos usados na definição destas duas interfaces e desta única classe, eles são transcritos e explicados nos itens que se seguem, tal qual usado no protótipo. Alguma eventual adaptação foi feita em relação ao código encontrado no "tutorial" de RMI da Ref. [6].

A Interface Compute

A interface "Compute" está contida num só arquivo (**Compute.java**). Pode ser vista abaixo.

```
package compute;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Compute extends Remote {
   Object executeTask(Task t) throws RemoteException;
}
```

OBS.: extraído do "tutorial" de RMI da Ref. [6].

A interface "Compute" é usada para permitir que qualquer tarefa seja submetida a um servidor remoto, sem entrar no mérito da tarefa em si. Possui somente um método: **executeTask**. Este método permite a passagem de um parâmetro de classe "Task" (ver item específico, abaixo) e retorna "Object". Pode ser invocado de forma transparente por um cliente remoto porque a interface **java.rmi.Remote** é estendida pela interface "Compute". A interface **Remote** é definida no "package" **java.rmi** padrão da Java e é usada para identificar interfaces cujos métodos podem ser invocados desde uma máquina virtual que não seja local.
A classe java.rmi.**RemoteException** é usada para prever o tratamento das exceções relacionadas com a comunicação que possam ocorrer durante a execução de uma chamada remota de método.

A interface Task

A interface "Task" está contida no arquivo "**Task.java**". Define como uma tarefa submetida pelo cliente é executada pelo mecanismo definido no objeto remoto do servidor ("Compute Engine").

```
package compute;
import java.io.Serializable;
public interface Task extends Serializable {
   Object execute();
}
```

OBS.: extraído do "tutorial" de RMI da Ref. [6].

A interface "Task" estende **java.io.Serializable** para permitir que os objetos sejam empacotados, desempacotados e transportados **por valor** entre as máquinas virtuais Java. Note-se que as implementações atuais de CORBA ainda não permitem isto.
Contém somente um método: **execute()**. Este método não tem parâmetros de entrada e retorna "Object".

Diferentes tipos de tarefas que implementem a interface "**Task**" podem ser executadas por um objeto "**Compute**". Como o objeto "**Task**" que se deseja executar remotamente estará escrito em Java, RMI carrega-o de forma transparente no servidor remoto, usando a interface "**Compute**", sem que o código do objeto "**Task**" tenha que ser explicitamente instalado no servidor.

Há a possibilidade de se usar RMI e JNI para interagir com programas escritos em outras linguagens utilizando "*wrappers*". Embora isto seja muito útil para mapear o servidor em aplicações herdadas já existentes nas centrais telefônicas, isto não é objeto do presente trabalho. Além disto, há outras formas e linguagens mais eficientes que Java para se implementar o servidor.

4.8.4 A implementação da Interface no Servidor Remoto (Classe *ComputeEngine*)

A implementação da interface remota é feita pela classe **ComputeEngine**, que reside num "package" "**engine**" em um diretório qualquer da rede, aqui denominado de "**d2**" (diretório do servidor).

```
package engine;
import java.rmi.*;
import java.rmi.server.*;
import compute.*;
public class ComputeEngine extends UnicastRemoteObject
                implements Compute
{
  public ComputeEngine() throws RemoteException {
    super();
  }
  public Object executeTask(Task t) {
    return t.execute();
  }
  public static void main(String[] args) {
    if (System.getSecurityManager() == null) {
      System.setSecurityManager(new RMISecurityManager());
    }
    String name = "//localhost/Compute";
    try {
      Compute engine = new ComputeEngine();
      Naming.rebind(name, engine);
      System.out.println("ComputeEngine bound");
    } catch (Exception e) {
      System.err.println("ComputeEngine exception: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

OBS.: extraído do "tutorial" de RMI da Ref. [6] e a seguir ligeiramente modificado.

A classe **ComputeEngine** estende "**UnicastRemoteObject**" para permitir a criação de um objeto remoto que suporte comunicação ponto a ponto e que permita aceitar as solicitações de chamadas dos clientes em uma porta qualquer.
A classe "ComputeEngine" somente implementa a interface "**Compute**".
Há outros métodos que somente podem ser acessados localmente: um "construtor" para os objetos "ComputeEngine" e o método "**main**" usado para criar o objeto "ComputeEngine" e deixá-lo disponível para os clientes. Note-se que este método "main" não varia para qualquer que seja a "task" recebida do cliente.
Para atender os casos em que ocorra uma exceção ao se tentar exportar um objeto é usada a exceção "**RemoteException**". Ela atua, por exemplo, quando não hajam recursos de comunicação disponíveis ou a classe "*stub*" adequada não for encontrada.
É implementado o método remoto especificado pela interface "Compute". Este método é o responsável pela execução da tarefa remota "**Task**". É este o método que implementa o protocolo entre "ComputeEngine" e seus clientes. Este método recebe uma tarefa da classe "**Task**" como parâmetro de entrada e retorna um objeto da classe "**Object**". Cabe ao cliente ter implementado "**Task**" previamente. Para isto o Objeto "**Task**" do cliente deve ter previamente implementado o método **execute** definido pela interface "**Task**". Como será mostrado na descrição do cliente, a classe **TrataESN7** foi definida no cliente para implementar a tarefa da classe "**Task**" a ser executada no servidor.
Um método "**main**" é usado para iniciar o servidor remoto. Ele não é um método remoto. Não pode ser chamado por nenhuma outra máquina virtual Java. Há necessidade de ser ativado por um operador que acesse a máquina servidora diretamente.
Um "security manager" provê a proteção contra os acessos indevidos aos recursos do sistema.
Para que o cliente possa identificar o objeto remoto que é capaz de executar a tarefa no servidor é dado um nome para o objeto remoto levando em conta o endereço IP do servidor e o nome da interface a ser usada ("**Compute**"). A seguir é criada uma instância de "ComputeEngine" e é feito o registro do nome ("**rebind**") usando o nome do objeto remoto previamente definido e a instância criada de "**ComputeEngine**". É transparente se o servidor e o registro de nomes estão localizados no mesmo equipamento ou não.
Através do "construtor" da superclasse "UnicastRemoteObject " o novo objeto criado é exportado e o objeto remoto "ComputeEngine" fica disponível para receber chamadas de entrada dos clientes através de uma porta anônima (escolhida pelo RMI ou pelo sistema operacional).
Uma mensagem é impressa na tela do servidor para indicar que o servidor já está pronto para receber invocações remotas dos clientes.
Por simplificação, as exceções que possam ocorrer são tratadas de forma genérica e simplificada: exibição na tela da mensagem relativa à exceção e conteúdo da pilha *("stack")*.

4.8.5 O Cliente

O cliente é ativado a partir do momento em que o botão de executar é "pressionado" na interface gráfica (classe **ButtonExecutar** na interface gráfica do usuário).
TrataIGU e **TrataESN7**, em conjunto com as classes que compõem a interface gráfica do usuário (**IGU**), residem no diretório "**d3**" e compõem as classes que definem o cliente.
Para a compreensão do que se segue é importante observar como foi definida, no cliente, de forma simplificada, a classe ESN7.

```
package client;
import java.io.Serializable;
public class Esn7 implements Serializable
{
   String slc = null;
   String canal_esn7 = null;
   String hw_terminal_dados = null;
   String hw_terminal_sn7 = null;
   String tipo_cps = null;
   String opc = null;
   String dpc = null;
   String id_esn7 = null;
   String id_cesn7 = null;
   String id_rsn7 = null;
   String tipo_ação = null;                  // "criar", "remover", "ver", "outros")
   String instante_execução = null;          //"agora", "em_batch" (só usado "agora")
   String resposta_servidor = null;          // "ok", "nok"
}
```

TrataIGU

Quando o botão de executar ação (que foi criado em **TabbedESN7**) é "clicado" pelo operador, o evento associado é identificado em **ButtonExecutar** (método de tratamento de eventos "**actionPerformed")** e é acionado um método de mesmo nome da classe **TrataIGU**.

```
package client;
import java.rmi.*;
import compute.*;
public class TrataIGU
{
 public Esn7 TrataIGU(String servidor, Esn7 esn7)
  {
  Esn7 resultado = new Esn7();
  if (System.getSecurityManager() == null)
   {
   System.setSecurityManager(new RMISecurityManager());
   }
  try
   {
   String name = "//" + servidor + "/Compute";
   Compute comp = (Compute) Naming.lookup(name);

// O esn7 a ser tratado no servidor deve ser passado para a task quando ela for
ativada

   TrataESN7 task = new TrataESN7(esn7);
   resultado = (Esn7) (comp.executeTask(task));
  } catch (Exception e)
```

```
        {
         System.err.println("TrataIGU exception: " + e.getMessage());
         e.printStackTrace();
        }
      return resultado;
     }
    }
```

OBS.: baseado num exemplo de cálculo do valor de "Pi" (¶) que pode ser encontrado no "tutorial" de RMI da Ref. [6].

O método **TrataIGU** prevê, como parâmetros de entrada, o endereço IP do servidor em que a tarefa deve ser executada e os parâmetros do ESN7 a ser tratado.
Tal como no servidor é usado um gerente de segurança ("**SecurityManager**") para controle dos acessos aos recursos do sistema.
Para invocar o objeto remoto que foi registrado no servidor, ele é localizado através de uma operação **"lookup"** em que o parâmetro de entrada é um nome composto do IP do servidor e o nome do objeto remoto. Note-se que na composição do nome do objeto remoto não entra nada específico da tarefa a ser executada mas da interface genérica "**Compute**".
Uma instância da "**Task**" **TrataESN7** é criada com os parâmetros do ESN7 a ser tratado. Espera-se que a tarefa seja executada e o resultado seja retornado desde o servidor remoto, via **TrataIGU,** para ser tratado em **ButtonExecutar**. Por simplificação, o tratamento do resultado se resume a mostrar tal resultado na tela do cliente.
Um tratamento simples das exceções é previsto em que somente é feita a exibição no vídeo do cliente da mensagem relativa à exceção.

TrataESN7

É acessada, desde **TrataIGU,** após o operador ter clicado no botão de executar ação na IGU. **ButtonExecutar** imprime na tela do cliente o conteúdo do objeto ESN7 que deve se passado para o servidor associado à central telefônica. **TrataESN7** emula a mudança na base de dados da central e a resposta da central. Para isto envia um "string" de volta ao cliente com o resultado preestabelecido como "o.k." e ecoa o objeto recebido do cliente na tela do servidor.
TrataESN7 fica no diretório do cliente e pode ter sido feita após o servidor ter sido colocado em execução. Implementa a tarefa que será serializada e enviada para ser executada no servidor. É a tarefa que trata a ação que o operador deseja que seja executada, para o ESN7, no servidor associado à central telefônica remota. Para isto **TrataESN7** implementa o método **"execute"** definido na interface "**Task**". Após tratar a ação relativa ao ESN7 que será executada no servidor providencia a resposta da execução da ação para ser colocada na IGU no cliente. Quando acessada recebe um parâmetro de entrada do tipo ESN7 – que já define, em si, a ação a ser executada, e os detalhes do que deve ser feito. Retorna um parâmetro do tipo ESN7 com o resultado da ação executada no servidor.
Foi previsto um método de **TrataESN7** (denominado "**trata_umEsn7**") no qual deve ser colocado tudo o que efetivamente deve ser feito no servidor.

