UNIVERSIDADE FEDERAL DE SANTA CATARINA

Curso de Pós-Graduação em Ciência da Computação

Gerenciamento de Serviços de Telecomunicações com CORBA e JAVA

Paulo Roberto Riccioni Gonçalves

Florianópolis,08 de setembro de 1998

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Curso de Pós-Graduação em Ciência da Computação

Gerenciamento de Serviços de Telecomunicações com CORBA e JAVA

Dissertação Submetida à Universidade Federal de Santa Catarina para Obtenção do Grau de Mestre em Ciência da Computação

Paulo Roberto Riccioni Gonçalves

Orientador: Prof. Dr. João Bosco Mangueira Sobral

Florianópolis, 08 de setembro de 1998

GERENCIAMENTO DE SERVIÇOS DE TELECOMUNICAÇÕES COM CORBA E JAVA

PAULO ROBERTO RICCIONI GONÇALVES

ESTA DISSERTAÇÃO FOI JULGADA ADEQUADA PARA A OBTENÇÃO DO TÍTULO
DE

MESTRE EM CIÊNCIA DA COMPUTAÇÃO

ESPECIALIDADE SISTEMAS DE COMPUTAÇÃO E APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Prof. Jorge Muniz Barreto, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. João Bosco Mangueira Sobral, Dr.
Orientador

Prof. Nelson L. S. Fonseca, Ph. D.

Prof. Carlos Becker Westphall, Dr.

Prof. Luis Fernando Friedrich, Dr.

"Bom mesmo é ir à luta com determinação,
abraçar a vida e viver com paixão,
perder com classe e vencer com ousadia,
pois o triunfo pertence a quem mais se atreve.
E a Vida é muito para ser insignificante."

Charles Chaplin

Agradecimentos

A minha amada esposa Yara Queiroz Gonçalves e a meus queridos filhos Ig Henrique, Bruno e Rafael. A meus pais Paulo Gonçalves Francisco e Helena Riccioni Gonçalves, pois sem eles não teria alcançado este mérito.

Agradecimentos

Agradeço mui especialmente ao meu amigo, Prof. Dr. João Bosco Mangueira Sobral pela confiança, apoio, estímulo e dedicação.

Agradecimentos

A Aloizio Francisco Gonçalves, por ter sido o meu segundo pai, dando-me o apoio necessário, para que eu terminasse a minha graduação, e estimulando-me a prosseguir em minha carreira.

A Ignácio Queiroz, meu querido sogro, o qual com paciência e dedicação corrigiu os erros de língua portuquesa cometidos em minha dissertação.

Ao amigo e irmão Dr. Salomão Antonio Ribas Junior, a quem mui estimo e admiro, dando-me o apoio necessário junto ao Tribunal de Contas.

Ao Prof. Dr. Carlos Becker Westphall, pelo auxílio dado e pelas discussões que tiveram influência neste trabalho.

A amiga Tatiana Custódio, que auxíliou-me diretamente para a conclusão deste trabalho.

A todos os meus amigos de baia...

A todos que me ajudaram...

A Deus.

SUMÁRIO

ÍNDICE DE FIGURAS

LISTAS DE ABREVIAÇÕES

ANSA	*Advanced Network System Architecture*
API	Interface de Programação de Aplicação
AOO	Análise Orientada a Objetos .
CORBA	*Commom Object Request Broker Archicteture*
COSS	*Common Object Services Specifications*
CCITT	*Consultative Committee for International Telegraph and Telephone*
CMIS	*Common Management Information Service*
CMIP	*Common Management Information Protocol*
DCE	*Distributed Computing Environment*
DCN	Rede de Comunicação de Dados
DCOM	Distributed Component Object Model
DII	Interface Invocação Dinâmica
DME	*Distributed Management Environmet*
DNS	*Domain Name Service*
GDMO	*Guideline for Definitions of Managed Objects*
GIOP	*General Inter ORB Protocol*
IDL	*Interface Definition Language*
IIOP	*Internet inter-ORB Protocol*
ITU-T	*International Telecommunications Union - Telecommunications Standardization Sector*
ISO	*International Organization for Standardization*
JDBC	*Java Database Connectivity*
JDK	*Java Development Kit*
JIDM	*Joint Inter-Domain Management*
LAN	Redes Locais
LLA	*Logical Layered Architeture*
NTP	*Network Time Protocol*

ODL *Object Definition Language*

ODMG *Object database Management Group's*

ODP *Open Distributed Processing*

OLE/COM *Object Linking and Embedding/Component Object Model*

OMA *Object Management Architeture*

OMG *Object Management Group*

OO Orientação a Objeto

OQL *Object Query Language*

OSF *Open Software Foundation*

OSI *Open Systems Interconection*

POSIX *Portable Operating System Interfaces for Unix*

RMI *Remote Method Invocation*

SNMP *Simple Network Management Protocol*

SQL *Structured Query Language*

TCP/IP *Transmission Control Protocol/Internet Protocol*

TELEBRÁS Telecomunicações Brasileiras S.A.

TMN *Telecommunications Management Network*

UDP *User Datagram Protocol*

UTC *Universal Time Coordenated*

UUID *Universal Unique ID*

XDS X.500 *Directory Services*

RESUMO

Atualmente os sistemas de gerenciamento de telecomunicações são poderosos, porém, possuem plataformas centralizadas e proprietárias, limitando com isso, o crescimento das redes de telecomunicações. Este fator gera a necessidade de sistemas escalonáveis e abertos, estimulando os setores da área de telecomunicações a buscarem soluções para o problema em ambientes distribuídos, heterogêneos e totalmente escalonáveis.

Os recentes avanços na área de objetos distribuídos - tecnologia que une os conceitos de programação orientada a objetos e sistemas distribuídos - têm permitido muitas expectativas positivas para o desenvolvimento de sistemas cliente/servidor que utilizam, como se fossem locais, recursos disponíveis através da rede.

Assim, o objetivo deste trabalho consiste na avaliação experimental da potencialidade dos principais serviços do CORBA em conjunto com a Linguagem Java, através dos requisitos necessários para as principais funcionalidades propostas pela arquitetura TMN, para implementação de serviços de telecomunicações com a reutilização de códigos usando reflexão computacional com meta-classes ou meta-nível, substituindo o gerente e classes ou nível-base substituindo os agentes.

PALAVRAS-CHAVES: Sistemas Distribuídos, Objetos Distribuídos, Reflexão Computacional, Arquitetura CORBA, JAVA e Gerenciamento de Serviços de Telecomunicação

ABSTRACT

Currently, the telecommunications management systems are powerful, however, maintain centralized and proprietary platforms, limiting this way, the growth of these telecommunications networks. This factor generates the need for scalable and open systems, stimulating the sectors of the telecommunications area to seek for solutions for the problem in distributed, heterogeneous and totally scalable environments. The recent advances in the area of distributed objects, has created many positive expectations for the development of client/server systems which use, as if they were local, resources available through the network. This way, the objective of this work consists in the experimental evaluation of the potential of the main CORBA services in conjunction with the Java language, through the necessary requirements for the main functionalities proposed by the TMN architecture. For the implementation of the telecommunication services with the reusability of codes we use computational refection with meta-classes substituting the manager and classes substituting the agents.

Keywords: Distributed Systems, Distributed Objects, Computational Reflection, CORBA, JAVA, TMN

1. INTRODUÇÃO

1.1. APRESENTAÇÃO

É incontestável a crescente importância que o mundo atual vem dando às organizações, notadamente às organizações econômicas, produtoras de bens e serviços, que tem por finalidade satisfazer as necessidades humanas.

As influências, positivas ou negativas, de suas atuações na sociedade vem sendo acompanhadas e estudadas com mais afinco, a partir da revolução industrial.

Se o lucro tem sido a principal força propulsora destas organizações, a produtividade tem sido imprescindível à formação do lucro, à disputa do mercado, a ampliação do consumo, uma vez que nela repousa a redução dos custos.

Investir, trata-se também de satisfazer sociedades e, então, não se trata de tarefa fácil, pois a sociedade é, ao mesmo tempo, cada vez mais consumista e também mais exigente: quer mais quantidade; mais qualidade; mais presteza; mais produtos novos e, por outro lado, quer pagar o menor preço e, assim, é mais uma evidência que emerge da Lei da Oferta e da Procura a superioridade do consumidor.

O consumo impõe esta difícil tarefa à produção, de modo a não restar alternativa as organizações, senão a de encontrar um jeito de bater o mais recente recorde de baixo custo, talvez o obtido no dia anterior, e isso significa adoção de melhores técnicas, de melhores processos, de melhores equipamentos, etc., etc., em suma, fazer chegar ao consumidor o produto mais competitivo.

Sem o contínuo aumento da produtividade o mundo não estaria vivendo esta acumulação de capitais, esta diversificação de bens e de serviços, a "globalização", e vivendo esta liberdade de investimentos, de transações comerciais e financeiras do nossos

dias, significa acompanhar os passos da produtividade no mundo inteiro, intensificar estratégias, técnicas, operacionalidade das ações que concorrem para a maior eficiência e eficácia dos objetivos a serem alcançados.

Sem a adequada preocupação em oferecer o melhor produto, seja ele bem, ou serviço, com a finalidade de cativar uma clientela, não se estará tendo boa visão do objetivo. Da mesma forma, resultarão ineficientes e ineficazes resultados se, ao lado de um objetivo mesmo perfeitamente definido e muito bem avaliado, não se dê a devida consideração a fatores importantes como o estratégico e o tático, considerados os níveis mais altos da organização.

Práticas organizacionais, cultura das empresas, com encorajamento à disseminação de idéias, experiências novas e tantos outros fatores concorrem para aumento de produtividade e, as Tecnologia de Informações, não deveremos ter a pretensão de que elas, sozinhas, venham resolver o problema da baixa produtividade.

Contudo estamos consciente da sua importância e que as Tecnologia de Informações pretendem cada vez mais, liderar o elenco dos fatores da produtividade.

Dentro deste domínio complexo, este trabalho pretende analisar serviços de telecomunicações, que é uma das áreas mais importante, dentro deste cenário de competitividade que encontramos.

Nas últimas décadas, temos experimentado os resultados de uma das maiores revoluções na História da Humanidade, a chamada *Revolução pela Informação*. O advento da Tecnologia da Informação penetrou nas grandes organizações, fornecendo soluções em um mundo de negócios complexos e altamente competitivo. Espera-se que as pressões geradas por uma competição, extremamente acirrada, aqueçam ainda mais os ânimos das grandes organizações na busca por inovações tecnológicas. O argumento a favor de investimento em Tecnologia da Informação será simples: a garantia de uma vantagem competitiva.

Muitas pessoas utilizam computadores e redes de comunicação, fazendo ligações telefônicas, enviando correspondências, movimentando conta bancaria, fazendo compra em *shopping centers*, etc.. Dessas pessoas, poucas estão conscientes da comutação de

computadores e de todos os elementos de teleprocessamento que estão suportando este crescente e sofisticado ambiente de negócios. Somente visionários poderiam prever, que a Tecnologia da Informação iria tornar-se parte integral da vida de todos nós.

O professor Michael Porter, da Universidade de Harvard, assim se exprimiu a respeito desta questão : **" A importância da revolução pela informação não é devida à disputa que se impõe. A questão não é se a Tecnologia da Informação irá impactar de maneira marcante a posição do mercado de uma empresa; a questão é como podemos tirar vantagem da oportunidade? As organizações que anteciparem a força da Tecnologia da Informação terão os eventos sob controle, e aquelas que não o fizerem estarão fadadas a uma severa desvantagem competitiva"** [MAR93B].

E, seguindo esta linha de raciocínio, que o presente trabalho pretende oferecer às empresas de telecomunicações uma proposta de integrar os seus serviços à esta Tecnologia de Informação, isto é, integrá-los à tecnologia *Internet* e *Intranet,* em um ambiente de computação distribuída, fornecendo às mesmas um gerenciamento dos serviços e, ainda, abrindo caminho para novos negócios e serviços, tornando-as mais competitivas neste novo ambiente de negócios. Os objetivos deste trabalho serão alcançados com a implementação de objetos de serviços, utilizando o paradigma da Reflexão Computacional com a linguagem Java e o ambiente CORBA.

Os recentes avanços na área de Objetos Distribuídos - tecnologia que une os conceitos de Programação Orientada a Objetos, Reflexão Computacional e Sistemas Distribuídos - têm permitido muitas expectativas positivas para o desenvolvimento de sistemas Cliente/Servidor, que utilizam, como se fossem locais, recursos disponíveis através da rede.

1.2. VISÃO HISTÓRICA DA PESQUISA

Por mais de uma década, a industria de telecomunicações tem se voltado para o estabelecimento de padrões para o gerenciamento de recursos em rede de Telecomunicações.

A arquitetura TMN, provê um ambiente para interfacear uma rede de telecomunicações com sistemas de computação, no sentido de proporcionar funções de gerenciamento TMN em diversos níveis distintos. A série de padrões ITU-T [X.700], foram desenvolvidas para gerenciar redes e elementos de redes em um ambiente TMN, constituindo a parte de gerência de recursos desta arquitetura. Muito esforço foi feito para definir objetos gerenciados e desenvolver plataformas de gerenciamneto para rede de telecomunicações. A série de padrões ITU-T [M3200], foram desenvolvidas para gerenciar serviços proporcionados pelos recursos de rede. Contudo, um menor esforço tem sido feito para gerenciamento de serviços de telecomunicações.

O gerenciamento de redes baseado em arquitetura OSI, não é particularmente adequado para gerenciar serviços, e assim novas tecnologias são necessárias. Neste sentido, CORBA é uma tecnologia candidata para gerenciamento de serviços de telecomunicações. Em [KON96] estão discutidas essas questões.

Por outro lado, a ampla competição mundial e a heterogeneidade de ambientes (padronizados ou não) na industria de telecomunicações tem intensificado a necessidade de gerenciamento de serviço, integrando: um ambiente de criação de serviço; um ambiente de oferecimento de serviço; um ambiente de provisão de serviço; um ambiente de implementação de serviço e um ambiente de gerenciamento de serviço. Como propôs G.Chen e Q.Kong em [KON97]. Em seu trabalho, Chen e Kong, propõem um ambiente integrado de provisão e gerenciamento de serviços, com ambiente de usuário baseado em Java, um ambiente de gerenciamento de serviço baseado em CORBA e um ambiente para gerenciamento de rede baseado em TMN.

No Brasil, a pesquisa sobre gerência de serviços, ainda é inicial, como se pode ver em [PEN96], sobre sistemas de gerenciamento distribuído com o uso de OSF/DCE.

1.3. OBJETIVOS E METAS

Atualmente os Sistemas de Gerenciamento de Redes de Telecomunicações (TMN) são poderosos, porém possuem plataformas centralizadas e proprietárias. A capacidade de gerenciar sistemas de grande porte e, principalmente, a possibilidade do sistema crescer,

juntamente com a rede de telecomunicações fica comprometida. Este fato gera uma necessidade de sistemas escaláveis e abertos, motivando os setores da área de telecomunicações a buscarem soluções para o problema em ambientes distribuídos. Neles poderiam ser elaboradas plataformas de gerenciamento abertas, distribuídas e totalmente escaláveis.

Mesmo que as empresas adotem esta solução para gerenciamento de rede de telecomunicação, isto não é suficiente para propiciar uma solução completa, ao gerenciamento de serviços, pois serviços são dinâmicos, enquanto os elementos de rede são estáticos. Novos serviços frequentemente precisam ser criados, implantados e disponibilizados com rapidez. Por exemplo, as facilidades de modelagem de objetos atualmente usadas (como GDMO), não são suficientes para modelar os objetos e nem funções de gerenciamento de serviços[KON96]. Para que se consiga dominar rapidamente as mudanças nos serviços, na criação de novos serviços e na interoperabilidade, verificou-se a necessidade de novas ferramentas.