```
package client;
import compute.*;
public class TrataESN7 implements Task
{
/* Notas:
  Emula o envio do CHM específico para central (imprime objeto na tela).
  Emula resposta da central: envia string com resultado ok e
  ecoa objeto recebido.
*/
 private Esn7 esn7;
 public TrataESN7 (Esn7 esn7)
 {
  this.esn7=esn7;
 }
 public Object execute()
 {
  return trata_umEsn7 (esn7);
 }
 public static Esn7 trata_umEsn7 (Esn7 esn7)
 {
/* NOTA:
 Aqui é onde deve entrar o que efetivamente deve ser feito no servidor. Por exemplo,
se se tratar de um mapeamento para comandos CHM específicos de cada central,
tal conversão deve entrar aqui. No entanto, nesta implementação, somente são
impressos, no servidor, os dados do ESN7 escolhidos na IGU pelo cliente.
*/
  Esn7 resultado = esn7;
  resultado.resposta_servidor="ok";
  System.out.println("slc="+resultado.slc);
  System.out.println("canal_esn7="+resultado.canal_esn7);
  System.out.println("hw_terminal_dados="+resultado.hw_terminal_dados);
  System.out.println("hw_terminal_sn7="+resultado.hw_terminal_sn7);
  System.out.println("tipo_cps="+resultado.tipo_cps);
  System.out.println("opc="+resultado.opc);
  System.out.println("dpc="+resultado.dpc);
  System.out.println("id_esn7="+resultado.id_esn7);
  System.out.println("id_cesn7="+resultado.id_cesn7);
  System.out.println("id_rsn7="+resultado.id_rsn7);
  System.out.println("tipo_ação="+resultado.tipo_ação);
  System.out.println("instante_execução="+resultado.instante_execução);
  System.out.println("resposta_servidor="+resultado.resposta_servidor);
  return resultado;
 }
}
```

OBS.: baseado num exemplo de cálculo do valor de "Pi" (¶ que pode ser encontrado no "tutorial" de RMI da Ref. [6].

Há várias soluções possíveis de mapeamento para um caso real, conforme será comentado no item 4.10.2, que trata as arquiteturas do servidor. Sem entrar em muitos detalhes sobre a arquitetura do servidor remoto efetivamente usada, uma alternativa inicial que pode ser feita é uma adaptação entre o conteúdo do objeto ESN7 recebido do cliente e os comandos CHM específicos de cada central. Isto poderia ser feito conhecendo e emulando o protocolo da interface CHM da central ou tendo acesso direto ao código fonte desta interface. Poderia também ser feito diretamente na base de dados da central, sem emular os comandos de CHM da central, acessando diretamente tal base de dados, se a interface da central assim o permitir. Havendo acesso ao código fonte da interface de CHM da central esta adaptação pode ser feita através de um "***wrapper***", de forma que o acesso à base de dados da central seja "*mapeado*" desde a linguagem Java até a linguagem em que a interface de CHM da central foi escrita, desde que, para isto, a linguagem usada para o desenvolvimento da interface de CHM seja uma daquelas que já estão previstas como compatíveis com JNI. Outras possibilidades serão apresentadas mais a frente no item de arquiteturas do servidor. No entanto, nesta implementação tudo é simulado. O servidor somente imprime os dados do ESN7 escolhidos na IGU pelo cliente e retorna um eco do que foi recebido do cliente. Os parâmetros recebidos do ESN7 são impressos no servidor e o resultado, no retorno, previamente definido como "o.k.", é preenchido pelo servidor e retornado para o cliente.
Embora **TrataESN7** pudesse ser genérica, só a criação de um ESN7 é considerada (e simplificadamente, conforme exposto). Para outras situações há que detalhar outros "clientes" – e IGU – de forma análoga ao que foi feito.

4.9 2ª Implementação da Interação Cliente / Servidor: Uso de IDL

É baseada em vários exemplos adaptados principalmente a partir do tutorial de IDL da página da Sun.
O cliente usa a mesma IGU que foi usada para a implementação com RMI. Depois de o operador escolher os parâmetros necessários para a criação do ESN7, ele pressiona o botão de executar a ação. O cliente providencia a execução da ação através da interface IDL com o servidor remoto.
O código fonte usado não será apresentado aqui porque, ao contrário da implementação feita em RMI, a sua omissão aqui não compromete a compreensão. Somente a definição da interface simplificada que será apresentada.

4.9.1 Definição da Interface em IDL

As definições são muito simples. Possui um módulo único (itf) com uma única interface (Esn7) e uma única operação (criarEsn7). Esta operação recebe os parâmetros de entrada relevantes e retorna um resultado único. Tanto os parâmetros de entrada quanto o resultado são "strings".