Aplicações de Gerenciamento de Serviços são necessárias para administrar serviços geograficamente dispersos com computadores heterogêneos e recursos de telecomunicação muito complexos. O propósito do processo de gerenciamento é realizar acessos e modificações transparentes, consistentes e confiáveis, destes recursos.

É importante para as indústrias desenvolver novas tecnologias afim de que os recursos da rede global possam ser compartilhados e gerenciados por aplicações de uma maneira eficiente e consistente. Empresas de telecomunicações distinguem-se umas das outras pelo oferecimento e gerenciamento dos serviços.

Uma rede TMN possui grande número de objetos relativamente simples. Tecnologias de rede TMN, baseada em arquitetura OSI, é adequada para gerenciamento de objetos e elementos de rede. Estes objetos modelam os elementos, a topologia e a hierarquia de uma rede real. Eles são relativamente simples, em razão de que os objetos não conterem operações e relacionamentos complicados. GDMO é a ferramenta apropriada para especificar os objetos e suas operações, comportamento e notificações. CMIS/CMIP são

usados para definir serviços de comunicação e protocolo, através de diferentes entidades responsáveis pela implementação e pela invocação de objetos.

Por outro lado, serviços TMN são normalmente modelados, através de um número relativamente pequeno de objetos complexos. Estes objetos representam um alto nível de abstração da rede, de relacionamento entre seus componentes e de serviços computacionais suportados pela rede. Por exemplo, um serviço de teleconferência pode ter um pequeno número de instâncias, representando serviços distribuídos e componentes gerenciáveis oferecidos. Isto, entretanto, é suportado por uma grande parte dos elementos de rede, e de vários sistemas de gerenciamento de informações. O acesso a um serviço de teleconferência pode invocar funções muito complexas, incluindo contabilidade, segurança, performance, interoperabilidade, e qualidade de serviços oferecidos.

Dado a estes parâmetros e características de gerenciamento, os serviços de telecomunicações são difíceis de modelar com as facilidades do GDMO e de implementar usando a arquitetura de gerenciamento de rede OSI.

A tecnologia CORBA oferece uma melhor alternativa para modelar estes complicados objetos, no Gerenciamento de Serviços Telecomunicações.

A arquitetura CORBA suporta um ambiente mais adequado a definição, à transporte, à implementação e a invocação de objetos. Também suporta separação e distribuição da implementação de objetos, das suas definições, de tal forma que os objetos de serviço poderão ser rapidamente projetados, implementados e gerenciados. Esta tecnologia oferece facilidades para especificar operações complicadas de objetos e interfaces genéricas , invocando estas operações.

Todas estas características fazem com que a arquitetura CORBA seja bastante apropriada a ambiente de implementação e gerenciamento. Para integrar o gerenciamento à tecnologia *Internet/Intranet,* é objetivo deste trabalho utilizar a linguagem *Java*, à qual utilizará o Paradigma de Reflexão Computacional para implementação dos objetos propostos.

O presente trabalho objetiva a construção de um modelo e a sua implementação no sentido de provisão e gerência de serviços de telecomunicações. Este sistema contém o

ambiente do usuário baseado em Java e o um ambiente de serviços distribuídos baseado em CORBA. Este trabalho, não têm como objetivo abordar gerenciamento de recursos da rede, por exemplo, o TMN, devido a independência entre aplicações de gerência de serviços e os próprios serviços em si, com relação aos elementos de rede e de seu gerenciamento.

O sistema construído fornece subsídios, através de aplicativo de gerência de serviço, para a gerência de rede configurar equipamentos e acionar força de trabalho, caso seja detectado alguma falha no serviço decorrente de alguma falha no elemento de rede. Além disso, pode disponibilizar informações referentes a contabilização armazenada numa MIB de serviço.

1.4. Publicações

Essa dissertação produziu artigos que foram publicados e selecionados para os seguintes eventos:

- ✓ ITS'98 Proceedings SBT/IEEE International Telecommunications Symposium, ocorrido de 09-13 de agosto de 1998, no Maksoud Plaza, São Paulo – Brasil. E Publicado nos Anais da IEEE, com o nº 0-7803-5030-8/98/$10.00 © 1998 – IEEE, nas páginas 486 até a 491;
- ✓ WEBSIM'99 International Conference On Web-Based Modelling & Simulation, que ocorrerá de 17-20 de janeiro de 1999, em Cathedral Hill Hotel, San Francisco, California.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. ORIENTAÇÃO A OBJETO

2.1.1. Introdução

O conceito fundamental de **Objeto** significa uma entidade, cujas informações podem incluir desde suas características, ou dados (Atributos), até os procedimentos para manipulação dessas características (Serviços). A um objeto sempre estarão associados seu estado, seu comportamento e sua identidade. O estado é definido pelas propriedades do objeto e pelos valores que as propriedades possuem. O comportamento define como o objeto age e reage, tanto em relação a mudança em seu estado, como à comunicação com outros objetos. A identidade de um objeto é a propriedade pela qual ele se distingue dos demais [KHO86].

Quando objetos possuem características comuns, pode-se agrupá-los em uma **classe**, na qual são definidos os dados e procedimentos associados a todos os objetos da classe. Este tipo de entidade denomina-se **Classe & Objetos**. Algumas classes são generalizações, e não possuem objetos. Estas são denominadas **Classes**. Dados e procedimentos associados a uma **Classe** ou a uma **Classe & Objeto,** são chamados de **membros** da classe.

O primeiro tipo de membros de uma classe são os **Atributos** . Eles descrevem as propriedades do objeto (estado). A manipulação dos atributos é efetivada pelos membros denominados **serviços** (ou métodos) associados à classe.

Quando o atributo deve ser distinto para cada objeto, ele caracteriza um **Atributo de Objeto**. Estes podem ter até mesmo valores idênticos mas são entidades separadas no sistema. Além destes, uma classe pode possuir atributos cujos valores devem ser compartilhados entre todos os seus objetos. A estes, dá-se o nome de **Atributos de Classe**.

Quando um Objeto modifica o valor de um Atributo de Classe, todos os demais objetos da classe têm seus valores modificados.

Um exemplo de objeto pode ser obtido quando consideramos um programa processador de textos. Nele podemos definir a seguinte classe & objeto :

Classe & Objeto : parágrafo;

Atributos : fonte e textos do parágrafo;

Método : formatar-se.

Um objeto desta classe seria, pôr exemplo, o parágrafo que você está lendo.

Ao definir se uma Classe, é necessário que se especifique o tipo de acesso que seus membros (atributos e serviços) terão: Acesso público permite que todas as outras entidades do sistema (funções e demais classes) acessem este membro; Acesso protegido permite que as classes, na hierarquia da classe em questão possam acessar os membros sob essa especificação; finalmente, os membros que só devem ser acessados pelos serviços da classe, possuem acesso privado.

Quando um programa orientado a objeto está sendo executado, ocorre processo de comunicação entre objetos, através do envio de **mensagens** (chamadas de funções ou procedimentos). Estes procedimentos definem o comportamento que o objeto receptor deverá ter. Para alcançar este objetivo, o objeto aciona aqueles serviços relacionados à mensagem enviada. Todo o processamento é definido pelo envio, pela interpretação e pelas respostas às mensagens entre objetos.

À medida que as características de objetos semelhantes vão sendo agrupadas, vão se formando uma *Classe*. Isto redefine Objeto como sendo sempre a **Instância** de uma classe.

Outro mecanismo importante na Orientação a Objetos, é o mecanismo de agrupamento de características, em que algumas classes (filhos) são variações especializadas de outras (pais), constituindo a **Herança**. Como exemplo, pode-se citar o fato de que alguns atributos da classe (Automóvel modelo, ano, etc.), também estão presentes em uma classe mais geral, denominada Veículo, cujas subclasses, além de Automóvel, comportam Avião, Navio, etc.

Quando a classe derivada possui características em mais de uma classe-base, a hierarquia constitui uma **Herança Múltipla**.

A **Abstração** consiste na formulação do problema sob a ótica mais adequada à solução. Na Análise Orientada a Objeto, todo o processo de definição de Classe, Hierarquia, Atributos, Métodos e Mensagens, que formarão o programa orientado a objeto, é fruto da abstração do analista e, como tal, é de suma importância no alcance dos objetivos.

Na programação fundamentada em procedimentos, os programas são baseados em coleções de funções. Estas modelam operações abstratas cujo intuito é resolver um problema de programação. Na Abstração de Dados, o foco está nas estruturas de dados e não nas operações que se fazem com elas, Elas são, inclusive, parte do dado que se modela . Em outras palavras, a abstração no paradigma da Programação Estruturada está nos procedimentos. No paradigma da Orientação a Objeto, tem-se Dados Abstratos, onde uma estrutura de dados inclui as operações que ocorrem com ela. Para efetivar este conceito, a técnica usada é encapsular dados e procedimentos em um tipo abstrato de dado.

Uma das principais diferenças entre a Análise Estruturada e a Análise Orientada a Objetos é a característica do **Encapsulamento** dos objetos. Segundo ela, a definição de um objeto inclui tanto seus atributos como os métodos que agem sobre esses atributos. Na Análise Orientada a Objetos, não se separam dados de procedimentos, como se faz na Análise Estruturada.

Outra característica relevante da Orientação a Objetos é o **Polimorfismo**. Trata-se da propriedade de métodos com diferentes códigos e de diferentes níveis na hierarquia de classes que apesar de possuírem o mesmo nome podem ser diferenciados pelo contexto em que estão sendo chamados. Exemplo comum é a operação de nome *on* (ligar), em vários eletrodomésticos, computadores, máquinas sofisticadas, etc. Embora os nomes sejam idênticos, os procedimentos em cada situação são, evidentemente, diferentes, e é o contexto que define a quais desses procedimentos está-se referindo.

Um sistema orientado a objetos deve apresentar, também, **modularidade** . As abstrações semelhantes devem ser agrupadas em módulos independentes que, quando juntos, formam o sistema. Isto permite maior flexibilidade e eficiência na implementação do sistema.

Outro conceito-chave da Orientação a Objetos é **Persistência**. Objetos Persistentes são aqueles que permanecem existindo, mesmo após o término da execução do programa.

Associados à persistência, estão o gerenciamento dinâmico da memória e o armazenamento de objetos em base de dados. Os objetos não-persistentes devem ter seu espaço de memória dinamicamente gerenciados, ou seja, alocados, quando necessário e, liberados para outro uso, quando não mais importar no processamento do programa. Em termos de memória auxiliar, os objetos armazenados em bases de dados orientadas a objetos são ditos persistentes.

Uma característica desejável no modelo da Orientação a Objetos é a **Tipificação**.

O uso do objeto de uma classe onde em que se deseje o uso do objeto de outra classe, só deve ser permitido sob condições controladas (conversões explícitas). Dentre as linguagens de Orientação a Objetos, encontraremos linguagens fortemente ou fracamente tipadas, ou mesmo linguagens não-tipadas, dependendo da liberdade que a linguagem fornece no intercâmbio de tipos.

Várias são as notações disponíveis para Sistemas Orientados a Objetos. As principais são: a de Booch ([BOO92a]; [BOO92b]) e a de Coad e Yourdan [COA92]. Para estas, inclusive, encontramos ferramentas (CASE) no mercado para especificação de Sistemas Orientados a Objetos. Ambas as metodologias são relevantes: Booch por sua contribuição ao nível de definição e ilustração de conceitos; Coad e Yourdon [COA92] por apresentar a notação gráfica que nos parece mais clara.

2.2. ARQUITETURAS DISTRIBUÍDAS ORIENTADAS A OBJETOS EM AMBIENTE HETEROGÊNEO

2.2.1. Introdução

O crescimento e a popularização de redes de computadores observado nos últimos anos, tanto em número, quanto em tamanho e em complexidade, vieram acompanhados da necessidade de interconectar redes anteriormente isoladas para permitir que computadores em diferentes redes compartilhem informação e trabalhem de forma cooperativa.

Para tornar possíveis tais procedimentos, tem se demonstrado a necessidade de sistemas abertos, em que seja viável a comunicação com máquinas de outros fabricantes e em redes com topologias diferentes e/ou redes que utilizem protocolos diferentes; cujos modelos possibilitem a comunicação, a sincronização, o compartilhamento de recursos e a nomeação global, e não resultem na perda da segurança do sistema, não sobrecarregando a administração da rede, não punam o usuário comum pela adoção de tecnologias totalmente novas, mas incompatíveis com as ferramentas que estava acostumado a lidar.

Tem-se mostrado necessário também um sistema de processamento distribuído adequado à capacidade das novas máquinas disponíveis a preços acessíveis para um grande número de usuários, e as novas redes de fibra ótica de alta velocidade e capacidade de transmissão que estão sendo instaladas. E ainda, com a popularização dos serviços de hipertexto via rede, o *voice mail,* as teleconferências, e outros serviços baseados em redes de computadores, tem crescido o interesse, tanto de parte dos usuários como das grandes empresas de *hardware* e *software*, por um sistema, que seja adequado a este novo cenário no qual a informática ocupa todos os espaços possíveis, e que possua as características descritas anteriormente.

Com o intuito de propor padrões para sistemas abertos com arquitetura de suporte ao desenvolvimento de aplicações distribuídas em ambientes heterogêneos. Várias empresas têm trabalhado para impor um padrão para este ambiente e para que este padrão seja aceito pelas organizações internacionais e/ou pelo mercado (padrão de fato). Dentre as arquiteturas projetadas até o momento temos : DCE (*Distributed Computing Environment*); ANSA

(*Advanced Network System Architecture)*; CORBA (*Common Object Request Broker Architecture*); OLE/COM, *OpenDoc*; DCOM e JAVA. Em ordem cronológica, DCE e ANSA são tecnologias mais antigas e as demais são as mais recentes, estas arquiteturas mais recentes utilizaram a maior parte dos conceitos e modelos de ambas para implementar suas novas propostas.

2.2.2. Histórico do Ambiente de Computação Distribuída

Com o objetivo de propor padrões para sistemas abertos, foi criada por um conglomerado de empresas a OSF - *Open Software Foundation* – que foi responsável pela proposição e pela implementação de novas Tecnologias nesta área. Os membros da OSF foram então convidados a propor aplicações baseadas nesta nova tecnologia, e a companhia escolhida (ou mais de uma delas, trabalhando em conjunto) desenvolvem o trabalho. Finalizando o trabalho de desenvolvimento, a OSF fornece o código fonte a todas as companhias que o licenciarem, que a transportam para as suas plataformas e vendem o produto, pagando *royalties* para OSF, que repassa uma percentagem para as companhias que desenvolveram o *software*.

Seguindo estes procedimentos, foi proposto o desenvolvimento de um ambiente de processamento distribuído que interconectasse um conjunto de redes de computadores em uma única rede de maior abrangência. Este sistema de *software* consistiria, então, de uma camada que esconde as diferenças entre os vários tipos de máquinas e de redes disponíveis no mercado. Desta forma, surgiu o DCE, um ambiente de desenvolvimento e de execução de Aplicações Distribuídas, que proporciona melhor aproveitamento do poder de processamento dos componentes de rede, como computadores, discos e memória. Pelo fato de partes de uma aplicação poderem ser executados concorrentemente e em máquinas diferentes, pode-se obter uma performance melhor, com maior disponibilidade e com maior robustez, do que a mesma aplicação se executada seqüencialmente em um sistema mono-processado.

Os Sistemas Distribuídos têm as seguintes vantagens sobre os centralizados:

Economia	Os microprocessadores oferecem uma melhor relação preço/performance do que a oferecida pelos *mainframes;*
Distribuição Inerente	Algumas aplicações envolvem máquinas separadas fisicamente;
Confiabilidade	Se uma máquina falhar, o sistema, como um todo, sobrevive;
Crescimento Incremental	O poder computacional pode crescer em doses homeopáticas;
Velocidade	Um sistema distribuído pode ter um poder de processamento maior do que o de qualquer *mainframe;*

Os Sistemas Distribuídos têm as seguintes vantagens sobre os computadores pessoais que operam isoladamente:

Compartilhamento de Dados	Permite que mais de um usuário acesse uma base de dados comum;
Comunicação	Torna muito mais simples a comunicação pessoa a pessoa, Por exemplo, empregando o correio eletrônico;
Flexibilidade	Espalha a carga de trabalho Por todas as máquinas disponíveis ao longo da rede;
Compartilhamento de Dispositivos	Permite que mais de um usuário possa ter acesso a periféricos muito caros, tais como impressoras laser.

Desvantagens encontradas em Sistemas Distribuídos:

Software	Até o momento não há muita disponibilidade de *software* para os sistemas distribuídos;
Ligação em Rede	A rede pode saturar;
Segurança	Os dados secretos também são acessíveis facilmente.

Um Ambiente de Computação Distribuída é mais do que simples recurso de comunicação. Oferece grande faixa de recursos computacionais para a aplicação, independente da localização do usuário, do tipo de aplicação ou dos recursos computacionais requeridos. Esses recursos permitem que as aplicações ofereçam aos usuários melhor desempenho e uso mais eficiente dos serviços computacionais do sistema. Este ambiente está baseado em muitos padrões, formais e de fato, tais como [ROS92]: protocolos TCP/IP *internet; threads* do POSIX 1003.4a *draft* e listas de controle de acesso do POSIX 1003.6 *draft*; Serviços de Diretório do CCITT X.500/ISO 9594; Sistema de Nome de Domínio *internet* (DNS) e Protocolos de sincronização de relógio em rede (padrão NPT - *Network Time Protocol*); Serviços de Diretório X/*Open* (XDS) e Interface de Programação da Aplicação X/*Open*.

2.2.3. Tecnologias de Computação Distribuídas

2.2.3.1. DCE (*Distributed Computing Environment*)

A arquitetura DCE oferece um ambiente de comunicação que permite as informações fluírem do lugar onde está armazenada, para o lugar onde está sendo requisitada, sem expor as complexidades da rede para: o usuário final; para o administrador do sistema; ou para o desenvolvedor de aplicação. A arquitetura do DCE mascara as complexidades físicas do ambiente, oferecendo uma camada de simplicidade lógica, composta de um conjunto de serviços que podem ser usados separadamente, ou em combinação, para formar um ambiente de computação distribuída mais fácil de ser compreendido.

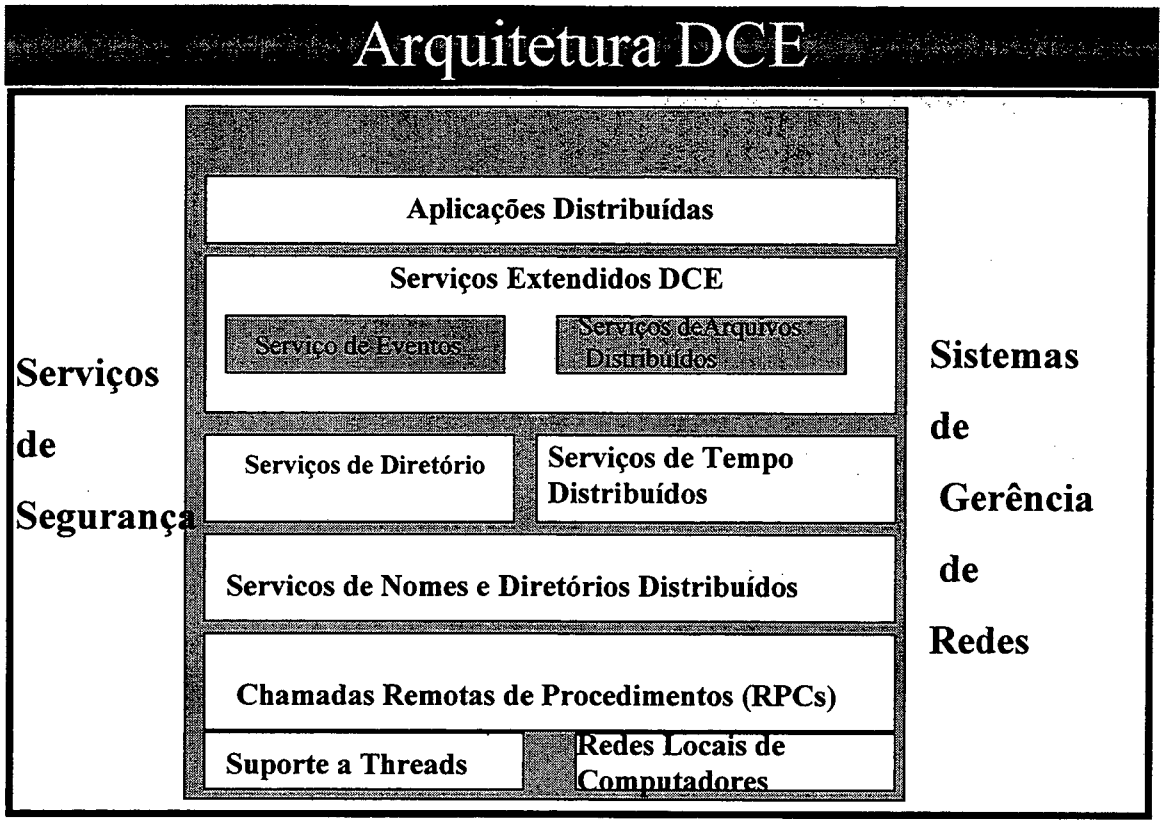


Figura 1 . A arquitetura DCE

A figura 1 mostra a composição da arquitetura DCE em um modelo de camadas, que integra um conjunto de tecnologias para Sistemas Distribuídos. A arquitetura fornece tanto serviços de mais baixo nível (sistema operacional) , como serviços de mais alto nível. O DCE consiste de um conjunto de serviços essenciais para a programação em Sistema distribuídos, e é composto dos seguintes componentes:

- Serviço de segurança : oferece um conjunto de mecanismo, para desenvolver aplicações com suporte a comunicação segura entre cliente e servidor, os mecanismos são :
  - Autenticação: seguindo o sistema *Kerberos* V5, clientes e servidores podem provar quem é quem, no sistema;
  - Autorização: servidores podem usar uma lista de controle de acesso para determinar quais clientes estão autorizados a utilizar seus serviços;
  - Integridade: garante que o conteúdo da informação recebida permanece exatamente a mesma, desde a sua transmissão;

• Privacidade: através de técnicas de criptografia, protege as informações confidenciais, e impedem que sejam acessadas durante as transmissões entre cliente e servidor.

• Serviço de Diretório (CDS): é o mecanismo pelo qual nomeia-se logicamente os objetos dentro de uma célula DCE (um grupo de máquinas cliente e servidor). As aplicações identificam os recursos por nomes, sem a necessidade de saber onde o recurso está localizado. Células DCE podem também participar de um serviço de diretório mundial, usando o serviço de diretório global do DCE (GDS), que é baseado no padrão X.500, ou no serviço de nome de domínio estilo *internet* (DNS);

• Serviço de Tempo Distribuído: é aquele que oferece um modo para sincronizar os relógios de todas as máquinas de uma célula DCE, bem como a sincronização entre células. Este serviço segue o padrão UTC (*Universal Time Coordenated*) e pode interoperar com o NTP (*Network Time Protocol)*;

• Chamada de Procedimento Remoto (RPC): é um mecanismo de comunicação fundamental. Permite que clientes façam chamadas de procedimentos em servidores remotos como se fossem chamadas de procedimentos local. Um cliente pode usar o serviço de diretório para conectar a um servidor particular de interesse em tempo de execução. Além disso, o cliente e servidor podem usar serviços de segurança para garantir os níveis de autenticação, autorização, integridade e de privacidade. O mecanismo de RPC, isola do cliente: os detalhes da localização dos servidores na rede; o tipo de plataforma de máquina ou do sistema operacional em que eles executam; das diferenças na representação dos dados entre plataformas; o transporte de rede em uso, e permite que programas distribuídos trabalharem de forma transparente em sistemas heterogêneos.

• Serviços de *Threads*: é um pacote de *threads* baseado no padrão POSIX 1003.4a (*draft 4*) que suporta a criação e gerenciamento de múltiplas *threads* de controle , dentro de um cliente ou servidor. As *threads* são linhas de execução (*lightweight)*, dentro de um programa, que podem ser usadas para desempenhar ações concorrentes.

Além destes serviços, existem também os serviços estendidos DCE (figura 1), que atualmente consistem do serviço de arquivos distribuídos (DFS), do sistema de eventos e

gerenciamento de rede. O DFS é uma aplicação que implementa através do serviço de diretório, um simples sistema lógico de arquivos, que está disponível na célula. É um método seguro, escalável, de alto desempenho para compartilhamento de arquivos remotos. O DFS suporta replicação de arquivos para alta disponibilidade e tolerância a e recuperação das falhas de *hardware*, buscada em *logs*. O DFS aparece para o usuário como um sistema de arquivo local, oferecendo acesso a qualquer arquivo da rede e para qualquer usuário, com o mesmo nome usado por todos (acesso uniforme). O serviço de gerenciamento de rede oferece meios para a Aplicações Específicas acessar informações de gerenciamento, usando protocolos de gerenciamento de rede padrão, tais como CMIP e SNMP. O Serviço de Eventos oferece um modo comum para Sistema e Aplicações de Usuário , direcionar, filtrar e guardar eventos do sistema.

O DCE é extremamente escalável e, para garantir esta propriedade, permite dividir um grande ambiente em unidades gerenciáveis independentes chamadas *células*. Uma célula é a unidade básica de operação e administração do DCE. É um grupo de usuários, sistemas e recursos que têm um propósito comum e compartilham serviços DCE. No mínimo, uma célula deve ter na sua configuração o *serviço de diretório, o serviço de segurança e o serviço de tempo* (figura 2).

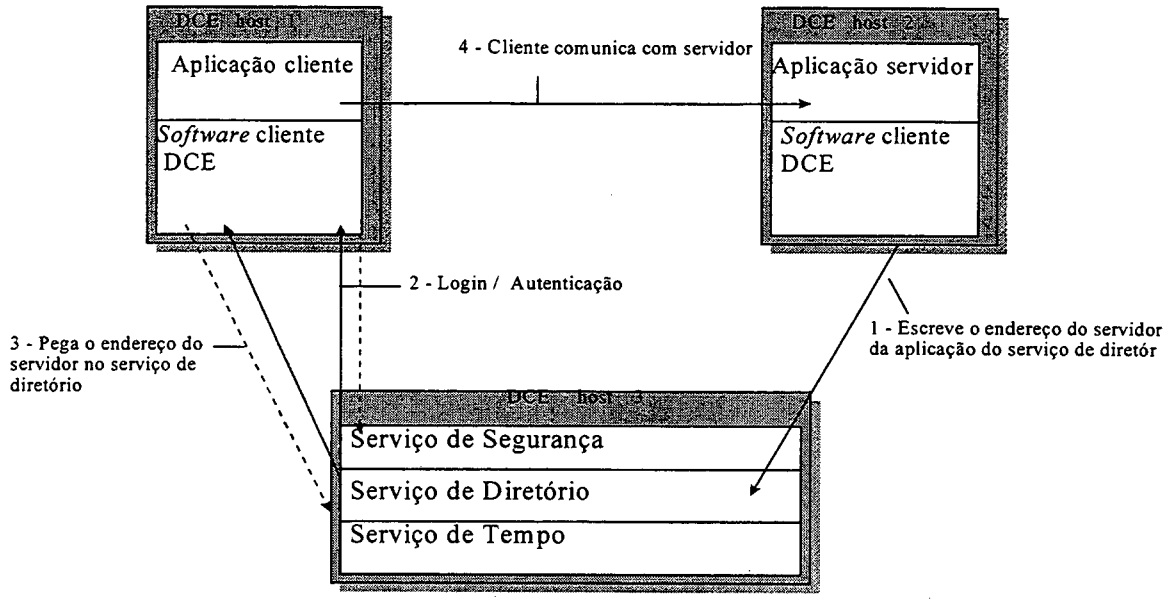


Figura 2 . Operação distribuída em uma célula DCE

Dentro da OSF foi proposto uma extensão do DCE para o desenvolvimento de Aplicações Distribuídas Orientadas a Objetos denominada OO-DCE [DIL93]. Esta extensão é composta por uma biblioteca de classes de objetos, que auxilia o programador no desenvolvimento das aplicações orientadas a objetos. Os métodos oferecidos por um servidor são declarados usando uma linguagem de definição de interface (IDL). No entanto, a IDL do DCE não oferece mecanismo de herança, e nem os mecanismos de invocação são suficientemente completos para abranger uma ampla faixa de modelos de objetos existentes.

2.2.3.2. ANSA (*Advanced Network Systems Architecture*)

O projeto ANSA [HER94] é uma implementação do modelo de referência ODP (*Open Distributed Processing*) [ISO 95]. Este modelo de referência foi definido pela ISO e tem como objetivo principal permitir a interoperabilidade entre aplicações e, também o compartilhamento de informações entre instituições. O modelo ODP consiste de um conjunto de níveis de abstração, que descrevem a estrutura do sistema distribuído sob

cinco pontos de vista: empresa, informação, computacional, engenharia e tecnologia. Com isto, é possível obter-se a relação entre os requisitos do sistema e as soluções técnicas para o desenvolvimento de produtos comerciais.

Projeto ANSA segue o modelo de referência ODP, visando o desenvolvimento de tecnologia para o modelagem de Sistemas Distribuídos complexos. ANSA consiste de um conjunto de modelos que estão de acordo com os vários aspectos de projeto de Sistemas Distribuídos e problemas de representação. Este conjunto de modelos, conhecido como modelo de projeção, é exatamente aquele especificado pelo ODP, definido da seguinte maneira:

- Modelo de empresa: expressa as políticas, os limites e os propósitos do sistema, trata, com as regras e funções globais da estrutura organizacional dentro do qual o problema em questão será tratado;
- Modelo de informação: representa o fluxo da informação dentro da empresa, o propósito da informação distribuída e a forma de processamento desta;
- Modelo computacional: trata dos aspectos relativos as estruturas de programação e das ferramentas de desenvolvimento de programas disponíveis ao programador, isto é, de questões como modularidade da aplicação, configuração, concorrência, sincronização, replicação e a extensão das linguagens existentes para suporte a computação distribuída;
- Modelo de engenharia e tecnologia: permite abstrair do modelo abstrato, obtido dos modelos anteriores em um modelo mais refinado e prático.

A definição da arquitetura ANSA é estruturada através de um conjunto de requisitos para assegurar que as aplicações possam ser portáveis e também para que a interoperabilidade seja alcançada mais facilmente. Esta arquitetura permite o desenvolvimento de diferentes projetos, cada qual direcionado para uma aplicação particular ou para área de aplicação. A arquitetura pode ser usada como um *framework,* para comparar as diferentes otimizações de projeto, e auxilia o projetista na análise de como combinar sistemas dissimilares, Esses requisitos são definidos da seguinte forma:

- Conjunto de componentes: formam a base para construir blocos e ferramentas da arquitetura;

- Conjunto de regras: restringe a forma como tais componentes podem ser combinados no projeto da arquitetura;
- Conjunto de receitas: oferece artifícios de como combinar componentes básicos usando as ferramentas para obter subsistemas com certas propriedades distintas;
- Conjunto de diretrizes: ajuda o projetista a fazer decisões de projeto, se sua preferência não oferecer bases suficientes para tais decisões.