```
Module itf
{
 interface Esn7
 {
  string criarEsn7(in string hw_terminal_dados
                   in string hw_terminal_ESN7
                   in string formato_spc
                   in string opc
                   in string dpc
                   in string canal_esn7
                   in string slc
                   in string id_ESN7
                   in string id_CESN7
                   in string id_rota_SN7
                   in string ip
                   );
 };
};
```

Uma implementação melhorada do protótipo deveria prever outros parâmetros de entrada para a operação "criarEsn7" e poderia também prever outras operações tais como "removerEsn7", "verEsn7", etc.

O compilador ***idltojava*** – encontrado na Ref. [145] – gera 5 arquivos a partir do arquivo IDL. Um dos arquivos contém a versão em Java da interface definida em IDL; outro contém o esqueleto do servidor; outro o "stub" do cliente; os 2 outros possuem outras funções auxiliares. Outros detalhes podem ser vistos no tutorial de IDL da página da Sun e na descrição do *"package"* **org.omg.CosNaming** nas especificações da API Java que podem ser encontradas na Ref. [148].

4.9.2 O Cliente

É o cliente que cria a IGU e que ativa o servidor. Enquanto o operador faz suas escolhas na IGU o cliente as recebe e espera até que o operador solicite a execução da ação de criação do ESN7. Na IGU há um evento associado à ação de o operador "*clicar*" no botão de execução da ação de criação do ESN7. Este evento dispara uma ação (***actionPerformed***), através da qual, uma instância de objeto ORB local é criada e iniciada. O mapeamento da API CORBA da OMG na linguagem Java é feita através do pacote (*package*) ***org.omg.CORBA***. A iniciação da objeto ORB é feita através da classe ***org.omg.CORBA.ORB***. Esta classe permite fornecer valores predefinidos de propriedades e parâmetros de ambiente para o ORB. O método *"**init**"* permite criar uma nova instância do ORB. O endereço IP passado na linha de comando é usado como parâmetro do método *"init"* para a iniciação do objeto ORB. Nenhuma propriedade específica da aplicação é usada *("null")*.
O objeto ORB recebe e providencia todas as solicitações do cliente. Cuida do "encapsulamento" ("*marshaling*") e das interações com o protocolo IIOP.
O cliente precisa localizar onde está o servidor com o recurso para criar o ESN7. Para isto o cliente precisa obter a referência inicial que identificará o servidor que atenderá à criação do Esn7. Para obter a referência inicial do objeto que o cliente quer invocar no servidor é usado

o serviço de nomes de objetos comuns (***COS Naming Service***) definido pela OMG. O serviço de nomes para o IDL da Java é fornecido através do pacote (*"package"*) ***org.omg.CosNaming***. Neste pacote, a interface ***NamingContext*** ("contexto de nomeação") permite a definição de um contexto de nomes (um objeto que contém um conjunto de ligações de objetos em que cada nome é único). O método ***resolve_initial_references*** da classe *org.omg.CORBA.ORB* permite a recuperação da referência de um objeto específico a partir do conjunto de nomes de serviços iniciais disponíveis ("contexto de nomeação"). Para isto recebe o nome do serviço inicial como parâmetro de entrada e devolve a referência de objeto associada ao nome dado. O contexto inicial do nome é obtido porque o "string" ***"NameService"*** ("serviço de nome") é definido para todos os ORBs CORBA. O método *resolve_initial_references* recebe o "string" *"NameService"* como parâmetro de entrada, resolve a referência específica de objeto a partir dos nomes de serviços iniciais disponíveis e retorna o contexto inicial do nome, isto é, um objeto de referência do servidor de nome (que tenha sido previamente associado ao nome dado como parâmetro de entrada).
O serviço de nomes pode ser implementado de várias formas. Para tratar isto, a classe ***org.omg.CosNaming.NameComponent*** tem já definido um construtor que permite associar uma identidade de um objeto a um *"path"* ("caminho") completo de qualquer arquivo ou disco do sistema. Uma instância da classe *NameComponent* é usada para encontrar o servidor para o objeto associado à interface Esn7.
A interface "*NamingContext*" (contexto de nomeação) permite o uso de um conjunto de "ligações de nomes" (*"name bindings"*) em que cada nome é único. Na interface "*NamingContext*" (contexto de nomeação) há um método ***("resolve")*** para recuperar um objeto que foi previamente ligado a um nome para um contexto dado. Um dos arquivos gerados pelo compilador *idltojava* ***(Esn7Helper)*** é usado para ajudar na obtenção desta referência de nome.
A referência de nome obtida é usada para invocar a operação (***criarEsn7***) no objeto CORBA. Devido à simplificação feita na definição da interface, para fins de testes, somente o canal do ESN7 ***(canalEsn7)*** está sendo efetivamente passado como parâmetro na interface. O processo de inserção dos demais parâmetros é idêntico ao feito para o número do canal. A IGU já os prevê e trata.
Após o servidor Esn7 ter sido acessado, o resultado é impresso na tela e as eventuais exceções são tratadas.

4.9.3 O servidor

A interface definida em IDL e gerada pelo compilador *idltojava* é implementada em uma classe "servente" (***Esn7Servant***). Ela contém a implementação do objeto do servidor (instância do objeto associado à classe "*Esn7Servant*"). Possui o único método definido pela interface Esn7 *(criarEsn7)* e que tão-somente imprime na tela do servidor uma mensagem com o número do canal recebido na invocação e retorna um string com o texto *"Esn7 criado OK !!!"*.
O servidor cria e inicia o ORB local que precisa para funcionar. Para isto usa o endereço IP do registro de nomes – que é informado através da linha de comando de ativação do servidor.
A iniciação da objeto ORB no servidor é feita, tal qual no cliente, através da classe *org.omg.CORBA.ORB*.
O servidor instancia o ORB local e "conecta" a instância do objeto "servente" ao ORB para que ele possa ser localizado quando uma invocação for recebida pelo ORB. Para isto é usado o método ***"connect"*** (conectar) definido na classe *org.omg.CORBA.ORB*. O novo

"servente" criado é passado como parâmetro de entrada em *"connect"*. Após isto o "servente" ***("servant")*** já pode receber invocações remotas.
O servidor precisa viabilizar uma referência de objeto do serviço de nome para permitir que as invocações da interface Esn7 sejam efetivamente encaminhadas para o objeto associado à implementação desta interface no servidor. O servidor usa o serviço de nomes e assim torna a implementação do objeto no servidor disponível para o cliente. O serviço de nomes para o IDL da Java é usado no servidor de forma semelhante à que foi feita no cliente. No servidor, no entanto, é usado o método de ligar ***("rebind")*** a referência do objeto ao serviço de nomes. Cabe ao cliente, por sua vez, usar o método de resolução de nomes.
Após isto o servidor imprime uma mensagem para indicar na tela que já está pronto para receber as solicitações do operador e fica, a partir daí, aguardando as invocações do cliente. As possíveis exceções são tratadas de forma simples: impressão na tela de uma mensagem de erro e do ***"StackTrace"*** ("rastreamento da pilha").

4.10 Comentários

4.10.1 A Definição da Interface

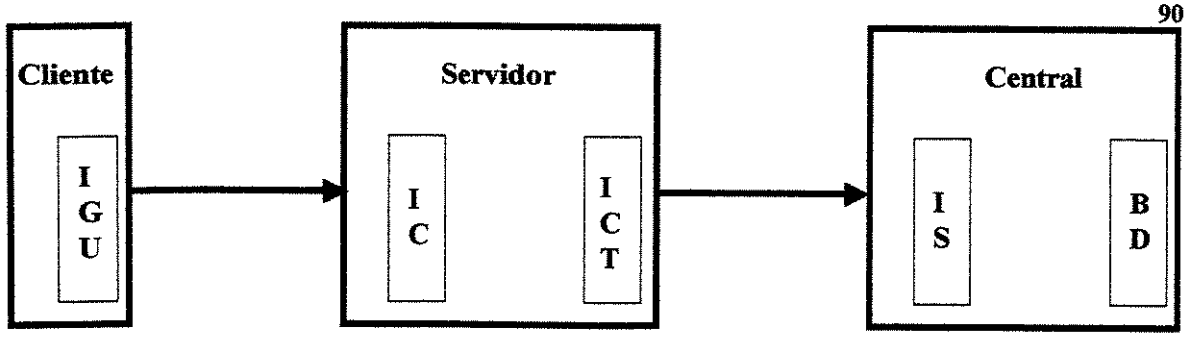
Usar todos os parâmetros como strings, embora trivial, facilita a construção de editores para o envio de comandos em ***"batch"*** para a construção de vários ESN7. Por isto é interessante que a definição da interface não seja feita com estruturas rebuscadas. Ser simples é de grande valia quando se tem que criar muitos ESN7 de uma só vez. Nestas situações é muito ineficiente interagir com a IGU para cada ESN7 a ser criado. É preferível editar o que se quer em um editor de texto ou em uma IGU de alto nível em que somente se entre com os parâmetros significativos e deixar que esta IGU crie um arquivo de *"batch"* para ser enviado para atuação na BD da central. Um programa pode enviar os dados, analisar os resultados e, em caso de erro, ou tomar alguma ação que possa ser antecipadamente prevista ou simplesmente anotar a parte os ESN7 que não puderam ser criados, ao lado da mensagem de erro recebida da central. Parece simples mas isto seria de grande valia para os fabricantes e também para as operadoras, principalmente se uma interface padronizada permitisse tal tipo de interação com vários tipos de centrais telefônicas. É muito comum em implantações ou atualizações de centrais serem dados milhares de comandos de CHM para se fazer isto, o que leva dias e é um trabalho muito cansativo, tedioso e feito por mão-de-obra muito especializada que precisa ficar atenta quase que a cada resultado de cada comando dado.

4.10.2 Arquiteturas Possíveis Para o Servidor

O servidor precisa estar em interação direta ou indireta com a BD da central telefônica. Isto sugere algumas arquiteturas ***"n-tier"*** conforme ilustrado a seguir.

Por um lado o servidor há que prover o interfuncionamento com o cliente (IC) e por outro com a BD da central telefônica (IBD).

Uma forma genérica de fazer esta interconexão entre o servidor e a central está mostrada na figura abaixo.



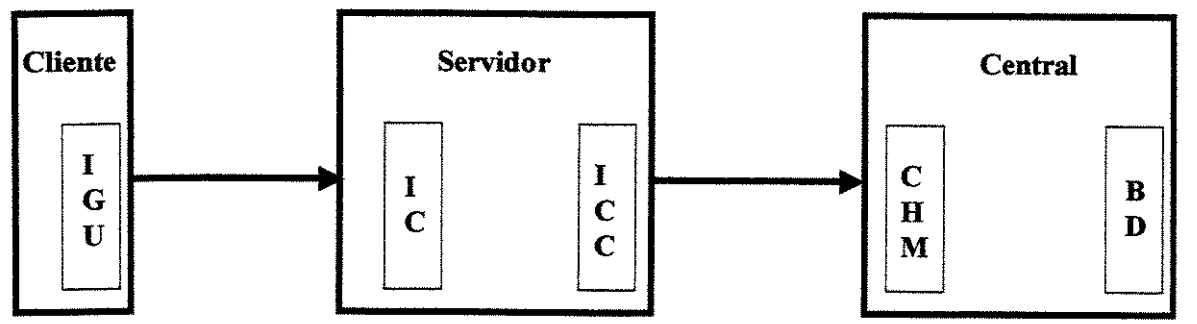
BD	Base de Dados da central telefônica
IC	Interfuncionamento com o Cliente
ICT	Interfuncionamento com a central telefônica
IGU	Interface Gráfica do Usuário
IS	Interfuncionamento com o servidor

Figura # 4.11 - Modelo Genérico de Interligação Entre o Servidor e a Central

Esta figura é bastante simplificada. Mesmo dentro do cliente, além da IGU ainda há uma parte para a interação com o servidor.

A configuração do servidor irá depender da forma utilizada para a interconexão com a BD da central.

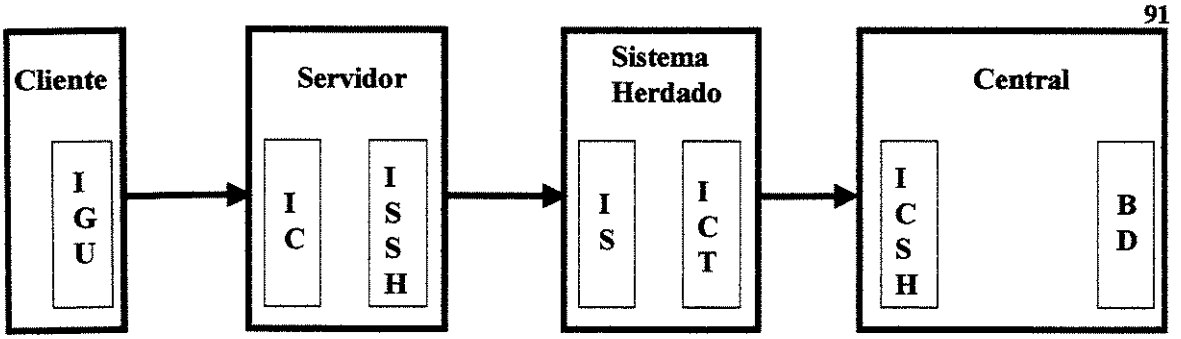
Se esta interconexão for feita usando os próprios comandos de CHM já existentes, isto é, quando o ICT é feito através de CHM a arquitetura do servidor pode ser como mostrado na figura abaixo.



BD	Base de Dados da central telefônica
CHM	Comandos Homem-Máquina da central telefônica
IC	Interfuncionamento com o Cliente
ICC	Interfuncionamento com os comandos de comunicação homem-máquina da central (CHM)
IGU	Interface Gráfica do Usuário

Figura # 4.12 - Interligação Entre o Servidor e a Central Através de CHM

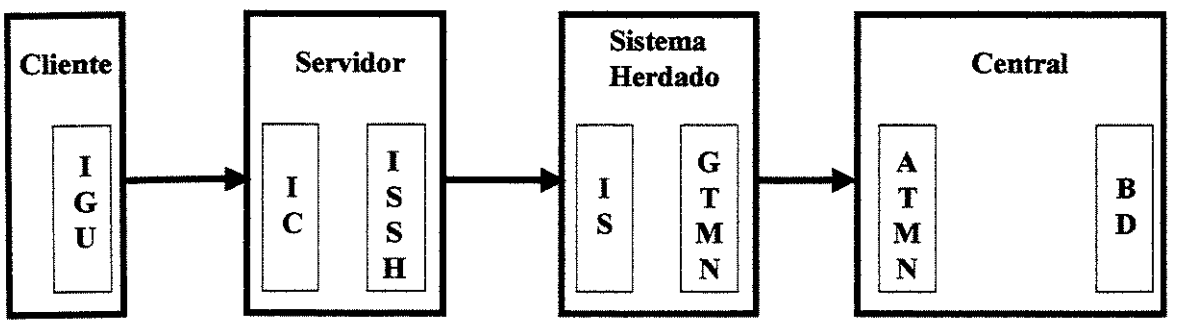
No entanto, se esta interconexão for feita usando algum outro sistema herdado que se inteconecte com a BD da central telefônica, a arquitetura do servidor será como mostrada na figura abaixo.



BD	Base de Dados da central telefônica
IC	Interfuncionamento com o Cliente
ICSH	Interfuncionamento da Central com o Sistema Herdado de interação com a central
ICT	Interfuncionamento com a Central Telefônica
IS	Interfuncionamento com o Servidor
ISSH	Interfuncionamento do Servidor com o Sistema Herdado de interação com a central
IGU	Interface Gráfica do Usuário

Figura # 4.13 - Interligação Entre o Servidor e a Central Através de Sistema Herdado

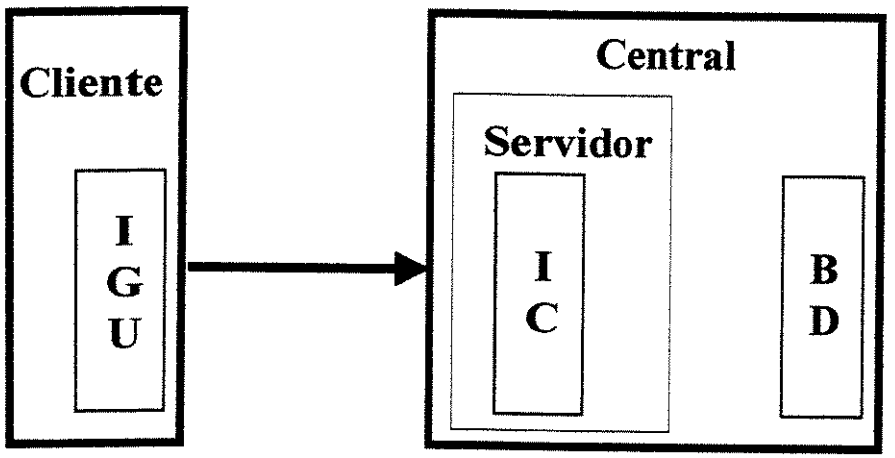
Este sistema herdado pode ser um sistema qualquer ou, em especial, um conjunto formado por um agente e um gerente TMN, conforme mostrado a seguir. Neste caso, por simplificação, estamos supondo que haja somente um agente e somente um gerente e que este último esteja implementado como um módulo interno da central.



ATMN	Agente TMN (interno à central telefônica)
BD	Base de Dados da central telefônica
GTMN	Gerente TMN
IC	Interfuncionamento com o Cliente
IS	Interfuncionamento com o Servidor
ISSH	Interfuncionamento do Servidor com o Sistema Herdado de interação com a central
IGU	Interface Gráfica do Usuário

Figura # 4.14 - Interligação Entre o Servidor e a Central Quando o Sistema Herdado é Composto de Um Agente e Um Gerente TMN