Em relação a objetos distribuídos, o modelo ANSA especifica um conjunto de conceitos e padrões para que objetos possam se interoperar em ambientes heterogêneos [HER94], isto é, como os objetos devem ser alcançado pela padronização do modo como os objetos devem ser invocados no sistema, independente se estão situados em sítios locais (*local sites*), ou remotos. Em termos práticos, cada componente do sistema descritos é como um objeto com uma interface bem definida e pública. Uma aplicação, com a permissão apropriada, pode acessar os serviços, obtendo um manipulador do objeto e invocar seus métodos, usando os procedimentos e funções definida nesta interface. Uma invocação no ambiente ANSA é feita de forma automática e transparente, isto é, o sistema garante isso, tratando de aspectos, tais como:

- Buscar o objeto no repositório;
- Lançar programas quando uma nova instância de um objeto for requisitada;
- Implementar o protocolo de invocação de objetos;
- e etc..

Uma outra característica deste sistema é a de ser possível obter serviços com alta disponibilidade e confiabilidade, através de serviços de transação e de processamento de grupo, para tolerância a falha e alta disponibilidade [OSK94]. Trabalhos recentes no ANSA têm sido propostos a adoção do padrão CORBA, como a arquitetura para objetos distribuídos em sistemas abertos.

2.2.3.3. Arquitetura CORBA (*Common Object Request Broker Architecture)*

O padrão CORBA/OMG (figura 3) é um conjunto de padrões e conceitos para objetos distribuídos em ambientes abertos, proposto pela OMG (*Object Management Group*) [OMG 91] [OMG 96]. Segundo essa arquitetura, métodos de objetos remotos podem ser ativados de forma transparente, em ambientes distribuídos e heterogêneos, através de um ORB (*Object Request Broker*). O ORB, num sentido genérico, é um canal de comunicação para objetos distribuídos. No CORBA, é possível também obter interoperabilidade entre objetos desenvolvidos em linguagens diferentes de programação (C, C++, Ada, Cobol, Java, Smaltalk), em que as diferenças de cada uma são mascaradas pelo CORBA através do mapeamento adequado da IDL (*Interface Definition Language*) [OMG 94], para a linguagem de programação desejada. Cada objeto CORBA tem sua interface especificada em IDL, uma linguagem declarativa, sem nenhuma estrutura algorítmica, com sintaxe e tipos predefinidos, baseados na linguagem C++. Portanto, o uso de uma linguagem de definição de interface (IDL) permite tratar das heterogenidades do ambiente, tais como os diferentes tipos de máquina, sistemas operacionais e linguagens de programação.

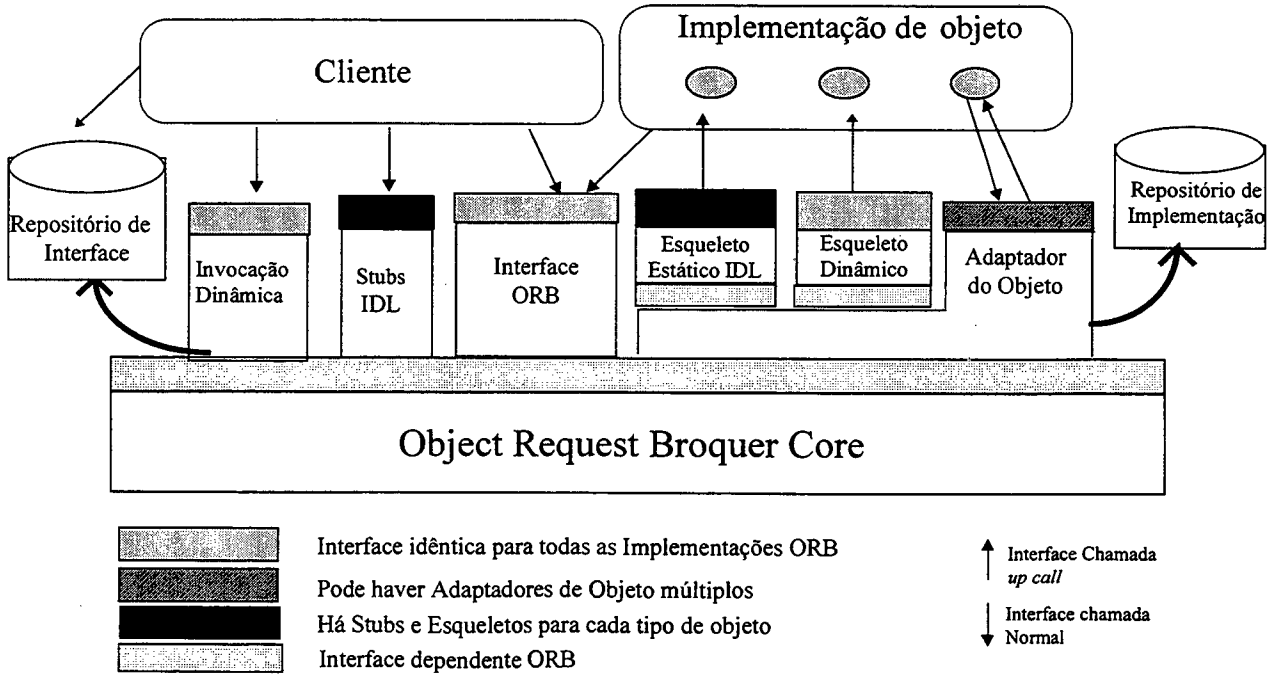


Figura 3: A arquitetura CORBA 2.0

Na arquitetura CORBA, é especificada um adaptador de objeto básico (BOA) que oferece um modo primário para implementação de objeto, acessando os serviços oferecidos pelo ORB. Alguns dos serviços oferecidos pelo ORB, através do BOA, são: geração e interpretação de referência de objetos; métodos de invocação; segurança de interações; ativação e desativação de objeto e implementação, mapeamento de referência de objeto para implementações e para registro de implementação.

O BOA é um adaptador de objeto padronizado pela OMG. Contudo, pode haver diferentes tipos de adaptadores de objetos, de acordo com as necessidades especificas de uma implementação de objeto que escolha qual irá utilizar, baseando-se no tipo de serviço que se deseja obter do ORB.

A interface do ORB, segundo o padrão CORBA, deve ser idêntica para as diferentes implementações CORBA. Esta interface deve ser independente de qualquer interface de objeto ou adaptador de objeto. Devido a maioria das funcionalidades do ORB serem oferecidas, através dos adaptadores de objetos, *stubs, skeleton* ou DII (Interface de Invocação Dinâmica), nesta interface é oferecida apenas um pequeno número de operações que são comuns, tanto para clientes como para implementações de objetos.

O Repositório de interfaces é um serviço que oferece objetos persistentes é que representam a informação IDL em uma forma disponível, em tempo de execução. As informações disponíveis no repositório são geralmente utilizadas pelo ORB para realizar invocações dinâmicas a objetos remotos. Usando a informação disponível no repositório, é possível a um programa encontrar um objeto remoto, mesmo desconhecendo sua interface, seus métodos e parâmetros, e sua forma de ativação. Já, o repositório de implementações contém informações que permitem um ORB localizar e ativar implementações de objetos.

As interações no ambiente CORBA seguem o modelo Cliente/Servidor.

Uma requisição do cliente a um servidor remoto é transmitida através da rede, usando o ORB que localiza o objeto remoto, transporta os dados e passa o controle para o adaptador de objeto básico (BOA). Neste ponto o BOA prepara a implementação de objeto

para receber a requisição e depois ativa a operação requisitada na implementação do objeto servidor, através do esqueleto(*Skeleton*) IDL. Quando o processamento da requisição termina, o resultado é passado para o ORB que o entrega ao cliente de forma transparente.

Na requisição de um serviço, o cliente pode realizar uma invocação no servidor de duas formas: estática através de *stubs* IDL, ou dinâmica, através da interface de invocação dinâmica (DII). Em ambas situações, a implementação de objeto (no servidor) não percebe o tipo de invocação utilizado na requisição pelo cliente. Na invocação estática, o cliente faz a invocação de um método no objeto-servidor, utilizando-se de *stubs* IDL apropriados, gerados no processo de compilação do arquivo IDL correspondente. Uma *stub* pode ser entendida como uma representação local do objeto remoto. O processo de definição de serviços, seguindo uma abordagem de invocação estática, é mostrada na figura 4.

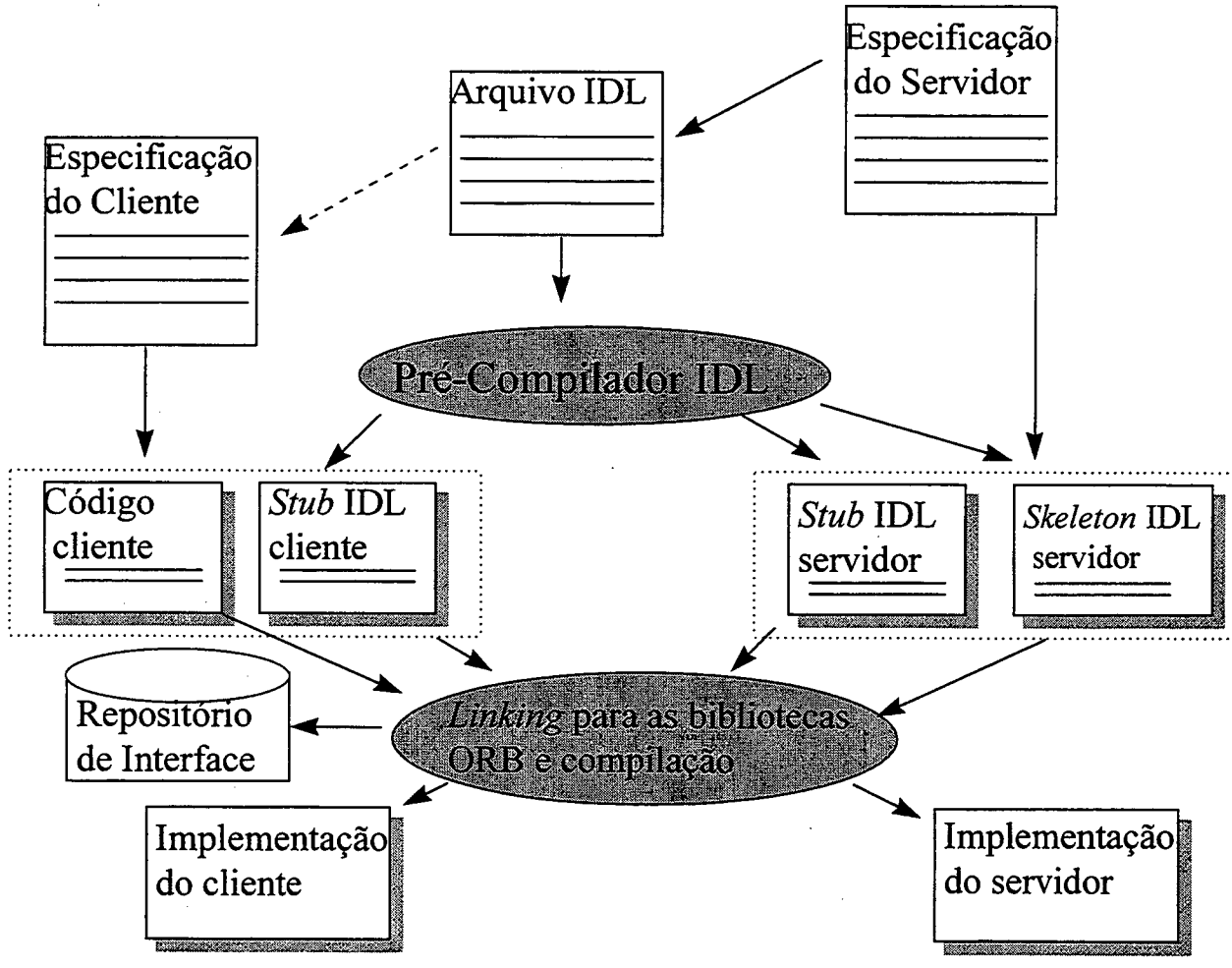


Figura 4. Abordagem de invocação estática utilizando stubs IDL

A partir da especificação do servidor, é feita a declaração dos serviços oferecidos por ele no arquivo de interface IDL. Este arquivo passa pelo pré-compilador IDL, que gera *stubs IDL* do cliente, e *stub e skeleton* IDL do servidor. As *stubs* são códigos responsáveis por interagir com o ORB, para permitir as interações entre cliente e servidor, e devem se manter inalterados. Uma vez, isto feito, o programador deve, a partir da especificação do servidor, inserir o código de implementação no *skeleton* IDL. O *skeleton* é um arquivo montado pelo compilador IDL, de forma estruturada e pronta para receber os códigos de implementação dos métodos, declarados no arquivo de IDL. Com o *skeleton* e *stubs* servidor prontos, é feito o *linking* com a biblioteca do ORB e com a compilação, para gerar o executável do servidor. Neste mesmo ponto, é feito o registro da interface do servidor, no repositório de interface. Para implementar um cliente, o programador deve ter a informação dos métodos oferecidos pelo servidor, através do arquivo IDL, e, também, a referência do objeto servidor. Com isso, é escrito o código de implementação dos procedimentos do cliente e o processo de geração do código executável do cliente é o mesmo do servidor.

Comparando com a figura 5, observa-se que a abordagem de invocação estática, utilizando *stubs* IDL é bem semelhante à uma aplicação utilizando RPC (Remote Procedure Call).

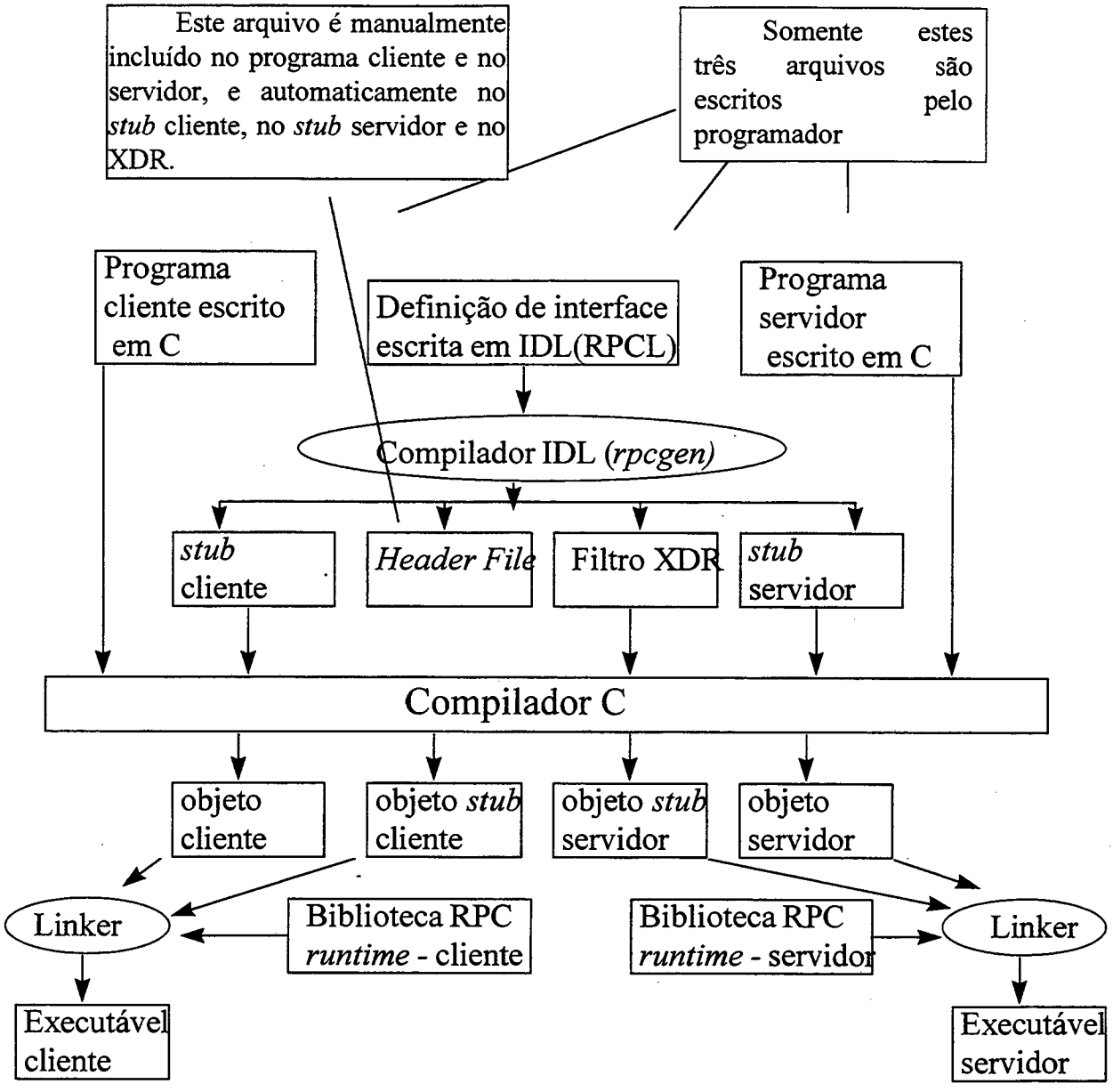


Figura 5 . Passos para criação de aplicações utilizando RPC.