No entanto, o melhor caso de interconexão é aquele em que o servidor já esteja implementado como parte da central telefônica, tal qual mostrado a seguir.



BD	Base de Dados da central telefônica
IC	Interfuncionamento com o Cliente
IGU	Interface Gráfica do Usuário

Figura # 4.15 - Interligação Direta Entre o Cliente e a Central com o Servidor Dentro da Central

Este é exatamente o modelo que depende de estudos adicionais para padronizar as interfaces entre o cliente e o servidor. Este modelo só poderá ser usado quando tais interfaces estiverem devidamente padronizadas. Aqui estamos essencialmente propondo a adoção de um conjunto de interfaces IDL, uma ou mais de uma para cada área de uma central telefônica, tais como a interface que foi usada no protótipo de criação de ESN7 objeto desta dissertação.

Note-se que mapear o servidor na BD da central passa por ter acesso à implementação desta base de dados ou pelo menos à ICHM específica de cada central. Sem dúvida alguma, tudo isto é uma informação confidencial de cada fabricante. Por tudo isto, não é objeto desta dissertação mapear o servidor na BD da central telefônica, mesmo porque, tal mapeamento pode variar, conforme mostrado anteriormente. O que se quer enfatizar nesta dissertação é a padronização das interfaces, as vantagens disto e a facilidade que hoje se encontra para o desenvolvimento de clientes amigáveis usando os recursos atuais da tecnologia da informação. Construir um sistema simples, embora incompleto, foi a forma usada nesta dissertação para verificar isto. Nos próximos capítulos são apresentadas as medições feitas nos protótipos, as conclusões tiradas e são também apresentadas sugestões para ser dada continuidade a este trabalho.

Capítulo 5 – Resultados Experimentais

5.1 As Medições Que Foram Feitas

Conforme dito anteriormente, aproveitar as máquinas existentes é um dos maiores objetivos a atingir, após a padronização da interface entre o cliente e o servidor. Por isto, muitas medidas de desempenho foram feitas para ver que tipos de máquinas poderiam ser aproveitadas.
Ambas implementações (RMI e IDL) foram inicialmente testadas em duas máquinas:

- 1ª. máquina (a pior – **P**): Pentium 133 MHz com 16 MB, Windows 95;
- 2ª. máquina (a melhor – **M**): Pentium 300 MHz com 128 MB, Windows 98.

Estas máquinas foram utilizadas em 3 configurações conforme a tabela abaixo.

Configuração	
#	**Descrição**
C1	**C, S e RN em P**
C2	**C, S e RN em M**
C3	**C em M; S e RN em P**
C4	**C em P; S e RN em M**

Abreviaturas
C = Cliente
S = Servidor
RN = Registro de Nomes
P = máquina pior (1ª)
M = máquina melhor (2a.)

Tabela # 5.1 – Configurações de Testes

Os seguintes tipos de tempos foram observados ao executar o protótipo:

Tipos de Tempos Observados	
C_e	Carga inicial do cliente ("cliente entrar")
$ESN7_1$	Desenhar tela de criação de ESN7 (1ª. vez)
$ESN7_2$	Desenhar tela de criação de ESN7 (após 1ª. vez)
E_{x1}	Resposta ao pressionar o botão "Executar" pela 1a. vez
E_{x2}	Resposta ao pressionar o botão "Executar" após a 1a. vez

Tabela # 5.2 - Tipos de Tempos Observados ao Executar o Protótipo

Os tempos foram medidos deste o instante em que ocorreu uma ação do operador até a obtenção da resposta associada. A média usada foi a média aritmética simples. Os valores foram arredondados. Os resultados obtidos estão apresentados nas tabelas abaixo. Nenhuma outra aplicação estava sendo executada ao mesmo tempo em que os testes foram feitos. A rede usada foi Ethernet 10 BT, sem servidor de nomes, apenas um hub e as 2 máquinas com endereços IP fixos.

Abreviaturas: **C** = Cliente; **S** = Servidor; **RN** = Registro de Nomes;
P = máquina pior (1ª); **M** = máquina melhor (2a.)
$\mathbf{C_e}$ = Carga inicial do cliente ("cliente entrar")
$\mathbf{ESN7_1}$ = Desenhar tela de criação de ESN7 (1ª. vez)
$\mathbf{ESN7_2}$ = Desenhar tela de criação de ESN7 (após 1ª. vez)
$\mathbf{E_{x1}}$ = Resposta ao pressionar o botão "Executar" pela 1ª. vez
$\mathbf{E_{x2}}$ = Resposta ao pressionar o botão "Executar" após a 1ª. vez

As conclusões imediatas que foram tiradas com base nas observações feitas são apresentadas logo a frente. Há algumas extrapolações que podem ser feitas e que estão apresentadas junto às sugestões mais a frente.

5.1.1 Primeira Implementação (RMI)

Configuração C1 : (C, S e RN em P)	Tipo de Tempos Observados					
		C_e	$ESN7_1$	$ESN7_2$	E_{x1}	E_{x2}
	T_{max}	≈ 84 s	≈ 39 s	≈ 65 s	≈ 126 s	≈ 134 s
	T_{min}	≈ 45 s	≈ 25 s	≈ 27 s	≈ 97 s	≈ 70 s
	Média	≈ 71 s	≈ 30 s	≈ 42 s	≈ 111 s	≈ 99 s
	$Qtd_{medidas}$	3	3	5		6

Tabela # 5.3 - Tempos Observados na Primeira Implementação (RMI)

(*) Nota: não se investiu mais tempo com a obtenção de um número maior de amostras; desempenho muito ruim; configuração descartada.

Configuração C2 : (C, S e RN em M)	Tipo de Tempos Observados					
		C_e	$ESN7_1$	$ESN7_2$	E_{x1}	E_{x2}
	T_{max}	≈ 13 s	≈ 4 s	≈ 2 s	≈ 2 s	≈ 2 s
	T_{min}	≈ 5 s	≈ 2 s	≈ 1 s	≈ 1 s	≈ 1 s
	Média	≈ 6 s	≈ 2 s	≈ 2 s	≈ 2 s	≈ 1 s
	$Qtd_{medidas}$	10	8	11	7	12

Tabela # 5.4 - Tempos Observados na Primeira Implementação (RMI) – C2

Configuração C3 : (C em M; S e RN em P)	Tipo de Tempos Observados					
		C_e	$ESN7_1$	$ESN7_2$	E_{x1}	E_{x2}
	T_{max}	≈ 15 s	≈ 3 s	≈ 2 s	≈ 4 s	≈ 3 s
	T_{min}	≈ 4 s	≈ 2 s	≈ 1 s	≈ 1 s	≈ 1 s
	Média	≈ 6 s	≈ 2 s	≈ 2 s	≈ 3 s	≈ 2 s
	$Qtd_{medidas}$	10	6	13	6	13

Tabela # 5.5 - Tempos Observados na Primeira Implementação (RMI) – C3

Configuração C4: (C em P; S e RN em M)	Tipo de Tempos Observados					
		C_e	$ESN7_1$	$ESN7_2$	E_{x1}	E_{x2}
	T_{max}	≈ 41 s	≈ 11 s	≈ 7 s	≈ 45 s	≈ 32 s
	T_{min}	≈ 33 s	≈ 8 s	≈ 3 s	≈ 39 s	≈ 10 s
	Média	≈ 35 s	≈ 9 s	≈ 4 s	≈ 41 s	≈ 17 s
	$Qtd_{medidas}$	5	5	17	5	17

Tabela # 5.6 - Tempos Observados na Primeira Implementação (RMI) – C4

Dado o baixo desempenho da 1ª máquina como cliente foram feitas algumas medidas adicionais para verificar a possível causa deste baixo desempenho. O programa foi então testado em uma 3ª máquina semelhante à 1ª mas com mais memória RAM:

- 1ª. máquina: Pentium 133 MHz com 16 MB, Windows 95;
- 2ª. máquina: Pentium 300 MHz com 128 MB, Windows 98;
- 3ª. máquina: Pentium 133 MHz com 32 MB, Windows 95.

A 3ª máquina foi observada for a de uma ligação em rede, trabalhando de forma autônoma onde ela executava somente a função de cliente. Esta configuração foi chamada de **Configuração C5**.

Configuração C5 : (só Cliente na 3ª. máquina)	**Tipo de Tempos Observados**					
		C_e	$ESN7_1$	$ESN7_2$	E_{x1}	E_{x2}
	T_{max}	≈ 19 s	≈ 5 s	≈ 4 s	-	-
	T_{min}	≈ 15 s	≈ 5 s	≈ 2 s	-	-
	Média	≈ 17 s	≈ 5 s	≈ 3 s	-	-
	$Qtd_{medidas}$	5	5	14	-	-

Tabela # 5.7 - Tempos Observados na Primeira Implementação (RMI) – C5

Abaixo estão os valores médios dos tempos relativos às observações feitas na configuração **C5** em comparação como os valores médios das outras duas máquinas: P em C4 e M em C3. Os valores foram obtidos das tabelas anteriormente apresentadas.

Somente o cliente sendo executado na máquina	**1ª. máquina (P em C4)**	**2ª. máquina (M em C3)**	**3ª. máquina (conformeC5)**
Carga inicial do cliente do protótipo (C_e)	≈ 35 s	≈ 6 s	≈ 17 s
Desenhar tela criação ESN7 1ª. vez ($ESN7_1$)	≈ 9 s	≈ 2 s	≈ 5 s
Desenhar tela criação ESN7 após 1ª. vez $ESN7_2$)	≈ 4 s	≈ 2 s	≈ 3 s

Tabela # 5.8 – Comparação dos Tempos Observados na 1a. Implementação (RMI)
(*) Nota: como não foi usada a rede não há medida de resposta após pressionar "Executar" (E_{x1} e E_{x2}).

5.1.2 Segunda Implementação (IDL)

Configuração C1 : (C, S e RN em P)	**Tipo de Tempos Observados**					
		C_e	$ESN7_1$	$ESN7_2$	E_{x1}	E_{x2}
	T_{max}	≈ 56 s	≈ 18 s	≈ 42 (19) s	≈ 57 s	≈ 40 s
	T_{min}	≈ 41 s	≈ 12 s	≈ 8 s	≈ 49 s	≈ 31 s
	Média	≈ 45 s	≈ 15 s	≈ 17 (14) s	≈ 53 s	≈ 35 s
	$Qtd_{medidas}$	5	5	9 (8) (*)	5	9

Tabela # 5.9 - Tempos Observados na 2a. Implementação (IDL) – C1

(*) **Nota:** houve uma única ocorrência de $ESN7_2$ = 42 s na tabela acima (C1). Os valores for a do parêntesis na coluna de $ESN7_2$ consideram esta ocorrência; os valores dentro dos parêntesis desconsideram-na.

Configuração C2 : (C, S e RN em M)	Tipo de Tempos Observados					
		C_e	$ESN7_1$	$ESN7_2$	E_{x1}	E_{x2}
	T_{max}	≈ 11 s	≈ 3 s	≈ 2 s	≈ 2 s	≈ 2 s
	T_{min}	≈ 4 s	≈ 2 s	≈ 1 s	≈ 1 s	≈ 1 s
	Média	≈ 5 s	≈ 2 s	≈ 1 s	≈ 2 s	≈ 1 s
	$Qtd_{medidas}$	7	4	5	4	5

Tabela # 5.10 - Tempos Observados na 2a.Implementação (IDL) – C2

Configuração C3 : (C em M; S e RN em P)	Tipo de Tempos Observados					
		C_e	$ESN7_1$	$ESN7_2$	E_{x1}	E_{x2}
	T_{max}	≈ 12 (5)s	≈ 3 s	≈ 1 s	≈ 24 s	≈ 17 s
	T_{min}	≈ 3 s	≈ 1 s	≈ 1 s	≈ 15 s	≈ 13 s
	Média	≈ 6 (4)s	≈ 2 s	≈ 1 s	≈ 17 s	≈ 14 s
	$Qtd_{medidas}$	5 (4) (*)	5	20	5	15

Tabela # 5.11 - Tempos Observados na 2a.Implementação (IDL) – C3

(*) Nota: houve uma única ocorrência de C_e = 12 s na tabela acima (C3). Os valores for a do parêntesis na coluna de $ESN7_2$ consideram esta ocorrência; os valores dentro dos parêntesis desconsideram-na.

Configuração C4 : (C em P; S e RN em M)	Tipo de Tempos Observados					
		C_e	$ESN7_1$	$ESN7_2$	E_{x1}	E_{x2}
	T_{max}	≈ 46 s	≈ 18 s	≈ 13 s	≈ 24 s	≈ 5 s
	T_{min}	≈ 38 s	≈ 12 s	≈ 2 s	≈ 21 s	≈ 1 s
	Média	≈ 41 s	≈ 15 s	≈ 6 s	≈ 22 s	≈ 2 s
	$Qtd_{medidas}$	6	6	11	4	11

Tabela # 5.12 - Tempos Observados na 2a.Implementação (IDL) – C4

Foi também feita para a implementação em IDL a mesma comparação feita para a implementação RMI entre a 1a e a 3a máquinas (**Configuração C5**). Ver resultados abaixo.

Configuração C5 : (só Cliente na 3ª. máquina)	Tipo de Tempos Observados					
		C_e	$ESN7_1$	$ESN7_2$	E_{x1}	E_{x2}
	T_{max}	≈ 19 s	≈ 6 s	≈ 3 s	-	-
	T_{min}	≈ 10 s	≈ 4 s	≈ 2 s	-	-
	Média	≈ 14 s	≈ 5 s	≈ 3 s	-	-
	$Qtd_{medidas}$	5	5	16	-	-

Tabela # 5.13 - Tempos Observados na 2a.Implementação (IDL) – C5

Abaixo estão os valores médios dos tempos relativos às observações feitas na configuração **C5** em comparação como os valores médios das outras duas máquinas: P em C4 e M em C3. Os valores foram obtidos das tabelas anteriormente apresentadas.

Somente o cliente sendo executado na máquina	1ª. máquina (P em C4)	2ª. máquina (M em C3)	3ª. máquina (conformeC5)
Carga inicial do cliente do protótipo (C_e)	≈ 41 s	≈ 6 s	≈ 14 s
Desenhar tela criação ESN7 1ª. vez ($ESN7_1$)	≈ 15 s	≈ 2 s	≈ 5 s
Desenhar tela criação ESN7 após 1ª. vez $ESN7_2$)	≈ 6 s	≈ 1 s	≈ 3 s

Tabela # 5.14 - Comparação dos Tempos Observados na Implementação em IDL

(*) **Nota**: como não foi usada a rede não há medida de resposta após pressionar "Executar" (E_{x1} e E_{x2}).

5.2 Adequação das Máquinas paras os Protótipo Desenvolvidos

Embora existam algumas diferenças de tempos entre as observações relativas ao protótipo RMI e ao IDL, para ambas implementações, quando a 1ª. máquina desempenhou os 3 papéis (cliente, servidor e registro de nomes) percebeu-se pela configuração C1 que o protótipo demorava muito tempo para ser carregado, para atender a uma solicitação do operador, executá-la e enviar a resposta. O tempo era suficiente para incomodar qualquer usuário. O baixo desempenho da 1ª. máquina na configuração C1 e na C4 mostra que esta máquina é inadequada para a execução do cliente. Mesmo quando a 1ª. máquina só exerceu o papel de cliente (configuração C4) o desempenho não foi bom. A 3ª. máquina, tanto na observação feita para RMI, quanto na de IDL, apresentou um desempenho bem melhor que o da 1ª máquina na configuração C4 mas ainda foi um desempenho insatisfatório para a função de cliente. As diferenças observadas entre a 1ª. e a 3ª. máquinas levam a concluir que o baixo desempenho (principalmente da 1ª máquina) poderia ser, em grande parte devido à pouca quantidade de memória disponível para aplicações gráficas (apenas 16 MB e 32 MB, respectivamente). Poderia ser devida também a um possível baixo desempenho do processador, o que condiz com as recomendações da SUN a respeito do ambiente necessário para executar tal tipo de aplicações: pelo menos 32 Mb de RAM e pelo menos Pentium 166 MHz.