Na abordagem dinâmica, o cliente usa a interface de invocação dinâmica DII (figura 3). Para permitir invocação dinâmica, interfaces de objetos devem ser armazenados no repositório de interface. Uma invocação dinâmica permite ao cliente, dinamicamente, construir e invocar métodos no servidor. Esta abordagem oferece uma grande flexibilidade para fazer com que novos tipos de objetos sejam adicionados ao sistema distribuído, em

tempo de execução. O cliente especifica o objeto a ser invocado, o método a ser executado e o conjunto de parâmetros desse método, através de uma chamada, ou de uma seqüência de chamadas. Na figura 6 são apresentados os passos para uma invocação dinâmica.

O primeiro passo, numa invocação dinâmica, é obter a localização e descrição do objeto dentro do repositório de interface. Após localizar o objeto, é ativada a função *describe* para obtenção de toda a definição de IDL do objeto. Uma vez obtido sucesso, é ativada a função *create_list* para criar a lista de argumentos e com várias chamadas *add_arg* para adicionar os argumentos à lista. No passo 3, é feito a criação da requisição, através da função *create_request*, onde é especificada a referência do objeto, o nome do método e a lista de parâmetros requerida para este método. Finalmente, a requisição pode ser feita de forma síncrona; usando RPC, assíncrona usando *send/receive* ou datagrama (chamada *one way*).

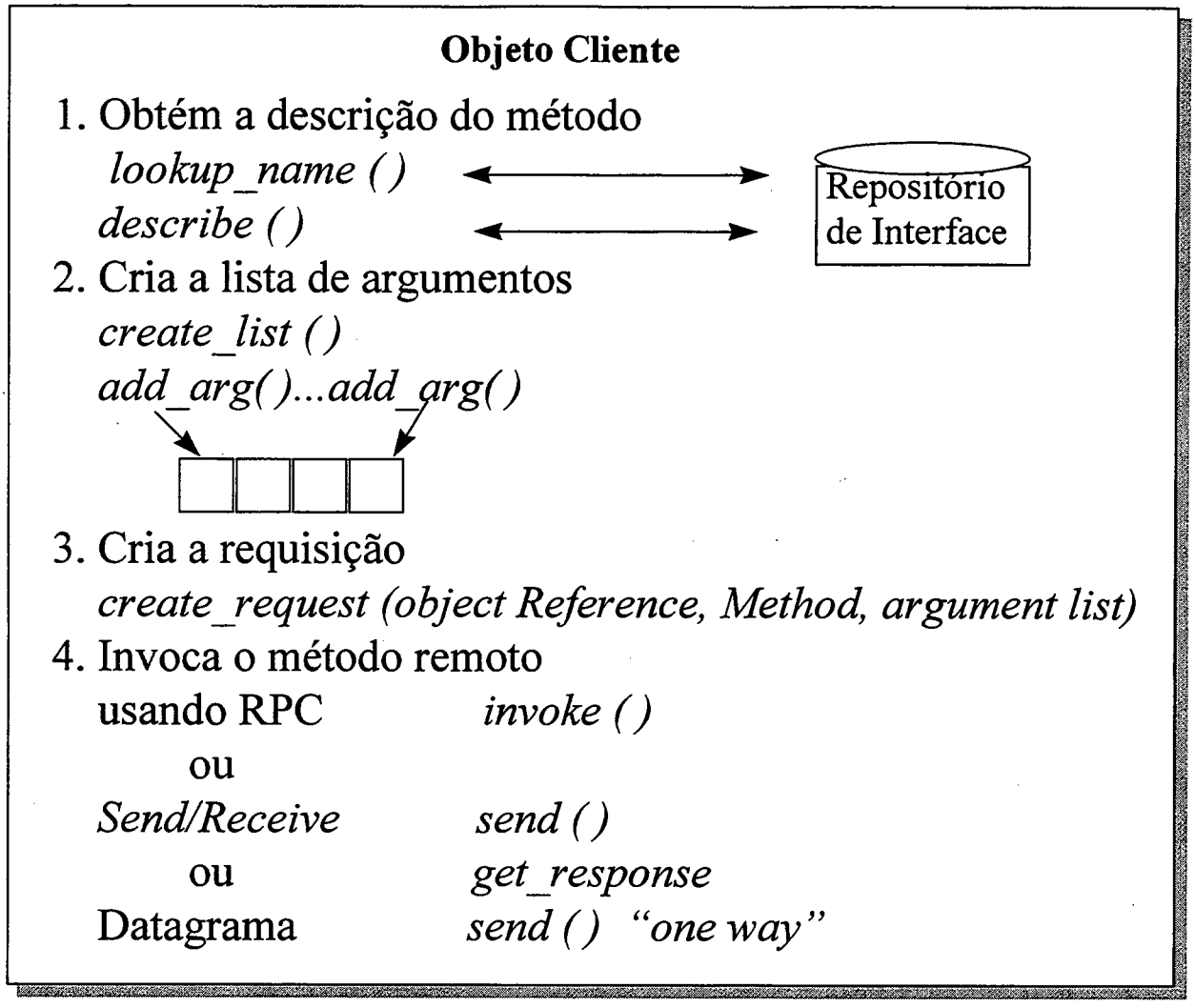


Figura 6. Invocação dinâmica no CORBA

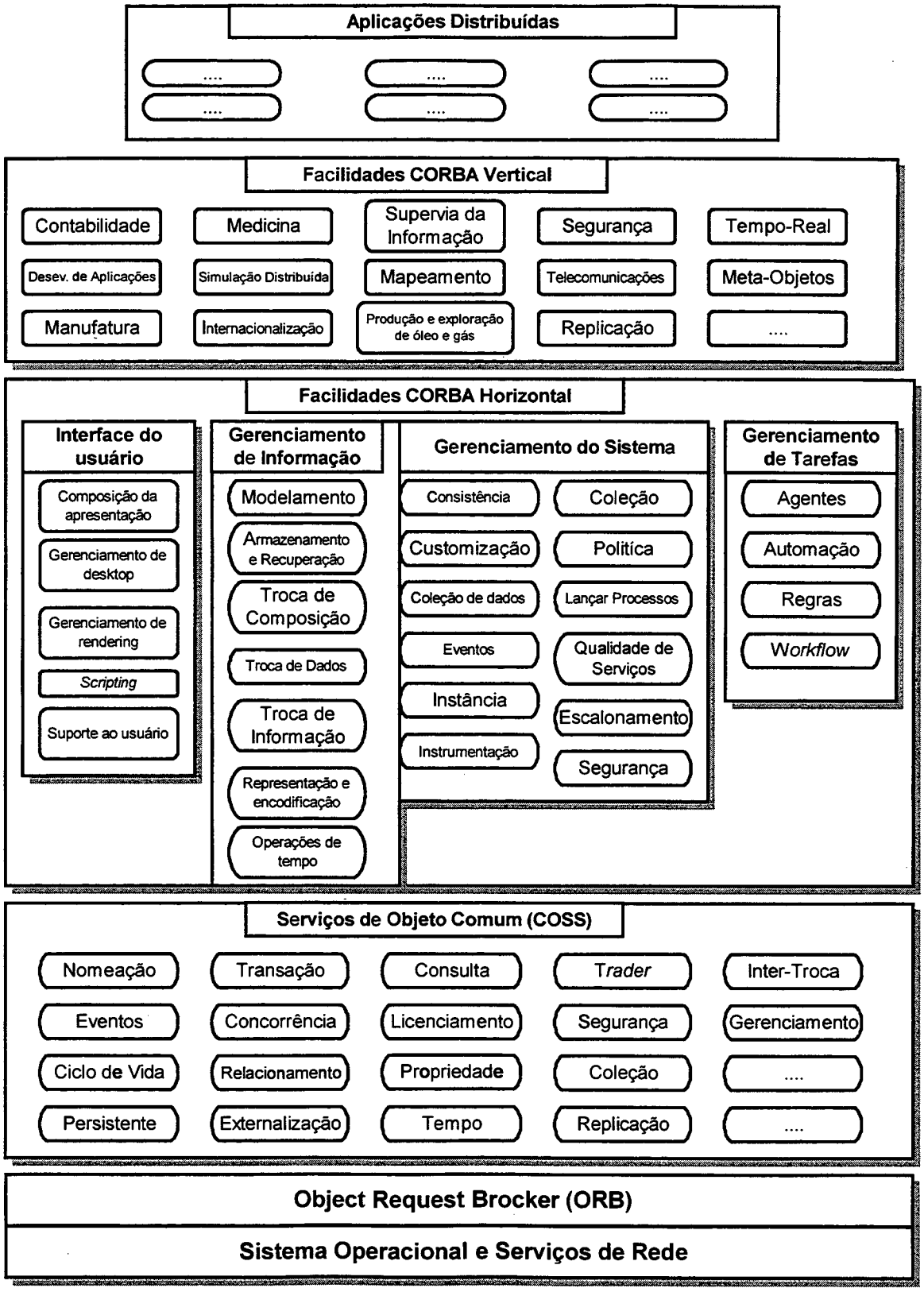


Figura 7: Arquitetura OMA (Object Management Architecture)

No desenvolvimento de aplicações orientada a objetos, o padrão CORBA/OMG oferece um pacote de serviços de objetos que facilitam o trabalho do projetista da aplicação. Esses serviços são padronizados pela OMG dentro da arquitetura OMA (*Object Management Architeture*), como apresentado na figura 7. Os serviços de objeto COSS (*Common Object Services Specifications)* formam uma coleção de serviços, a nível de sistema (interfaces e objetos), que oferecem funções básicas para utilizar e implementar os objetos. O COSS pode ser entendido ou como uma extensão ou uma complementação das funcionalidades do ORB. A OMG tem definido até agora vários serviços nesta camada, tais como:

- **Ciclo de vida**

Define serviços e convenções para criar, deletar, copiar e mover objetos. Devido aos ambientes baseados no CORBA, este serviço define serviços e convenções que permitem ao cliente realizar operações nos objetos em diferentes ligações.

- **Persistência**

Define uma interface única para o armazenamento persistente do estado dos objetos nos diversos servidores de armazenamento, incluíndo objetos base de dados (ODBMSs), base de dados relacionais (RDBMSs) e arquivos simples. O estado do objeto pode ser considerado de duas partes: o *estado dinâmico*, que está tipicamente na memória e não fazendo parte do ciclo de vida do objeto como um todo (ex.: não seria preservado em um evento de falha do sistema); o *estado persistente*, cujo objeto usaria para construir o estado dinâmico.

- **Nome**

Define o suporte, a implantação e administração distribuída e heterogênea de nomes bem como seus contextos. Permite que componentes do ORB localizam outros componentes por nome; também é definido suporte de nomeação de contextos federativo - (*federated naming contexts*). Uma associação "nome para objeto" é chamada *name binding.* Um *name binding* já define o *naming context.* Um *naming context* é um objeto que contém um conjunto de *name binding*, no qual cada nome é único. Diferentes nomes podem ser limitados a um objeto do mesmo, ou diferente contexto, ao mesmo tempo.

• **Notificação de eventos**

Define a forma como o objetos devem registrar, dinamicamente, seus interesses em eventos específicos. O serviço define um objeto chamado **canal de eventos,** que coleta e distribui eventos entre componentes do ORB. Esse serviço desacopla as comunicações entre objetos cliente e servidor. O serviço define duas funções: a do fornecedor do evento e a do consumidor do evento. O fornecedor produz o evento e o consumidor processa o evento. Os eventos são transmitidos entre o fornecedor e o consumidor, através de requisições especificadas, segundo o CORBA. São definidas duas abordagens para este serviço: o modelo *push* e o modelo *pull*. O modelo *push* permite ao fornecedor do evento iniciar a transferência do evento ao consumidor. O modelo *pull* permite ao consumidor de eventos requisitar o evento ao fornecedor do evento.

• **Controle de concorrência**

Define um gerenciador de transações e *threads* para garantir a consistência e a coerência do objeto. É utilizado, por exemplo, em aplicações que necessitam de um suporte a *multi-threads*, para serializar o acesso aos recursos compartilhados.

• **Transação**

Define a coordenação entre objetos recuperáveis, baseada no protocolo de efetivação em duas fases (aninhadas e agrupadas), assegurando as propriedades de atomicidade, consistência, isolação e durabilidade.

⇒ Atomicidade: assegura que uma transação é completamente feita ou não. Uma transação, cujo trabalho é completamente feito, é dito em acordo (*commit*); quando não, é dito *rollback*.

⇒ Consistência: éo efeito de uma transação preservar suas propriedades.

⇒ Isolamento: transações podem ser executadas concorrentemente, mas o resultado seria o mesmo, se as transações fossem executadas serialmente. Isolamento assegura que transações executando concorrentemente não podem perceber inconsistências.

⇒ Durabilidade: se uma transação for completada com sucesso, o resultado de suas operações nunca serão perdidos, exceto em um evento de catástrofe. Sistemas podem ser projetados para reduzir o risco de catástrofes (falhas).

- **Relacionamento**

Define uma forma de criar associações dinâmicas (ou *links*) entre os objetos, e os mecanismos necessários para percorrê-los, em um grupo de objetos. O serviço define operações para criar, excluir, navegar e gerenciar relações entre objetos. O serviço de relacionamento permite que entidades e relacionamento sejam explicitamente representados. Entidades são representadas como objetos CORBA. O serviço define duas classes de objetos: relacionamento e função. A função representa um objeto CORBA em um relacionamento. Um relacionamento é criado, passando um conjunto de funções para um *relationship factory.*

- **Externalização**

Define uma forma padrão para a externalização e internalização de objetos (" *stream* de dados"). A especificação define protocolos e convenções para externalização e internalização de objetos. Externalizar um objeto é registrar o estado do objeto em uma *stream* de dados. Objetos que suportam a interface apropriada para externalização e cuja implementação adere convenções próprias, podem ser externalizados para um *stream* (na memória, num arquivo em disco, sobre a rede, etc.) e subseqüentemente, ser internalizada dentro de um novo objeto num mesmo ou diferente processo. Em relação ao serviço de ciclo de vida, o serviço de externalização separa a operação mover e também a operação copiar em dois passos. Isto permite que algum processamento diferente possa ser feito entre estes passos.