Com relação aos testes feitos na Configuração C1, não se trata de uma sobrecarga irreal submeter a mesma máquina às 3 funções de forma concorrente: registro, servidor e cliente porque isto pode ocorrer de fato se um terminal de ICHM de uma central for usado para fazer a interface com a base de dados da central e ao mesmo tempo for usado para executar o cliente e o servidor Java do protótipo. Isto poderia permitir que um operador interagisse com o cliente Java e o servidor Java interagisse com o controlador da ICHM da central. Assim, um operador poderia operar tal equipamento sem conhecer a sintaxe específica da linguagem de ICHM da central com que estivesse interagindo.

A 1ª máquina teve bom desempenho tanto para RMI quanto para IDL quando executando somente as funções de registro de nomes e de servidor (configuração C3). Há no entanto que lembrar que o servidor deste protótipo é muito simples em ambas implementações.

Pelos resultados das configurações C2 e C3 percebe-se que a máquina 2 teve bom desempenho como cliente em qualquer situação mesmo quando configurada com as 3 funções (cliente, servidor e registro de nomes) de forma concorrente.

O desempenho da máquina 1 quando usada para as funções de servidor e de registro de nomes (C3) foi melhor na implementação de RMI do que na de IDL quando os tempos de respostas aos comandos de criação do ESN7 são considerados (E_{x1} e E_{x2}). Como a interface

gráfica de usuário é a mesma, isto nos induz à conclusão que a implementação em IDL exigiu mais do equipamento que aquela em RMI.
Para ambos protótipos, em praticamente todas as configurações (exceto na C1), o tempo gasto na 1ª solicitação do operador foi maior que o gasto para a seguinte, tanto para $\mathbf{ESN7_1}$ em relação a $\mathbf{ESN7_2}$ quanto para $\mathbf{E_{x1}}$ em relação a $\mathbf{E_{x2}}$. Isto é um resultado favorável quando há situações em que é necessário criar vários ESN7. A Configuração C1 não invalida esta conclusão por ser uma configuração inadequada e que não deve ser usada.
É interessante fazer uma comparação do desempenho da máquina 2 nas configurações C2 e C3. Na C2 não há invocação remota. A invocação é feita localmente. Os atrasos devidos à rede estão eliminados porque tanto o cliente quanto o servidor e o registro de nomes estão na mesma máquina. Isto deixa mais claro o desempenho da máquina 2.
A partir do exposto acima é razoável supor que em um caso real, dificilmente poderão ser aproveitadas todas máquinas existentes no parque de uma operadora de telefonia.
Os dados obtidos nos levam a concluir que, para ambas implementações feitas, a 1ª. e a 3ª. máquinas podem ser utilizadas para as funções de servidor e de registro de nomes mas não para a função de cliente.

5.3 Eficiência da Linguagem Java

Na construção da interface gráfica era muito importante definir o que estaria em cada tela, como estaria disposto, em que seqüência de apresentação e que componente Java seria usado em cada caso para a interação com o usuário, de forma a não sobrecarregar a tela, ser completo, amigável, etc. Os recurso gráficos da linguagem Java permitiram muito grau de liberdade e muita flexibilidade nestes aspectos. A linguagem Java se mostrou muito adequada para a construção da interface gráfica. A seqüência de apresentação das telas que foi escolhida para ser usada no protótipo (barra de menus, etiquetas e componentes – caixas de combinações, botões, etc.) é apenas uma entre muitas outras que poderiam ter sido usadas. Pelas telas mostradas anteriormente se pode concluir que Java permitiu uma forma amigável de interação com o usuário final.

Java também permitiu tratar ambas implementações de uma forma isolada dos detalhes dos protocolos e dos detalhes da base de dados da central. Isto é importante para o programador porque permite independência de conhecimentos específicos da central telefônica com a qual a IGU estiver interagindo. Como o uso do protocolo é transparente para o programador e o uso da interface gráfica, após aprendida é relativamente fácil, o código fonte do protótipo é relativamente simples.

A criação da IGU foi muito trabalhosa no início, quando ainda não se sabia como usar os recursos Java.

Quando foi necessário modificar o código fonte para corrigir erros cometidos na concepção inicial da solução a linguagem se mostrou de uma eficiência muito grande. A IGU se restringiu ao uso de objetos gráficos padronizados que podem ser combinados conforme adequado. O fato de os componentes Java irem sendo "ajuntados" facilita muito a construção dos programas e a sua depuração. Pelo fato de as classes estarem contidas em arquivos separados e os componentes serem "ajuntados" na solução, fica mais fácil identificar onde está o problema. Além disto, a quantidade de linhas de código fonte necessárias para ambas implementações do protótipo foi muito pequena graças ao elevado

nível de abstração da linguagem. Isto simplificou a solução e ajudou a ter uma idéia de conjunto mais fácil – o que foi muito importante para depurar os erros.

Para medir a dificuldade de uso da linguagem todos os programas fontes usados foram propositadamente editados com um editor de texto comum do Windows (Notepad). Ocuparam aproximadamente 65 Kb para a implementação em RMI e 59 Kb para a implementação em IDL. Após compilados todas as classes ocuparam aproximadamente 58 Kb para a implementação RMI e aproximadamente 56 Kb para a em IDL. Isto inclui todo o código, comentários, registro histórico de mudanças, recursos para testes usados durante a depuração, etc. Isto permitiu concluir que pouco código fonte foi necessário para construir o protótipo e que o tamanho do código gerado foi pequeno.

Implementações	RMI	IDL
Código Fonte	65 Kb	59 Kb
Código Compilado	58 Kb	56 Kb

Tabela # 5.15 – Tamanhos dos Códigos

O tempo levado para assimilar a linguagem e escolher as opções adequadas foi muito grande. Isto foi devido, entre outros fatores, à variedade de opções que a linguagem Java permite e às dificuldades iniciais para a obtenção da bibliografia e do rumo corretos. Embora a quantidade e a qualidade da documentação existente seja boa, escolher a opção a usar envolveu "gastar tempo" com muitas outras que tiveram que ser entendidas e a seguir não foram usadas.

Capítulo 6 – Conclusões e Sugestões

6.1 Aproveitamento das máquinas já existentes

Os desenvolvimentos propostos foram feitos com interfaces pardronizadas e que permitem o desenvolvimento de aplicações com diferentes níveis de complexidade. O uso de clientes implementados com Java permite executar aplicações amigávies que atendem o perfil dos usuários das operadoras de telefonia. Java também permite o uso de vários tipos de hardware e o uso dos sistemas operacionais predominantes nas operadoras.
O uso de IDL permite a padronização de interfaces independentes da linguagem de programação usada para a implementação nos servidores. Isto é interessante já que os servidores em geral precisam ser implementados com linguagens mais eficientes que Java.
O uso da interface IDL também permite que as aplicações possam ser desenvolvidas em diferentes níveis de complexidade e para diferentes domínios de aplicação sem, no entanto, perderem a compatibilidade entre si e a possibilidade de interação mútua. A complexidade fica restrita ao servidores – que são poucos (em relação à quantidade de clientes) e de uso comum. Isto permite desenvolver aplicações menores nos clientes, dedicadas aos fins específicos de uma certa área de uma operadora, mesmo em máquinas de pequeno porte, tais como as usadas nos protótipos. Ou seja, é já possível usar tal tipo de solução com os recursos já existentes nas operadoras, sem ter que substituir, necessariamete, o microcomputador de trabalho de cada usuário. Assim, por exemplo, é possível desenvolver aplicações separadas, orientadas a soluções específicas, separadas por áreas de atuação e por funcionalidade hierárquica, para a obtenção de dados de desempenho de tráfego, para o levantamento da árvore de encaminhamento da central, para a obtenção de dados de tarifação, para mudar a configuração dos recursos da central telefônica, etc.

6.2 Eficiência da forma de desenvolvimento adotada

Usar UML não foi uma escolha inicial. Considerando a baixa complexidade das implementações, a disponibilidade de somente uma pessoa para todo o desenvolvimento, que a natureza acadêmica do desenvolvimento não iria exigir manutenção posterior, etc., optou-se por não usar UML para favorecer o aprendizado da linguagem Java e viabilizar o desenvolvimento em uma máquina com poucos recursos. A máquina 1 foi usada para isto. Além disto, a própria ansiedade para aprender e para gerar solução rápida e partir logo para gerar código, o desconhecimento inicial de alguns dos vários temas envolvidos e o longo tempo de aprendizado, fez com que as coisas acontecessem de forma inversa: partiu-se de um desenvolvimento sem uso de UML e Rose para se chegar até à importância e à necessidade deles. Percebeu-se que tal forma de condução não apresentou qualquer vantagem: tudo que havia sido feito poderia ter sido feito com melhor qualidade e menor tempo se a plataforma Rose tivesse sido usada desde o início.

6.3 Implementação usando RMI versus usando IDL

Sob o ponto de vista do desempenho dos protocolos usados, através das medições de tempos mostradas nas tabelas dos itens anteriores (configurações C2 e C3) é possível concluir que a

linguagem Java se mostrou adequada, tanto na implementação RMI quando na IDL. Pela configuração C3 percebe-se que a implementação em IDL exigiu mais capacidade da rede e das máquinas (foi mais lenta) que a RMI.

Considerando as ressalvas feitas para a máquina 1, o desempenho de ambas implementações foi adequada para a criação de um ESN7, obedecidas as simplificações inerentes ao protótipo feito. Não se viu grandes diferenças entre ambas implementações, embora RMI tenha exigido menos recursos de máquina, conforme exposto anteriormente.

IDL apresenta a vantagem de não estar restrita à implementação de um fabricante ou a uma linguagem, como o está a RMI. IDL permite construir um cliente em Java e um servidor em C, tirando vantagens do que cada linguagem tenha de melhor. Java também permite usar RMI e JNI, de forma a mapear Java em outras linguagens mas mesmo assim se continuaria usando Java no servidor quando há linguagens mais eficientes para o servidor. O fator padronização tem, no entanto, peso muito maior a favor da opção por IDL. Uma solução intermediária que use RMI entre o cliente e o servidor e IDL (diretamente ou indiretamente) entre o servidor e as base de dados das centrais telefônicas precisaria ser analisada mas não foi objeto de estudo do presente protótipo.

6.4 *Simplificar, Descentralizar e Agilizar a Obtenção de Soluções*

Um dos maiores propósitos deste protótipo é desenvolver algo que possa vir a ser útil sem depender de uma pilha de protocolos, plataformas e softwares de acesso difícil e restrito. Se por um lado o protótipo permite atingir isto, por outro lado é verdade que ele se depara com uma dificuldade muito grande inerente à necessidade de estabelecimento de interfaces padronizadas. É um modelo diferente e que certamente conduz a diferentes opiniões tanto por parte das operadoras quanto por parte dos fornecedores. Pode parecer, em uma 1ª. análise mais favorável às operadoras que aos fornecedores. No entanto, muita flexibilidade pode ser atingida com o uso destas interfaces padronizadas. Isto permitirá atingir uma modularidade em que sempre será possível criar várias soluções criativas e úteis e deixar espaço para muita negociação entre fabricantes e operadoras. Tal solução acrescentará pelo menos alguns níveis a mais de liberdade para a obtenção das soluções. As operadoras não precisam ficar em "*dead lock*" aguardando soluções simples que, pelos mais variados motivos, duram tempos excessivamente longos por parte dos fornecedores e os fabricantes podem sempre estar à frente oferecendo mais um módulo que atenda às necessidades específicas das operadoras. Podem ser criadas soluções mais simples e dedicadas para serem executas em máquinas mais simples ou soluções mais abrangentes para serem executadas em máquinas melhores.
Com pouco treinamento seria possível obter soluções mais rápidas para os problemas do dia a dia de interesse para os vários peritos das mais diversas equipes de comutação telefônica. As ferramentas necessárias não são muitas para a confecção dos programas. No caso deste protótipo bastou um PC e a linguagem Java.

6.5 *Disponibilizar os códigos via "Intranet"*

A idéia é permitir que somente o que estiver sendo usado no momento esteja carregado na máquina hospedeira. Dividir os problemas em várias partes, evitando aplicações excessivamente abrangentes, torna os códigos possíveis de serem executados em máquinas pequenas já existentes na operadora. Os programas podem ser desenvolvidos sob

necessidade, por especialistas que eventualmente podem pertencer ao próprio corpo da operadora. Podem ser comprados prontos ou desenvolvidos, testados e disponibilizados com antecedência em uma página de uma biblioteca de programas de aplicação da rede da empresa a partir da qual podem ser usados. Um controle automático de versões pode ser usado para gerenciar atualizações, melhorias e compatibilidade entre aplicações. Uma quantidade muito grande de aplicações é possível, algumas já citadas antes. É também possível estender a aplicação da solução abordada por este modelo para outras áreas de aplicações como a telefonia celular móvel e a rede inteligente (ver Ref. [75], Ref. [103], Ref. [59], e Ref. [160] a [158]). Com ou sem uma interação com a TMN é possível tratar não só aplicações relativas ao gerenciamento das centrais telefônicas, tal como a melhoria da interface para comunicação homem – máquina (ICHM), como também vários outros pontos.

6.6 *Interfaces Para Interação com as Bases de Dados das Centrais*

A partir do esboço feito nesta dissertação, usando os princípios apresentados na implementação feita em IDL, outros protótipos poderiam ser construídos para testar aplicações que usem as interfaces com bases de dados reais. É um trabalho que tem grandes chances de poder contar com os interesses de operadoras, fabricantes e órgãos de padronização.

Há várias alternativas possíveis para fazer mapeamentos de um protótipo Java nos modelos que atualmente existem para interação com bases de dados de centrais telefônicas. Excluindo IDL, as formas atualmente existentes são essencialmente as seguintes:

1) GDMO/ASN.1,
2) CHM específicos de cada fabricante e
3) outras soluções desenvolvidas para fins específicos.

Já existe hoje um mapeamento de IDL para GDMO/ASN.1. Ver Ref. [119], [120], [118] e [123]. Isto permite interagir com muita implementação já feita em GDMO/ASN.1.

Java fornece recursos para permitir seu mapeamento em outras linguagens, de forma a atender os demais casos que envolvem as soluções particulares ("*wrappers*"). Faz-se um "invólucro" escrito em Java que acesa o código escrito em outra linguagem de programação. Traz algumas vantagens mas exige acesso ao código fonte da implementação herdada.

Dentro deste contexto é útil padronizar uma interface em IDL para a interação com as bases de dados das centrais telefônicas. A proposta feita nesta dissertação poderia ser usada como ponto de partida para tal interface.

Os clientes para as várias aplicações descritas no capítulo 1 podem ser feitos a partir do protótipo apresentado nesta dissertação, em geral sem muito esforço. Alguns casos (tal como a obtenção de dados de tarifação) exigirão cuidados especiais de segurança. Se tais clientes contassem com o respaldo de interfaces padronizadas já existentes nos servidores, o acesso à informação das BD das centrais seria relativamente fácil.

Usar interfaces padronizadas permite a separação e interação de áreas de atuação de uma forma organizada conforme citado no capítulo 1. O uso conjunto de IDL e Java JNI permite aproveitar soluções herdadas de produtos de diferentes fabricantes mas tem a desvantagem

de ter que ser usada, no servidor, a linguagem Java (ao invés de outra de melhor desempenho).

Ao contrário de ser um modelo a mais, este vem para possibilitar a integração de diferentes modelos já existentes.

A evolução futura está garantida pelo uso da interface IDL. O uso isolado de RMI não poderia garantir esta evolução tão amplamente.

Aproveitar equipamentos e softwares já existentes é de grande valia para as empresas operadoras porque, dentre vários outros motivos, pode permitir que o novo modelo de integração de soluções seja usado nos próprios equipamentos que são objetos de integração.

Os recursos usados foram bem mais simples e bem mais acessíveis que as plataformas de desenvolvimento que a solução TMN do ITU-T exige.

Serviços funcionalmente semelhantes aos do CMIS na TMN do ITU-T podem ser providos através de padronização das interfaces e sem qualquer dependência de uma pilha de protocolos cara e de uma MIB de difícil acesso, tal como a solução TMN exige.

Os parâmetros associados às interfaces a serem padronizadas podem ser detalhados com base nos parâmetros usados no CMIS. A título de exemplo, usando os conceitos de escopo, base e filtro, pode ser definida uma interface para permitir enviar comandos mais poderosos, úteis e genéricos a vários servidores, simultaneamente, a partir de um mesmo cliente. Assim seria possível, para ilustrar, a partir de uma única solicitação do operador, listar todos os ESN7 com uma mesma característica em uma certa região. Ex.: listar todos ESN7 fora de funcionamento entre uma certa central e as demais centrais de uma certa cidade.

Somente uma padronização criteriosa e completa das interfaces IDL permitirão evoluir para a construção de aplicações mais completas que possam atender, de forma equivalente, à arquitetura funcional da TMN tal qual visto no capítulo 2. São necessárias diferentes interfaces para cobrir os diferentes níveis de gerenciamento expostos. É evidente que diferentes complexidades de software e de hardware são necessários para atender tais tipos de soluções e para atender o que foi apresentado pelo presente protótipo. Conforme já citado anteriormente, no capítulo 2, note-se que o protótipo deste trabalho se enquadra funcionalmente, por analogia com a TMN, no 2° nível do modelo de referência da arquitetura funcional de OS da TMN, isto é, na camada de gerenciamento de elemento de rede. No entanto, há espaço para muito trabalho em todos os níveis de tal arquitetura.

As referências do modelo de informação da TMN dão uma idéia do que precisaria ser atendido a nível de interfaces IDL. Como muitos deste requisitos já estão atendidos, de forma alternativa, pelas padronizações da OMG, isto fortalece a adoção das interfaces IDL ao invés de RMI.

As operadoras de telefonia querem produtos padronizados no mercado para comprarem. Por isto buscam na TMN do ITU-T a solução ideal. No entanto, até certo ponto isto é uma ilusão. Colocam suas expectativas nos produtos que seguem a TMN do ITU-T e se deparam com produtos caros, muitas vezes somente flexíveis dentro do escopo de um mesmo fabricante, cujas aplicações estão restritas a serem executadas somente em plataformas caras

e raras. Para qualquer problema que saia daquilo que já compraram dificilmente têm solução rápida porque em geral as MIBs são proprietárias.

O novo modelo proposto permite agilizar soluções, viabilizar desenvolvimentos tanto pelos fabricantes, quanto pelas operadoras, como também por outras casas de desenvolvimento de software. Permite que a necessidade, a importância, a complexidade e a urgência definam a forma de prover a solução. Permite também usar máquinas de baixa capacidade de processamento. Conforme já dito anteriormente, a idéia básica do novo modelo é colocar a complexidade na definição da interface e na implementação do servidor e deixar o cliente simples o máximo para ser flexível e de rápida implementação. O único ponto para evitar o caos é exatamente o gargalo da proposta: carece da padronização das interfaces IDL a serem usadas. Para isto a sugestão feita para a interface ESN7 precisa ser expandida e melhorada para atender outros aspectos também ligados aos ESN7 e para contemplar outras áreas.

Abreviaturas e Acrônimos

ACSE	*"Association Control Service Element"* – Elemento de Serviço de Controle de Associação
ADSL	*Asymmetrical Digital Subscriber Line* – Linha de assinante digital assimétrica
ANATEL	*Agência Nacional de Telecomunicações*
ANSI	*"American National Standards Institute"* – Instituto de Padrões Nacionais Americanos
API	*"Application Program Interface"* – Interface de Programa de Aplicação
ASE	*Application Service Element*" – elemento de serviço da aplicação
ASN.1	*"Abstract Syntax Notation One"* – Notação de Sintaxe Abstrata 1
ATMN	Agente TMN (interno à central telefônica)
AVA	"*Attribute Value Assertion*" – "atribuição de valor a um atributo"
AWT	"*Abstract Window Toolkit*" – caixa de ferramentas de janelas abstratas
BD	Base de Dados
BER	*"Basic Encoding Rules for ASN.1"* – regras de codificação básica para a ASN.1
BOA	"*Basic Object Adapter*" – adaptador básico de objeto
CHM	Comunicação Homem – Máquina OU Comandos Homem-Máquina
CIC	"*Circuit Identification code*" – Código de identificação de circuito
CMIP	*"Common Management Information Protocol"* – Protocolo de informação comum de gerenciamento
CMISE	*"Common Management Information Service Element"* – Elemento de serviço de informação comum de gerenciamento
COM	*"Component Object Model*" – modelo de objetos componentes
CORBA	"*Common Object Request Broker Architecture*" – Arquitetura Comum de "Agenciador" para Solicitação de Objeto
COS	"*CORBA Services*" – objetos de serviços CORBA
CPCT	Central Privada de Comutação Telefônica
CPS	Código do Ponto de Sinalização
DAF	"*Directory Access Function*" - Função de Acesso de Diretório
DCF	*"Data Communication Function"*- Função de comunicação de dados da TMN
DCN	"*Data Communication Network*" – rede de comunicação de dados
DPC	*"Destination Point Code*" – Código de Ponto de Destino
DUP	*"Data User Part"* – Parte de Usuário de Dados
DSF	*"Directory System Function"* – Função de Diretório de Sistema
EMBRATEL	*Empresa Brasileira de Telecomunicações*
ESIOP	"*Environment-Specific Inter-ORB Protocol*" - protocolo entre ORB específico do ambiente
ESN7	Enlace de Sinalização Número 7
FDN, DN	"*Fully (Distinguishing) Name*" – nome (de distinção) completo
FTAM	*"File Transfer Access and Management Protocol"*- Protocolo transferência, acesso e gerenciamento de arquivos
FTP	*"File Transfer Protocol"*– Protocolo de transferência de arquivo
GDMO	*"Guidelines for the Definition of Managed Objects"* – Diretrizes para a

	Definição de Objetos Gerenciados
GIOP	*"General Inter-ORB Protocol"* – protocolo geral entre ORB
GRT	*"Global Registration Tree – GRT"*- árvore de registro global
GTMN	Gerente TMN
GTT	*"Global Title Translation"* – Tradução de Título Global
GUI	*"Graphical User Interface"* – interface gráfica de usuário
HTML	*"Hypertext Markup Language"*
IBD	Interfuncionamento com Base de Dados
IC	Interfuncionamento com o Cliente
ICC	Interfuncionamento com os comandos de comunicação homem-máquina da central (CHM)
ICF	*Information Conversion Function*"- Função de Conversão da Aplicação
ICHM	Interface para Comunicação Homem – Máquina
ICSH	Interfuncionamento da Central com o Sistema Herdado de interação com a central