- **Segurança**

Define serviços de segurança para autenticação, autorização, auditoria e não-repúdio.

Autenticação permite que cliente e servidor possam provar qual deles esta no sistema; **autorização** permite que servidores possam usar uma lista de controle de acesso para determinar quais clientes estão autorizados a utilizar seus serviços; **auditoria** permite ao gerenciador do sistema monitorar os eventos no ORB, incluindo tentativas de acesso (*logon*), em que servidores ou objetos são usados. Este serviço é fundamental para gerenciadores de sistema detectarem intrusos em suas organizações; **não-repúdio** é um serviço que utiliza mecanismos de criptografia com *checksum* (assegurando que o conteúdo da informação recebida em nada foi alterada, desde a transmissão.) Protege as informações

confidenciais, através de mecanismos, impedindo que sejam acessadas durante a transmissão entre cliente e servidor. Com isso, o serviço de segurança deve garantir os seguintes aspectos:

⇒ Confiabilidade: informações só podem ser acessadas por usuários autorizados;

⇒ Integridade: informações só são modificadas por usuários que tiverem autorização;

⇒ Responsabilidade: usuários são responsáveis pela segurança das ações relevantes;

⇒ Disponibilidade: uso do sistema não pode ser maliciosamente negado a usuários autorizados.

• **Temporização**

Define um mecanismo para sincronização de relógios em um ambiente heterogêneo. Este serviço é importante para a ordenação dos eventos que ocorrem no sistema. Pode ser usado também para gerar eventos baseados no tempo, através de temporização e alarmes, ou para computar o intervalo entre eventos. Este serviço segue o padrão UTC (*Universal Time Coordinated*).

• **Consulta (*Query*)**

Define operações de consulta para objetos. É um super conjunto do SQL (*Structured Query Language*), baseado nas especificações recentes SQL3 e do ODMG (*Object database Management Group's*), com as especificações OQL (*Object Query Language*). O serviço permite encontrar objetos cujos atributos reúnem o critério de procurar o que o usuário especifica.

• **Licenciamento**

Define o gerenciamento de políticas para licenças e métricas de utilização de *software*. Os fornecedores de aplicação necessitam de métodos para controlar o acesso e o uso de seus produtos. O método de controle mais comum é o serviço de licenciamento, cuja licença pode ser oferecida por mecanismos tecnológicos (por *software* e *hardware*) ou contratual. O licenciamento contratual é uma opção viável mas não oferece o mesmo nível de controle que o licenciamento técnico, em que é usado ferramentas de *software* e *hardware* para controle de licença.

• **Propriedades**

Define operações para associar dinamicamente valores (ou propriedades) a qualquer objeto (ao seu estado). A definição dos atributos dinâmicos (ou propriedades) pode ser feita em tempo de execução sem usar a IDL. Isto permite associações com objetos já existentes.

As Facilidades Comuns (figura 7) são coleções de serviços de propósitos gerais, utilizados pelas mais diversas aplicações. As facilidades comuns são divididas em dois aspectos: as facilidades comuns horizontais e as facilidades comuns verticais. **Facilidades horizontais** são as utilizadas por várias aplicações, independente da área da aplicação, e são divididas segundo quatro categorias: interface do usuário; gerenciamento de sistema; gerenciamento de informação e gerenciamento de tarefa. As **facilidades verticais** são utilizadas em áreas de aplicações específicas, a exemplo de imagens; supervias de informação; manufatura integrada por computador; medicina; contabilidade; simulação distribuída, etc..

Nas especificações CORBA 1.1, não estavam claras as formas de garantir a interoperabilidade entre objetos pertencentes a diferentes implementações de ORBs. A política da OMG é a padronização do CORBA. Seus serviços, na maioria das vezes, apenas a nível de interface, deixada em aberto aos desenvolvedores de *software* a parte de implementação. Devido a isto, fica difícil garantir uma interoperabilidade entre objetos de diferentes implementações de ORB, sem uma padronização do protocolo de comunicação e do formato das mensagens. Para suprir esta necessidade, a OMG lançou o padrão CORBA 2.0. Nesta especificação estão definidos os seguintes protocolos (figura 8):

- Protocolo Inter-ORB Geral (GIOP): especifica um conjunto de formatos das mensagens e dados para a comunicação entre ORBs;
- Protocolo Inter-ORB Internet (IIOP): especifica como mensagens GIOP e são transmitidas numa rede TCP/IP (GIOP + TCP/IP = IIOP);
- Protocolo Inter-ORB para Ambiente Específico (ESIOPs): é uma especificação para permitir a interoperabilidade do ORB com outros ambientes (ex: DCE, DCOM).

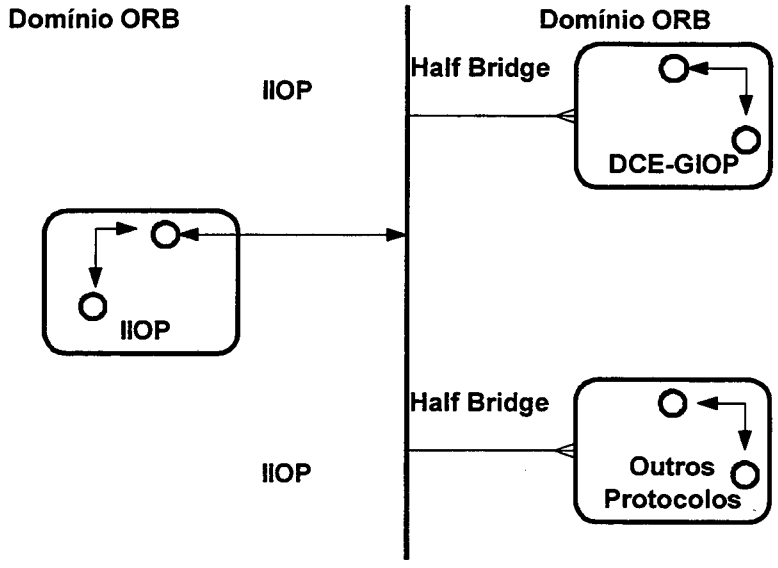


Figura 8. Interoperabilidade entre ORB

O *Half Bridge* é um mecanismo de mapeamento. Ele transforma uma requisição expressa em termos do modelo de domínio origem, em modelo do domínio destino. Isto é, converte o formato das mensagens do domínio não CORBA, para o domínio CORBA (GIOP), desde que possam ser transmitidas através do Protocolo IIOP. Foi acrescentada na versão CORBA 2.0, a interface *skeleton* dinâmico (DSI), que é utilizada para interconexão de ORBs distintos. Quando o BOA recebe invocação para implementação de objeto que não tem *Skeleton* pré-compilado, esta invocação é passada para a interface do *skeleton* dinâmico. O *skeleton* dinâmico então ativa a DII de outro ORB, em que será feita a invocação do objeto que possua *skeleton* - *Bridge Request-Level* (figura 9). Outra forma de interoperabilidade é através da aboradagem *Bridge In_Line*, na qual a *Bridge* é construída dentro do núcleo do ORB.

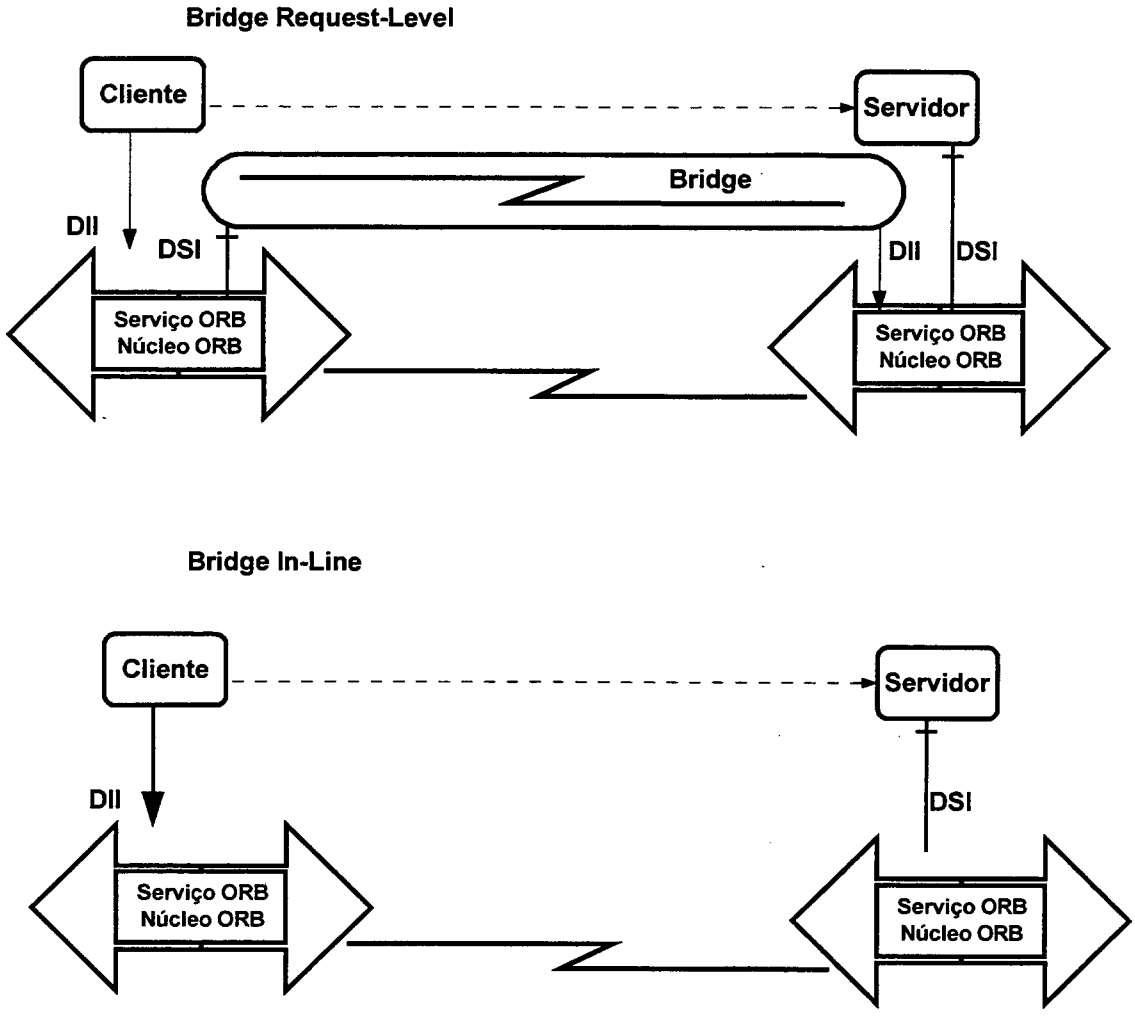


Figura 9. Duas formas para interoperabilidade entre ORBs

2.2.3.4. Java/RMI

O projeto começou há seis anos atrás, voltado não só para computadores mas, também, para produtos eletrônicos de grande consumo, como televisores, videocassete e alarmes para carro. James Gosling e sua equipe tentavam construir programas interativos para esses produtos mas sempre esbarravam em ferramentas que não funcionavam direito. No acerto dessas ferramentas, nasceu Java. O Java, não. Na verdade, nasceu uma linguagem batizada de Oak (carvalho, em português). Era essa a árvore que ficava junto a janela do escritório de Gosling. Mas não foi possível registrar a marca Oak. Mas tanta gente havia

feito isto antes da Sun, que a companhia jogou a toalha e adotou o nome Java, (café da Indonésia). Foi uma homenagem às xícaras da bebida que Gosling e sua equipe haviam virado, uma após a outra, enquanto quebravam a cabeça no projeto. Segundo Gosling, o mais difícil era achar uma aplicação prática para o Java. "Foi quase no desespero que posicionamos para a Web", diz Gosling.

Antes de falarmos de Java, tecnicamente, veremos a sua importância na História da Internet.

Há apenas alguns anos, parece que quase ninguém ouvira falar da Internet. Certamente ninguém, fora das Universidades e Instituições Governamentais, prestava qualquer atenção a ela. Muitos de nós, que fazíamos parte da comunidade Internet, tínhamos uma certa satisfação por ter acesso a um fabuloso recurso, que as empresas e o cidadão médio jamais entenderiam. Daí surgiu um fabuloso programa do *National Center for SuperComputer Applications* (Centro Nacional de Aplicações de Supercomputadores) : o NCSA Mosaic. Dá noite para o dia, a Internet foi inundada por conversas sobre a tal da *World Wide Web* (WWW), apelidada simplesmente, *Web ou WWW.* É que o Mosaic estava se tornando popular. Então algo ainda mais fantástico aconteceu: o entusiasmo que a WWW gerou começou a se espalhar além dos murros das Universidades e das Instituições de pesquisa - o mundo dos negócios começou a percebê-la.

Referências à Internet começaram a aparecer regularmente, nos grandes jornais e revistas de todo o mundo. A Internet - e particularmente a Web - chamaram a atenção da cultura popular e as elites começaram a percebê-la. A recém descoberta facilidade de navegação na Internet para ter texto, ver gráficos, ouvir "clipes" de som e assistir animações, estava levando milhões de pessoas a explora-la. Só que as pessoas queriam mais. Esta facilidade de navegação era interessante mas as pessoas queriam interatividade. Logo as pessoas estavam usando mapas de imagens clicáveis e formulários da Web para criar aplicativos reais através da Internet. Os novos cidadãos da Web começaram a ver novas formas de resolver problemas, e até ganhar dinheiro, com a Web.

O problema era que desenvolver aplicativos para a Web fora difícil. Os primeiros programas, com base em formulários, não tinham (e nem podiam ter) os recursos e respostas interativas que as pessoas passaram a esperar dos aplicativos modernos, gerando

frustração tanto a desenvolvedores quanto a usuários de aplicativos Web (figura 10). Se houvesse uma forma de tornar os programas da Web mais interativos sem se abandonar as vantagens de se estenderem a usuários de PC`s, *Macintoshes*, máquinas *Unix* e todos os outros, com um mesmo programa....Foi aí que Java entrou em cena. Java tinha sido desenvolvida na premissa de aplicativos desenvolvidos em CGI, na Web, contudo oferecendo muito mais: prometendo eliminar muitas das frustrações dos primeiros programadores para Web; acrescentar animação a aplicativos Web; e rodar programas independente da plataforma de *hardware*, usando o mesmo código. Além disso, Java parecia-se com C/C++. Os milhares de programadores C/C++, que habitam a Internet descobriram que eles "pegavam" os conceitos básicos da linguagem rapidamente. Eles estavam prontos para escrever *applets* e aplicativos em Java.