ICT	Interfuncionamento com a Central Telefônica
IDL	*"Interface Definition Language"* – linguagem de definição de interfaces
IGU	interface gráfica de usuário
IIOP	*"Internet Inter-ORB Protocol"* – Protocolo entre ORB para Internet
IP	*"Internet Protocol"* – protocolo da Internet
IS	Interfuncionamento com o Servidor
ISO	*"International Organization for Standardization"* – Organização Internacional para Padronizações
ISDN	*"Integrated Service Digital Network (ISDN)"* – Rede Digital de Serviços Integrados (RDSI)
ISSH	Interfuncionamento do Servidor com o Sistema Herdado de interação com a central
ISUP	*"Integrated Service Digital Network (ISDN) User Part"* – Parte de Usuário de Rede Digital de Serviços Integrados (RDSI)
ITU-T	União Internacional de Telecomunicação – Setor de Padronização de Telecomunicações *("International Telecommunication Union")*
JFC	*"Java Foundation Class"* – classes fundamentais Java
JNI	*"Java Native Interface"* – Interface Java para códigos "nativos"
MAF	*"Management Application Function"* – Função de Gerenciamento da Aplicação
MD	*"Mediation Device"* – dispositivo de mediação
MF	*"Mediation Function"* – função de mediação
MFC	Multi-Frequencial Compelido
MIB	*Management Information Base"* - Base de informação de gerenciamento
MIT	*"Management Information Tree"*- árvore de gerenciamento de informação
MSF	*"Message Communication Function"* – Função de Comunicação de Mensagem
MTP	*"(Message Transfer Part)"*- Parte de Transferência de Mensagem
N7	(Sinalização) Número 7
NE	*Network Element"* – elemento de rede
NEF	*"Network Element Function"* – função de gerenciamento da rede
NSAP	*"Network Service Access Point"*- ponto de acesso ao serviço de rede

OA	*"Object Adapter"* – adaptadores de objeto
OID	*"Object Identifier"*- identificador de objeto
OMG	*"Object Management Group"* – Grupo de Gerenciamento de Objeto
OPC	*"Origination Point Code"* – Código de Ponto de Origem
ORB	*"Object Request Broker"* – "Agenciador" para Solicitação de Objeto
OS	*"Operations System"* – sistema de operações
OSF	*"Operations Systems Function"* – Função de Sistemas de Operações
OSI	*"Open System Interconnection"* – Interconexão de Sistema Aberto
PABX	*"Public Automatic Branch Exchange"* – Central Privada de Comutação Automática (ver também CPCT)
PCM	Modulação por pulsos codificados (*"Pulse Code Modulation"*)
PS	Ponto de Sinalização
POA	*"Portable Object Adapter"* – adaptador portátil de objeto
POSIX	*"Portable Operating System Interface for Computer Environment"* – Interface de Sistema Operacional "Portátil" para Ambiente de Computador
POTS	*"Public Old Telephone System"* – Sistema de Telefones Convencionais
PTS	Ponto de Transferência de Sinalização
QA	*"Q Adaptor"* – adaptador Q
QAF	*"Q Adaptor Function"* – função de adaptação para a interface Q
RDSI	Rede Digital de Serviços Integrados (o mesmo que ISDN)
RDN	*"(instance's) relative distinguishing name"* – nome de distinção relativa (da instância)
RI	Rede Inteligente
RMI	Invocação Remota de Método
ROSE	*"Remote Operations Service Element"* – Elemento de Serviço de Operações Remotas(Obs.: Note-se que é distinto de **"Plataforma Rose")**
RSN7	Rota de Sinalização Número 7
SCCP	*"Signaling Connection Control Part"* – Parte De Controle da Conexão de Sinalização
SF	*"Security Function"* – Função de Segurança
SLC	*"Signaling Link Code"* – Código do Enlace de Sinalização
SN7	Sinalização Número 7
SNMP	*"System Network Management Protocol"* – protocolo de gerenciamento de rede de sistema
SSN	*"Sub System Number"* – Número de subsistema
SP, SPC	*"Signaling Point (Code)"* – (código) do ponto de sinalização
SQL	*"System Query Language"* – Linguagem de Consulta de Sistema
STP	*"Signaling Transfer Point"* - Ponto de Transferência de Sinalização
TCAP	*"Transaction Capability Application Part"* - Parte da Aplicação de Capacidade de Transação
TCP/IP	*"Transmission Control Protocol / Internet Protocol"* - Protocolo de Controle de Transmissão / Protocolo da Internet
TMN	*"Telecommunication Management Network"* - Rede de Gerenciamento de Telecomunicação
TP	Telefone Público
TUP	*"Telephonic User Part"* - Parte de Usuário Telefônico
UISF	*"User Interface Support Function"*, Função de Suporte de Interface de

	Usuário
URL	"*Uniform Resource Locator*" - Localizador uniforme de recurso
VOCODER	"*Voice Coder / Decoder*" Codificador e decodificador de Voz
WS	"*Workstation*" - estação de trabalho
WSF	"*Work Station Function*" - função de estação de trabalho
WSSF	"*Workstation Support Function*", Função de Suporte de Estação de Trabalho
XOM	"*X/Open Abstract Data Manipulation*" - manuseio de dados abstratos X/Open
XMP	"*X/Open Management Protocol*" – protocolo de gerenciamento X/Open

Bibliografia

[1] ACL - Advanced Computing Laboratory- http://www.acl.lanl.gov

[2] Alcântara, Andreia de Almeida & outros - Introdução a Java –Anexo V Escola Regional de Informática da SBC Regional Sul - Universidade Federal de Pernambuco - 5 a 10 de Maio de 1997

[3] Appeldorn, Menso e outros - TMN + IN = TINA –IEEE Communications Magazine - Mars/93

[4] Bandes, Kenneth - Java and COM Automation – Dr. Dobb's Journal, January 1998

[5] Barr, Willian J. e outros - The TINA Initiative – IEEE Communications Magazine - Mars/93

[6] Campione, Mary and Walrath, Kathy - The Java Tutorial - "Object" - Oriented Programming for the Internet (Java-Series) - Disponível em http://java.sun.com/docs/books/tutorial/

[7] Caruso, Raymond and Kean, Mark - Power Programming in HP Open View – Developing CMIS Applications –Hewllet Packard Professional Books - Prentice Hall PTR - Disponível em http://www.prenhall.com

[8] Carvalho, Tereza Cristina Melo de Brito - Arquitetura de Redes de Computadores OSI e TCP/IP - EMBRATEL - BRISA - SGA - Makron Books.

[9] Case, J.; Fedor, M.; Schoffstall, M. and Davin, J. – Internet Activities Board (IAB) - RFC 1157 - A Simple Network Management Protocol (SNMP) - May/1990.

[10] CCITT – E.410 - 10/92 - Quality of Service, Network Management and Traffic Engineering - International Network Management - General Information

[11] CCITT – E.411 - 10/92 - Quality of Service, Network Management and Traffic Engineering - International Network Management - Operational Guidance

[12] CCITT – E.412 - 10/92 - Quality of Service, Network Management and Traffic Engineering - Network Management Controls

[13] CCITT – E.413 - 1993 - Quality of Service, Network Management and Traffic Engineering - International Network Management - Planning

[14] CCITT - E.414 - 1993 - Quality of Service, Network Management and Traffic Engineering - International Network Management - Organisation

[15] CCITT - E.415 - 1991 - Quality of Service, Network Management and Traffic Engineering - International Network Management - Guidance for Common Channel Signalling System No.7

[16] CCITT - E.500 (rev.1) 1992 - Quality of Service, Network Management and Traffic Engineering - Traffic Intensity Measurements Principles

[17] CCITT - M.3100 - 10/92 - Telecommunications Management Network - Generic Network Information Model

[18] CCITT - M3180 - 10/92 - Maintenance – Telecommunications Management Network - Catalogue of TMN Management Information

[19] CCITT - M3200 - 10/92 - Maintenance – Telecommunications Management Network - TMN Management Services: Overview

[20] CCITT - M3300 - 10/92 - Maintenance – Telecommunications Management Network - TMN Management Capabilities Presented at the F Interface

[21] CCITT - M3400 - 10/92 - Maintenance – Telecommunications Management

Network - TMN Management Function
[22] CCITT - X.200 – Reference Model for Open Systems Interconnections for CCITT Applications - 1988
[23] CCITT - X.208 – Specification of Abstract Syntax Notation One (ASN.1), 1988
[24] CCITT - X.209 – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), 1988
[25] CCITT - X.210 - Open Systems Interconnection (OSI) Layer Service Conventions - 1988
[26] CCITT - X.215 – Session Service Definition for Open Systems Interconnections for CCITT Applications - 1988
[27] CCITT - X.216 – Presentation Service Definition for Open Systems Interconnections for CCITT Applications - 1988
[28] CCITT - X.217 – Association Control Service Definition for Open Systems Interconnections for CCITT Applications – 1988
[29] CCITT - X.219 – Remote Operations: Model, Notation and Service Definition – 1988
[30] CCITT - X.226 – Presentation Protocol Specification for Open Systems Interconnections for CCITT Applications – 1988
[31] CCITT - X.227 – Association Control Protocol Specification for Open Systems Interconnections for CCITT Applications – 1988
[32] CCITT - X.229 – Remote Operations: Protocol Specification – 1988
[33] CCITT - X.700 - 09/92 - Management Framework for Open Systems Interconnection (OSI) for CCITT Applications.
[34] CCITT - X.701 – 1992 - OSI - Systems Management Overview
[35] CCITT - X.710 – 1991 - Common Management Information Service for CCITT Applications.
[36] CCITT - X.711 – 1991 - Common Management Information Protocol Specifications for CCITT Applications.
[37] CCITT - X.712 - 09/92 - OSI - Common Management Information Protocol : Protocol Implementation Conformance Statement Proforma.
[38] CCITT - X.720 - 01/92 - OSI - Structure of Management Information: Management Information Model
[39] CCITT - X.721 - 1992 - OSI - Structure of Management Information: Definition of Management Information
[40] CCITT - X.722 - 01/92 - OSI - Structure of Management Information: Guidelines for the Definition of Managed Objects.
[41] CCITT - X.730 - 01/92 - OSI - Object Management Function
[42] CCITT - X.731 - 01/92 - OSI - State Management Function
[43] CCITT - X.732 - 01/92 - OSI - Attributes for Representing Relationships
[44] CCITT - X.733 - 1992 - OSI - Alarm Reporting Function
[45] CCITT - X.734 - 1992 - OSI - Event Report Management Function
[46] CCITT - X.735 - 09/92 - OSI - Log Control Function
[47] CCITT - X.736 - 1992 - OSI - Security Alarm Reporting Function
[48] CCITT - X.740 - 09/92 - OSI - Security Audit Trail Function
[49] CERFNET - http://www.cerfnet.com/~mpcline/CORBA-FAQ/
[50] Chaffee, Alex and Martin, Bruce - Introduction to CORBA - JGuru Training by the Magelang Institute – Disponível em http://developer.java.sun.com/
[51] Computer Science Dept – University of Wisconsin – Madison – Ghostscript, Ghostview and GSview -http://www.cs.wisc.edu/~ghost/

[52] Conchon, A. e outros - Does CORBA Fit With TMN ? – XVI World Telecom Congress Proceedings

[53] Cruz, Elaine Álvares e outros - Aspectos de Modelo de Informações no Contexto da GIRS para Gerência de Centrais CPA-T – Revista TELEBRÁS - Maio/97

[54] Dupuy, Fabrice e outros - The TINA Consortium: Toward Networking Telecommunications Information Services – IEEE Communications Magazine - November/95

[55] Eckel, Bruce - http://www.BruceEckel.com

[56] Eckel, Bruce - Thinking in Java –Printice Hall - 1998 - Disponível em http://www.BruceEckel.com

[57] ETSI I-ETS 300 293 – TMN Generic Managed "Object" s – 1996

[58] ETSI I-ETS 300 653 – TMN Generic Managed "Object" s Class Library for Network Level View - 1996

[59] Expersoft - Bridging the Gap Between CORBA and COM – Disponível em www.expersoft.com/Products/CORBAActiveX/white.htm

[60] Expersoft - CORBAplus, ActiveX Bridge - Disponível em www.expersoft.com/Products/CORBAActiveX/toc.htm

[61] Expersoft - http://www.expersoft.com

[62] Expersoft - Tutorial on Distributed Objects - Disponível em www.expersoft.com/Resources/DistTech/tutorial.htm

[63] Expersoft - Wrapping Legacy Applications: A Distributed Computing Solution - Disponível em www.expersoft.com/Resources/Wrapers/legacy.htm

[64] Fayan, Benedito Luís e outros - Análise da Utilização das API XOM/XMP no Desenvolvimento de Gerência para a Rede de Telecomunicações e Computadores – Revista TELEBRÁS – Maio/97