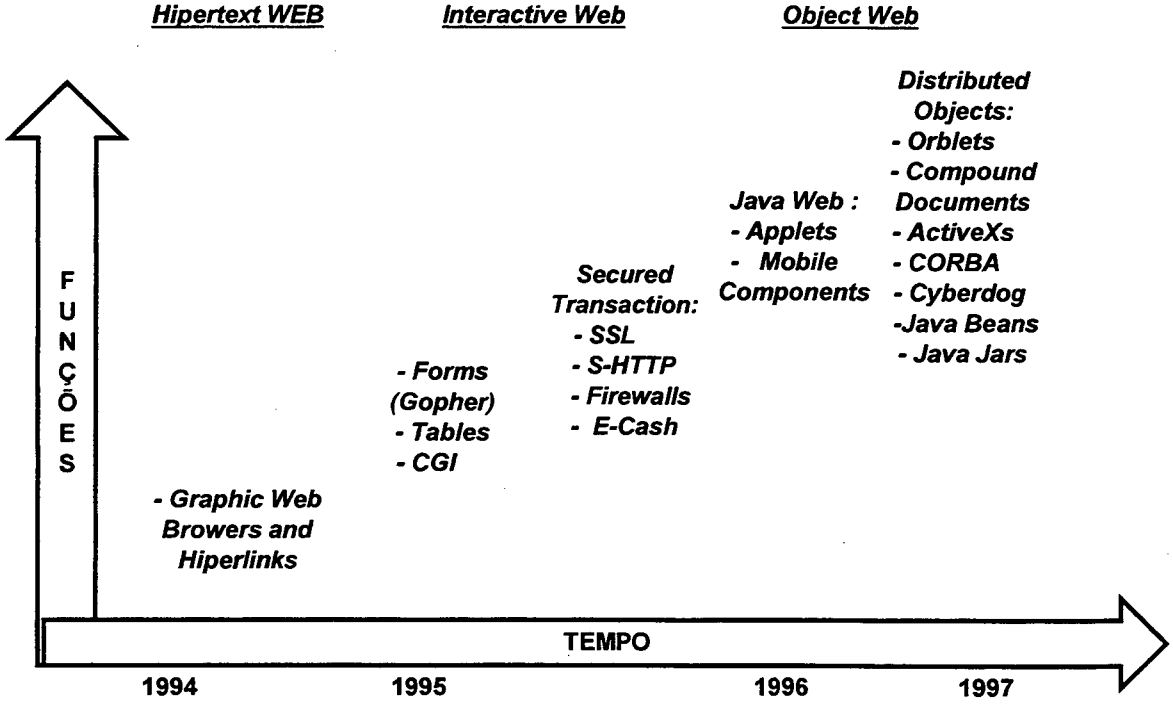


Figura 10. Evolução da Tecnologia Web

Após a criação da linguagem orientada a objetos Java, a *Sun Microsystems* criou o grupo *JavaSoft*. O objetivo atual deste grupo é explorar esta linguagem de programação para computação confiável sobre rede de grande escala, tal como, a *Internet*, e oferecer produtos, serviços e recursos que, do ponto de vista do programador e do usuário, devem

estar disponíveis na própria rede. Java é uma linguagem de programação, que é caracterizada por sua portabilidade, robustez, segurança, suporte a programação distribuída e amplos recursos para o desenvolvimento de aplicações multimídia. A quase perfeita portabilidade das aplicações desenvolvidas em Java é uma das suas grandes vantagens em sistemas heterogêneos. A habilidade de se carregar *applets* Java e a sua integração com o *WEB browser* tornam-na ferramenta ideal para a *Internet* e WWW. Esta portabilidade é alcançada porque Java é uma linguagem interpretada ou seja, quando compilamos um programa escrito em Java é gerado um código intermediário e não um código de máquina, tal como ocorre na maioria das linguagens. Com isso, se garante-se a independência de plataforma de *software* e *hardware*. A execução do código Java é realizada sobre qualquer plataforma através da máquina virtual Java (JVM), que nada mais é do que o interpretador do código intermediário (*Byte-code* Java), para a plataforma em que se deseja executar a aplicação (figura 11). A partir da versão 1.1 do Java, foi lançado o compilador JIT (*Just In Time*), como alternativa de melhor desempenho na execução dos *Byte-code* Java. O compilador JIT permite gerar a partir do *Byte-code,* o código nativo da plataforma do cliente.

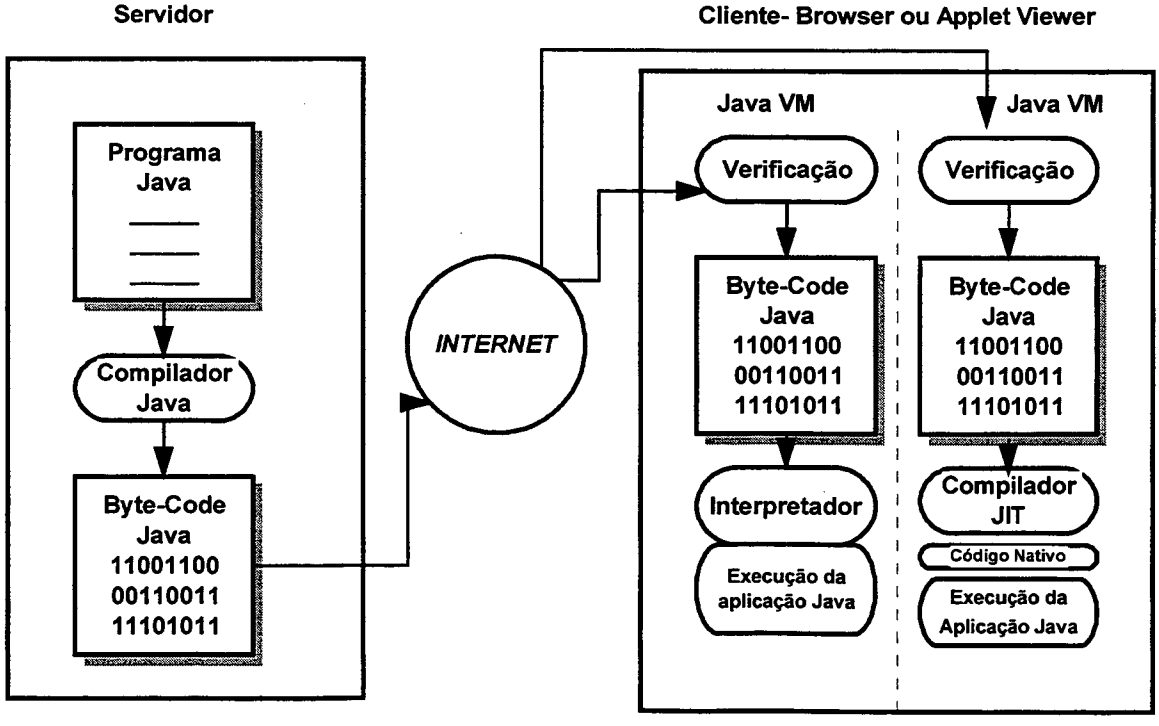


Figura 11. Execução de uma aplicação Java

Java é mais que uma linguagem de programação, pode ser tratada como um *framework* que comporta vários componentes, tais como: o picoJava, uma implementação de *hardware* da máquina virtual Java; o Java OS, uma implementação de sistema operacional; e muitas interfaces de programação das aplicações para uma ampla faixa de aplicações, são as APIs Java. Além do núcleo API Java, que consiste da linguagem básica e o sistema de janelas. Java oferece várias outras APIs que facilitam o desenvolvimento de aplicações, tais como:

- **Java *Enterprise*:** oferece suporte para as aplicações Java interagirem, com dados e aplicações que podem estar distribuídas em toda a empresa, obtendo o suporte de três componentes: Java IDL; Java RMI (*Remote Method Invocation*); Java JNDI (*Java Naming and Directory Interface*);

⇒ **Java IDL:** oferece um modo para objetos Java cliente e servidor interagirem com outros servidores e clientes, compatíveis com CORBA. Ele oferece um mecanismo direto pelo qual programas Java podem interagir com outros serviços em uma linguagem neutra;

⇒ **Java RMI:** é uma classe que permite o desenvolvimento de aplicações distribuídas em Java. É similar ao mecanismo de RPC (*Remote Procedure Call*) e adequado a construção de aplicações distribuídas, em ambientes homogêneos, escrita totalmente em Java;

⇒ **Java JNDI:** oferece mecanismo para os programas Java interagirem com diretórios distribuídos e com ambientes heterogêneos e ainda com serviços de nomes;

- ***Java Database Connectivity (JDBC)***: é uma classe que permite ao cliente Java interagir com banco de dados relacional compatível com M*icrosoft Open Database Connectivity* (ODBC);

- ***Java Security:*** oferece um *framework* para escrever programas Java com mecanismo de segurança, tais como, autenticação, assinatura digital e técnicas de criptografia;

- ***Java Media*:** oferece um conjunto de facilidades ao desenvolvimento de aplicações Java que usam gráficos e multimídia. O pacote contém o *Java 2D, o Java Media, Java Management, Java Collaboration, Java Telephony, Java Speech, Java Animation e Java 3D;*

- ***Java Collaboration:*** oferece uma *framework* para escrever aplicações *collaboration-aware,* bem como aplicações *collaboration-unaware* para trabalhos coorporativos. Oferece mecanismo para compartilhar quadro-negro eletrônico (blackboard), sistema de compartilhamento de aplicações, etc..

Além destes, existem várias outras APIs disponíveis na *Internet* para aplicações gerais e específicas.

O suporte ao desenvolvimento de aplicações distribuídas na primeira versão do JDK (*Java Development Kit*) era através de programação de *sockets* TCP/IP, bem rudimentar. Recentemente, a JavaSoft apresentou seu próprio suporte para sistemas distribuídos, denominado **RMI** (*Remote Method Invocation*) [SUN 96]. Este suporte pode ser tratado como um ORB nativo Java, isto é, RMI é um ORB (não compatível ao CORBA) no sentido genérico, que suporta invocações de métodos em objetos remotos.

O RMI é uma extensão do núcleo da linguagem e dependente de muitos aspectos do Java, tais como: serialização, portabilidade, implementação de objeto carregável (*down-loadable*), definições de interface Java, entre outros. A limitação do RMI é exatamente o seu uso em ambiente heterogêneos, a exemplo da interação com objetos desenvolvidos em outras linguagens. Na figura 12 é apresentada a arquitetura **Java/RMI,** que é constituída de três camadas: a **camada de *stub/skeleton*** para cliente e servidor, respectivamente; a **camada de referência remota,** que é responsável por conduzir as semânticas da invocação (ex.: determinar se o servidor é um objeto simples ou replicado); abstrair os diferentes modos de referenciar os objetos (ex.: servidores que já estão executando em algumas máquinas, ou servidores que só executam quando são invocados); **camada de transporte** que trata da configuração da conexão, gerenciamento da conexão e de manter ligação do *dispaching* com os objetos remotos (o alvo da chamada remota), residentes no espaço de endereçamento. O RMI também pode ser usado da mesma forma que o já conhecido RPC, sem uso de *browsers ou appletviewer.*

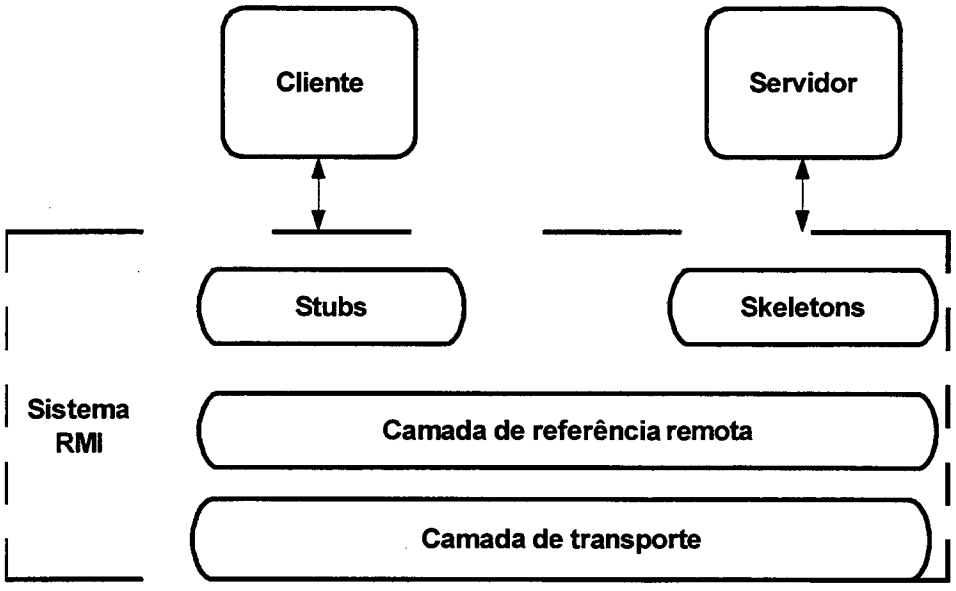


Figura 12. Arquitetura Java/RMI.

Na figura 13, é mostrada outra forma de utilizar RMI. O cliente necessitando de serviço de servidor remoto, faz um *download* da *stub* do cliente, em *bytecode*, do lado do servidor, via HTTP (*Hiper Text Transfer Protocol).* Este *bytecode* é executado dentro de uma máquina virtual Java cliente, através de um dos dois métodos explicado anteriormente. De posse da *stub,* o cliente faz uma chamada RMI ao servidor para acessar os serviços desejados, descartando o protocolo HTTP.

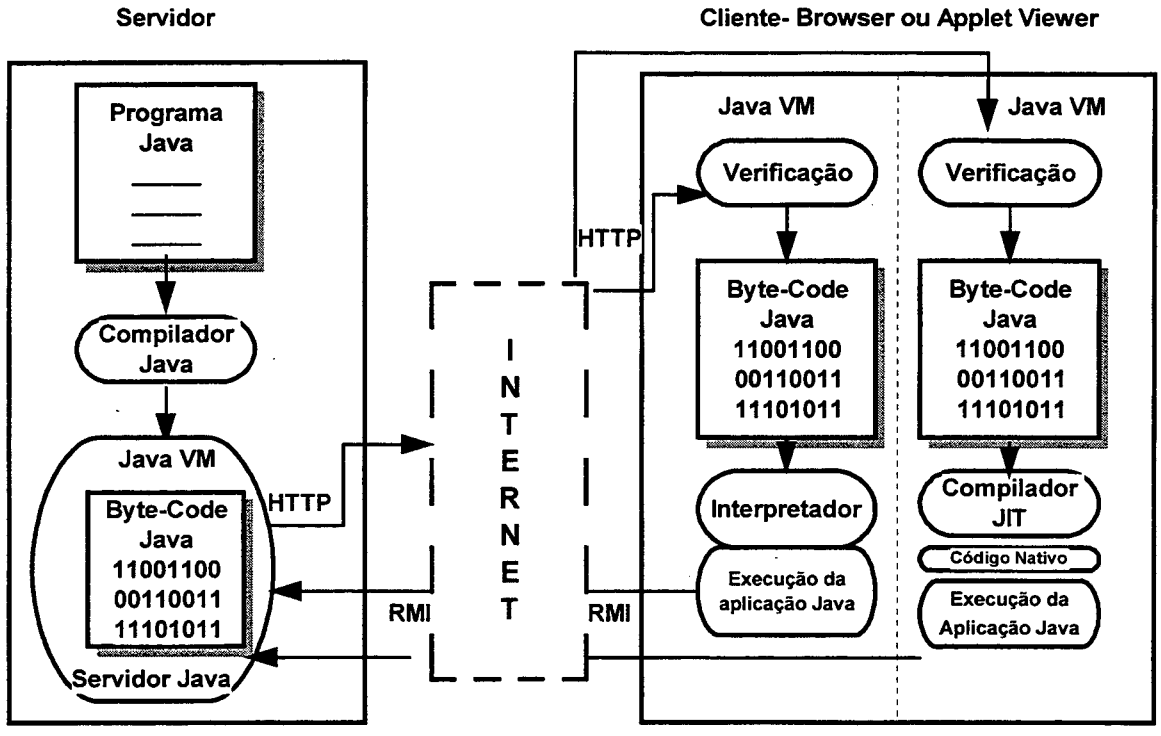


Figura 13. Execução de uma aplicação Java/RMI.

2.2.3.5. OLE/COM

O OLE/COM (*Object Link and Embedding/Component Object Model)* formam uma estrutura orientada a objetos distribuídos, desenvolvida pela *Microsoft Corporation* [KRA96].