[65] Feridun, M. e outros - Implementing OSI Agents/Managers for TMN - IEEE Communications Magazine – September/96

[66] Finkelstein, Rich - The New Middleware – DBMS 02/95

[67] Ford, Dan - Mutual Exclusion and Synchronisation in Java – Dr. Dobb's Journal, January 1998

[68] Fowler, Martin and Scott, Kendall - UML Distilled - Applying The Standard Object Modeling Language -Addison-Wesley - 1998.

[69] Fraizer, Colin and Bond, Jill - API Java - Manual de Referência - Makron Books - 1997

[70] Glitho, Roch H. and Hayes, Stephen - Approaches for Introducing TMN in Legacy Networks: A Critical Look - IEEE Communications Magazine - September/96

[71] Glitho, Roch H. and Hayes, Stephen - Making TMN a Reality - IEEE Communications Magazine - September/96

[72] Gokhale, A.; Schmidt, D. and Mayer, S. - Software Tools for Automating the Migration from DCE to CORBA - XVI World Telecom Congress Proceedings

[73] Halsall, Fred - Data Communications, Computer Networks and OSI - Addison - Wesley.

[74] Hart, T. J. - Questioning CORBA – DBMS Mars/97

[75] Hersog, U. and Magedanz, T. - Intelligent Networks and TINA - Migration and Interworking Issues - XVI World Telecom Congress Proceedings

[76] Hughes, Cameron; Hamilton, Thomas and Hughes, Tracey - Object-Oriented I/O Using C++ IOSTREAMS - John Wiley & Sons, Inc.

[77] IAB - Internet Activities Board - RFC 1189 – The Common Management

Information Services and Protocol for the Internet - 10/26/1990. (Nota: esta RFC substitui a RFC 1095 - CMOT - Common Management Information Services and Protocol over TCP/IP - 4/1/1989)

[78] Iona - http://www.iona.com/

[79] ITU-T - M.3000 - 10/94 - Overview of TMN Recommendations

[80] ITU-T - M.3010 - 05/96 - Principles for a Telecommunications Management Network

[81] ITU-T - M.3020 - 07/95 - Maintenance – Telecommunications Management Network - TMN Interface Specification Methodology

[82] ITU-T - Q.542 - 03/93 - Digital Exchange Design "Object" ives – Operations and Maintenance

[83] ITU-T - Q.811, Lower Layer Protocol Profiles for the Q3 Interface.

[84] ITU-T - Q.812, Upper Layer Protocol Profiles for the Q3 Interface.

[85] ITU-T - Q.822 - 04/94 - Specifications of Signalling System No. 7 - Stage 1, 2 and 3 Description for the Q3 Interface – Performance Management

[86] ITU-T - Q.823 - 07/96 - Specifications of Signalling System No. 7 - Q3 Interface - Stage 2 and 3 Functional Specification for Traffic Management

[87] ITU-T - X.500 (1993), ISO/IEC9594-1:1995 - The Directory: Overviews of concepts, models and services.

[88] ITU-T - X.739 - 11/93 - OSI - System Management: Metric "Object" s and Attributes

[89] ITU-T - X902 - 11/95 - Open Distributed Processing – Reference Model: Foundations

[90] ITU-T - X903 - 11/95 - Open Distributed Processing – Reference Model: Architecture

[91] ITU-T Q.700 - Introduction to CCITT Signalling System no. 7 - 03/93.

[92] ITU-T Q.701 - Functional Description of the Message Transfer Part (MTP) of the Signalling System no. 7 - 03/93.

[93] ITU-T Q.702 - Specification of Signalling System no. 7 – Signalling Data Link - 03/93.

[94] ITU-T Q.703 - Specification of Signalling System no. 7 – Message Transfer Part - Signalling Link - 07/96.

[95] ITU-T Q.704 - Specification of Signalling System no. 7 – Message Transfer Part - Signalling Network Functions and Messages – 07/96.

[96] ITU-T Q.705 - Specification of Signalling System no. 7 – Signalling Network Structure - 03/93.

[97] ITU-T Q.708 - Numbering of International Signalling Point Codes - 03/93.

[98] ITU-T Q.751.1 - Specification of Signalling System no. 7 – Network Element Management Information Model for Message Transfer Part (MTP)- 10/95.

[99] ITU-T Q.752 - Specification of Signalling System no. 7 – Signalling System no. 7 Management - Monitoring and Measurements for Signalling System No. 7 Networks - 06/97.

[100] Javasoft - http://www.javasoft.com/products/jdk/idl/refs.html

[101] Jones, Carol - The Java Internationalisation API – Dr. Dobb's Journal, January 1998

[102] Keuffel, Warren - CORBA – DBMS Mars/97

[103] Lengdell, Magnus e outros - The TINA Network Resource Model – IEEE Communications Magazine – Mars/96

[104] Linthicum, David S. - OLE-Enabled Middleware – DBMS January/97

[105] Linthicum, David S. - Revaluating Distributed "Object" s – DBMS January/97

[106] Maffeis, Silvano and Schimidt, Douglas C. - Constructing Reliable Distributed Communications Systems with CORBA – IEEE Communications Magazine - Febrary/97

[107] MageLang Institute - Fundamental of Swing: Part I - Short Course by MageLang Institute - Disponível em http:// developer.java.sun.com /developer/onlineTraining/GUI/Swing1/shortcourse.html

[108] MageLang Institute – Introduction to CORBA - Short Course - Disponível em http://developer.java.sun.com/

[109] Manley, Adrian and Thomas, Clare - Evolution of TMN Network "Object" Models for Broadband Management - IEEE Communications Magazine - 10/97

[110] Melita – http://www.melita.com/

[111] North, Ken - Database Programming with OLE and ActiveX – DBMS 11/96

[112] North, Ken - Understanding OLE – DBMS 06/95

[113] OM&S Telecom - Seminário "TMN para Gerentes" - 22 a 24/4/98

[114] OMG - CORBA Common Facilities Architecture - Ver. 4.0 – Nov 95

[115] OMG - CORBA services: Common "Object" Services Specification - Nov 1997

[116] OMG - CORBA V2.2 - The Common "Object" Request Broker Architecture and Specification - Fev/1998 – Disponível em http://www.omg.org/corba/corbiiop.htm

[117] OMG - Interworking between CORBA and Intelligent Network Systems - Request for Proposal - OMG Document: telecom/97-12-06 - 6/jun/98

[118] OMG - Interworking between CORBA and TMN Systems – Request for Proposal - OMG Document: telecom/97-09-04 - 12/fev/98

[119] OMG - Interworking between CORBA and TMN Systems Ver 1.0 - 25/fev/98 - Response to RFP 97-09-04 (DSET, Expersoft, Iona, SUN, TCSI)

[120] OMG - JIDM Interaction Translation - Initial Submission to OMG's CORBA/TMN Interworking RFP - Ed. 4.0 - fev/98 - (Telefónica, Nortel, Lucent, HP, Alcatel, IBM, ISR, Highlander Comm., Visigenic)

[121] OMG - Joint Submission - Mobile Agent System Interoperability Facilities Specification - GMD FOKUS - 10/nov/97 - OMG TC Document orbos/97-10-05

[122] OMG - OMG Unified Modeling Language Specification – Version 1.3, June 1999 - Disponível em http://www.omg.org

[123] OMG - Telecom Domain "Task" Force - Draft: Request for Information - OMG DTC Document: telecom/96-12-02

[124] Opengroup - http://www.camb.opengroup.org

[125] Orfali, Robert and Harkey, Dan - Client/Server Programming with Java and CORBA - John Wiley & Sons, Inc. - 1997

[126] OSF - The OSF Distributed Computing Environment Building on International Standards – 1992

[127] Otte, Randy; Patrick, Paul and Roy, Mark -Understanding CORBA - The Common "Object" Request Broker Architecture - Prentice Hall PTR - 1996

[128] Pavlou, George and Tin, Thurain. - A CMIS-Capable Scripting Language and Associated Lightweight Protocol for TMN Applications - IEEE Communications Magazine - September/96

[129] Payne, Jeffery E.; Schatz, M.A. and Schmid M. N. - Implemmenting Assertions for Java - Dr. Dobb's Journal, January 1998

[130] Piscitello, David M. and Chapin, A. Lyman - Open systems Networking - TCP/IP and OSI – Addison - Wesley Professional Computing Series.L

[131] Poo, Gee Swee and Chew, Chye-Guan - Modelling of the XOM/XMP API - IEEE Communications Magazine - August/96

[132] Raoy, Mark and Ewald, Alan - Inside DCOM - DBMS April/97

[133] Rational - CD-ROM - Inside the Unified Modeling Language (UML) - 1999 - Disponível em http://www.rational.com/

[134] Resende, Marcos Cabral - Faça Sua Home Page - Ediouro – 1999.

[135] Rose, M. and McCloghrie, K. - Internet Activities Board (IAB) - RFC 1155 - Structure and Identification of Management Information for TCP/IP-based Internets - May/1990.

[136] Schmidt, Claus and Sevcik, Max - Do-It-Yourself TMN Applications by Visual Programming Methods - IEEE Communications Magazine - November/95

[137] Schmidt, Douglas C. - Distributed "Object" Computing – IEEE Communications Magazine - Febrary/97

[138] Schmidt, Douglas C. – Tutorials on CORBA, Design Patterns and OO Communication Framework - Dept of Science – Washington University – May/2000 - http://www.cs.wustl.edu/~schmidt/

[139] Soares, Luiz Fernando Gomes; Lemos, Guido; Colcher, Sérgio - Redes de Computadores - Das LANs, MANs e WANs às Redes ATM - EMBRATEL - Editora Campus.

[140] Sortica, Eduardo - Redes de Telecomunicações, TMN e Gerência Integrada de Redes e Serviços – ^{3}it - Ilha Instituto Integrado de Tecnologia - 1ª. edição – Limitada – Maio/1999

[141] Stallings, Willian - Networking Standards - A guide to OSI, ISDN, LAN and MAN Standards - Addison-Wesley Publishing Company.

[142] SUN - http://developer.java.sun.com/developer/earlyAccess/

[143] SUN - http://java.sun.com/j2se/1.3/docs/index.html

[144] SUN - http://java.sun.com/j2se/1.3/docs/tooldocs/tools.html

[145] SUN - http://java.sun.com/products/

[146] SUN - http://java.sun.com/products/jdk/1.3/docs/index.html

[147] SUN - Java Remote Method Invocation Specification - Revision 1.50 – JDK 1.2 - Sun Microsystems Inc. - Oct/98 – Disponível em http://www.java.sun.com

[148] SUN – JDK1.3 API Doc. - http://www.java.sun.com/products/jdk/1.3/docs/api/index.html

[149] SUN – Understanding Containers - July/1998 - Disponível em http://java.sun.com/products/jfc/tsc

[150] Synantec - http://www.symantec.com/product/

[151] Tadashi, Carlos; Sortica, Eduardo Almansa e Faber, Milton Bem-Hur – Plataformas TMN Comerciais – Revista TELEBRÁS – Maio/97

[152] Tanenbaum, Andrew S. - Computer Networks – Prentice Hill Int. Inc.

[153] TELEBRÁS - BRISA – Gerenciamento de Redes - Uma Abordagem de Sistemas Abertos - Makron Books.

[154] The Agent Society Organization - http://www.agent.org/

[155] Thomas, L.; Suchter, S. and Rifkin, A. - Developing Peer-to-Peer Applications for the Internet – Dr. Dobb's Journal, January 1998

[156] TINAC - http://www.tinac.com/

[157] TINAC - TINA "Object" Definition Language Manual Version 2.3 – 1996

[158] TINAC - TINAC Engineering Modelling Concepts (DPE Architecture) Version 2.0 - 1994

[159] TINAC - TINA-C Network Resource Architecture Version 3.0 – 1997

[160] TINAC - TINA-C Service Architecture Version 5.0 – 1997
[161] Tremblett, Paul - Java Reflection – Dr. Dobb's Journal, January 1998
[162] UML World - http://www.umlworld.com/
[163] van der Linen, Peter – Just Java - Makron Books - 1997
[164] van Hoff, Arthur - Java: Getting Down to Business - Dr. Dobb's Journal, January 1998
[165] van Hoff, Arthur & outros - Ligado em Java - Addison Wesley – 1996
[166] Vasconcellos, Francisco J.S. e Madeira, Edmundo R.M. - Projeto e Desenvolvimento de um Suporte a Agentes Móveis baseado em CORBA.
[167] Vinoski, Steve – CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments – IEEE Communications Magazine - Febrary/97
[168] Visigenic - http://www.visigenic.com/
[169] Visigenic - Visibroker for Java - Gatekeeper Guide - Version 3.0 – 1998
[170] Visigenic - Visibroker for Java - Programmer's Guide - Version 3.0 – 1998
[171] Vogel, Andreas and Duddy, Keith - J ava Programming with CORBA - Wiley Computer Publishing - 1997
[172] Waters, M., Bergström, D. – The Convergence of Management and Control Using CORBA – XVI World Telecom Congress Proceedings
[173] X/Open - Inter-domain Management: Specification Translation – Preliminary Specification - 1997 - X/Open Document Number: P509 - Disponível em http://www.opengroup.org/onlinepubs/8349099/toc.htm
[174] Yang, Zhonghua and Duddy, Keith – CORBA: A Platform for Distributed "Object" Computing - Disponível em www.omg.org/news/begin.htm
[175] Yourst, Matt T. - Inside Java Class Files - Dr. Dobb's Journal, January 1998