Um sistema operacional oferece um conjunto de serviços básicos, a fim de que o projetista possa utilizá-los no desenvolvimento de sua aplicação. OLE/COM oferece habilidade para estender o sistema aos novos serviços, sem a necessidade de alterar o sistema operacional em si. OLE pode ser descrito como uma tecnologia de sistema de objetos extensível, cuja arquitetura pode acomodar desenvolvimentos existentes e novos. A **tecnologia de sistemas orientados a objeto** tem por finalidade permitir o desenvolvimento de novos serviços num contexto de sistema operacional, em execução, isto é, novos componentes desenvolvidos por qualquer um e em qualquer tempo, podem ser adicionados dentro do sistema em execução, possibilitando com isso, estender imediatamente os serviços oferecidos às aplicações, mesmo que essas já estejam em execução. OLE é mais do

que uma tecnologia de documento composto (*compound document*) - é um ambiente completo para componentes e não deve ser considerado um sistema separado - ele é praticamente o sistema operacional *Windows*. O *Windows* envolve uma coleção de interfaces OLE com o COM, oferecendo o barramento básico para comunicação entre objetos (ORB) e um conjunto de serviços. Para suportar essas funcionalidades, a infra-estrutura OLE apoia-se num núcleo simples, poderoso, que possuiuma arquitetura extensível chamada COM [ORF96]. Esse núcleo é equivalente a um ORB mas não segue o padrão CORBA/OMG. O COM é similar ao RPC do padrão DCE, só que em uma abordagem orientada a objetos e com algumas extensões proprietárias. Na figura 14, são apresentadas as tecnologias constituintes da arquitetura OLE/COM. OS quatro módulos acima do barramento de comunicação COM formam a infra-estrutura do OLE para controle e gerenciamento de documentos compostos.

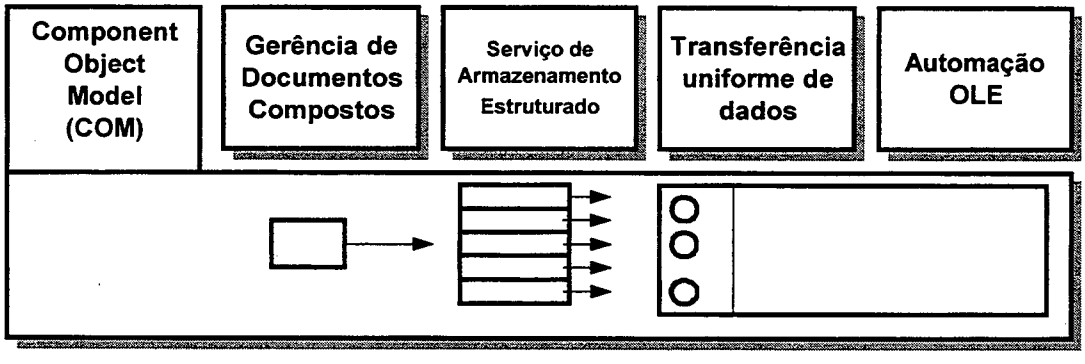


Figura 14. Tecnologias constituintes do OLE/COM.

O módulo Gerência de Documentos Compostos (GDC) faz o tratamento das interações entre objetos componentes *containers* e *servers*. O primeiro oferece o lugar e o segundo, as coisas (os objetos texto, figuras, gráficos, etc.) que residirão nesse lugar. Em outras palavras, o *container* é o componente que gerencia o documento, que trata do *layout* visual e gerencia as relações entre servidores que representam o conteúdo do documento, deste modo o *container* é um cliente dos *servers* (Figura 15). O GDC introduz também a interface que define o *container* e as funções do *server*. As interfaces definem os protocolos entre *containers* e *servers* e também os seus diferentes tipos. A nova versão do OLE/COM introduz um novo tipo de servidor chamado de *custom control* (também

conhecido como OCX). O OCX introduz um conjunto de novos serviços de componentes tais como: objetos contáveis; conjunto de eventos; licenciamento e auto registro. Esses serviços são considerados, agora como serviços COM, onde também estão os serviços de persistência, bibliotecas de tipos e transferência uniforme de dados [ORF96].

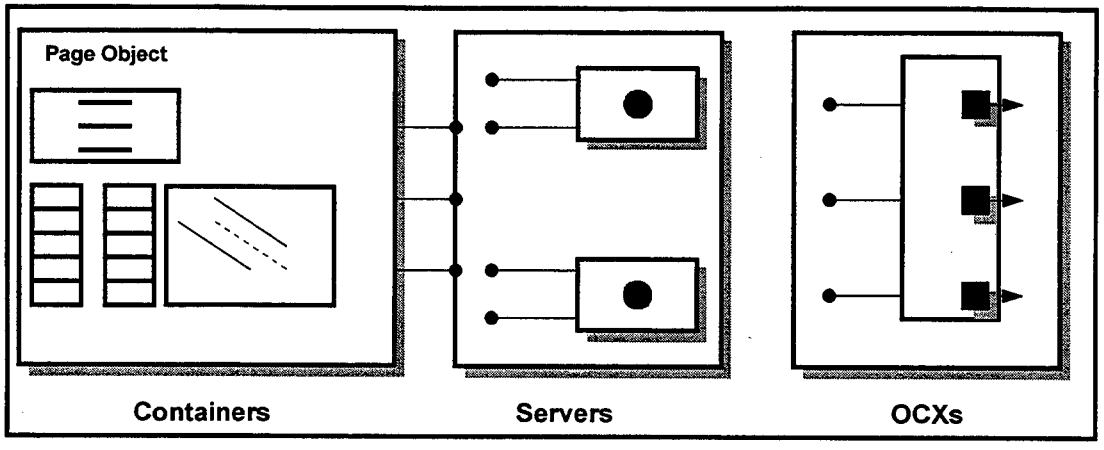


Figura 15. Gerência de Documento Composto OLE

O serviço de armazenamento estruturado (figura 14) permite que componentes desenvolvidos independentemente possam salvar seus conteúdos dentro do mesmo arquivo. Este serviço oferece mecanismo para componentes salvarem seus estados persistentes (são aqueles que permanecem existindo, mesmo após o término da execução do programa) e para os clientes procurarem estes objetos persistentes e carregá-los dentro da memória. Na versão corrente deste serviço, o OLE/COM oferece um meio compartilhado que pode ser usado para trocar e armazenar o estado persistente dos componentes. Os arquivos compostos são criados no topo do sistema de arquivo existente.

O OLE oferece um mecanismo para transferência de dados inter-componentes que pode ser usado por uma ampla faixa de situações e entre uma variedade de mídia. O serviço de transferência uniforme de dados (figura 14) permite a troca de dados, usando-se protocolos tais como: *clipboard; drag-in-drop; links* ou documentos compostos. Os dados trocados podem ser copiados, ou recortados, e então serem colados dentro do mesmo documento, em um documento diferente ou em uma aplicação diferente. Esse mecanismo pode utilizar memória compartilhada ou armazenamento em arquivos para realizar a

transferência de dados. Notificações assíncronas podem ser enviadas ao cliente quando o *link* ao dado fonte tiver sido alterado.

E por último, o serviço de automação OLE, permite aos clientes invocarem dinamicamente, métodos que manipulam o conteúdo dos objetos. Este serviço não é específico para qualquer linguagem textual (*scripting),* mas pode trabalhar com qualquer programa capaz de emitir invocação dinâmica de métodos. O serviço oferece meios para clientes descobrirem os métodos que um objeto servidor dispõe em tempo de execução. O serviço de automação usa a COM e pode ser implementado independentemente do resto do OLE. Com este serviço é possível criar ferramentas do tipo *browsers*, compiladores e linguagens textuais (*scripting*) que acessem e manipulem os objetos automação.

O modelo de objetos distribuídos COM especifica as interfaces entre objetos componentes dentro de uma simples aplicação, ou entre aplicações e também a separação da interface, da implementação. Oferece APIs para descobrir, dinamicamente, as interfaces de um objeto e como fazer para invoca-los. O COM utiliza também o conceito de IDL. Neste caso a *Microsoft* dispõe de duas linguagens de definição de interface proprietária: a IDL e a ODL (*Object Definition Language*). O IDL *Microsoft* é baseado no DCE (incompatível ao CORBA e ao próprio DCE); o ODL é um super conjunto IDL DCE, que descreve as interfaces para as bibliotecas de tipos (*type library*). O ODL é equivalente a um repositório de interface CORBA. O modelo de objetos COM não suporta múltiplas heranças, ao invés disso pode suportar múltiplas interfaces (figura 16). O COM supre essa necessidade para garantir reuso de *software,* através da técnica conhecida como **agregação,** permitindo ao componente encapsular serviços de outros componentes. (Vale ressaltar que objetos COM não são objetos no sentido orientação a objetos). Interfaces COM não possuem estado e não podem ser instanciados para criar um objeto. Uma interface COM é simplesmente um grupo de funções relacionadas. Clientes COM obtêm um ponteiro para acessar as funções em um objeto. Este ponteiro não está relacionado ao estado da informação. O cliente COM não pode re-conectar exatamente a mesma instância do objeto com o mesmo estado em um tempo posterior. Ele só pode re-conectar a um ponteiro da interface da mesma classe. Em outras palavras, objetos COM não mantém estado [ORF96].

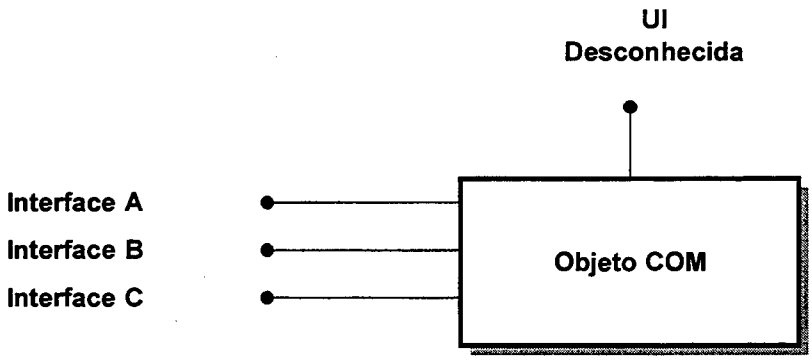


Figura 16. Um objeto COM.

2.2.3.6. OpenDoc

OpenDoc é um conjunto de APIs e *software* para documentos compostos. OpenDoc criado pela *Apple,* é similar em escopo e em funções como o OLE 2.0, e é considerado como uma alternativa ao OLE 2.0. OpenDoc é administrado e divulgado pelo Laboratório de Integração de Componentes (CI Labs) - fundado em setembro de 1993, com a união das empresas *Apple, IBM, Oracle, Novell, Taligent, Sunsoft, Word Perfect* e *Xerox.*

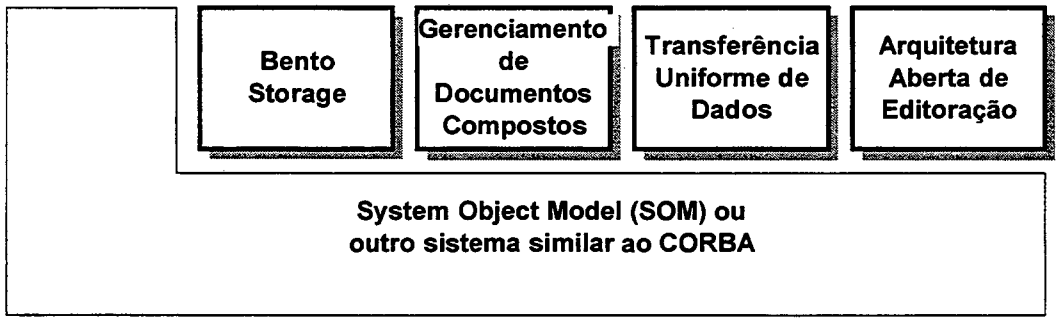


Figura 17. Mostra os componentes do OpenDoc.

Os Componentes do OpenDoc são os seguintes:

⇒ O *System Object Model* oferece a interoperabilidade do OpenDoc no mesmo espaço de endereço (objeto local), sobre o espaço do endereço (objeto remoto), na mesma máquina ou através da rede, isto é, o SOM suporta invocação estática e dinâmica semelhante ao CORBA;

⇒ O *Bento Storage* é o nome dado a compartimento de comidas japonesa. E em nosso caso, refere-se ao local onde são guardado objetos diferentes. Recipiente Bento permite que aplicações recupere e armazene conjunto de objetos.

⇒ A Transferência Uniforme de Dados oferece transferência de dados via cópia e cola, *drag e drop* e ligações.

⇒ O Gerenciamento de Documentos Compostos suporta vários tipos de documentos na construção do OpenDoc e no mesmo documento (por exemplo: um documento composto de video, som, figuras e outros documentos compostos).

⇒ A Arquitetura Aberta de Editoração é baseado no *Mac AppleScript.* Ela é definida aproximadamente por uma dúzia de comandos polimórficos (como por exemplo: *delete* que significa a exclusão de uma palavra, som, vídeo, figura, ...dependendo do que foi selecionado pelo usuário).

A CI Labs pretende disponibilizar, para a indústria a construção do código fontes da tecnologia OpenDoc. A CI Labs também tem planos para tornar o OpenDoc compatível com o OLE, sendo que um OpenDoc poderá ver um objeto OLE e vice-versa. O OpenDoc é hoje oferecido para as seguintes plataformas: *Microsoft; OS/2; Macintoch* e UNIX.

2.2.3.7. DCOM (*Distributed Component Object Model*)

Para entendermos um pouco mais o DCOM, veremos o *ActiveX.* Em março de 1996, a *Microsoft* anunciou o *ActiveX* , que é uma combinação do OLE nos serviços de desktop com a *World Wide Web.* Em outras palavras *ActiveX*, combina *Web browsers* e *applets* Java com os serviços oferecidos pelo *desktop*, como por exemplo: documento *MS-Word;* suporte as transações; planilhas; *scripting; documentos compostos* e outros serviços. Com isso, a *Microsoft* deu um grande passo para objetos distribuídos.

ActiveX usa DCOM para oferecer comunicação entre componentes remotos *ActiveX* e outros serviços. E neste sentido, o DCOM tornou-se um ORB para o *ActiveX* .

A idéia é que, no futuro, o MS-*Windows,* se torne uma grande coleção de componentes e interfaces de *ActiveX* e o DCOM serve como ORB. É esperado que todos os serviços do sistema sejam escritos como objetos DCOM.

O DCOM oferece basicamente um serviço de localização de *ActiveX, APIs* . e de objetos através de invocação estática e dinâmica. O DCOM utiliza o DCE RPC (Chamada Remota a Procedimento do DCE) para interações entre objetos. O modelo de objetos DCOM é algo limitado porque o DCOM não suporta herança múltipla. Em outras palavras, DCOM suporta herança através de ponteiros *(pointers)* que ligam várias interfaces. Além disso o DCOM oferece ainda as seguintes facilidades:

⇒ ***Interface Definition Language* (IDL)** DCOM usa interfaces que são muito parecidas com a do CORBA. Basicamente, uma interface define um conjunto de funções relacionadas.

⇒ ***Object Definition Language* (ODL) *and Type Libraries*** . O DCOM suporta uma linguagem de definição de objetos (ODL) que é usada para descrever Metadados. As especificações de interfaces e Metadados são armazenados em um repositório, que é chamado de *Type Library*. *Type Library* equivale ao Repositório de Interface CORBA.

⇒ ***Object Services.*** O DCOM oferece serviços de objetos muito rudimentares.(exemplo de serviços oferecidos: mecanismo de licença; serviço de diretório local, baseado sobre o *Registry* do *Windows;* serviços de persistência de sistema e um serviço de eventos, muito simples, chamado *connectable objects)* e, em adição, o serviço de nomeação, oferecido pela *Type Libraries.* Existe expectativa da Platform *ActiveX* suportar outros serviços, como por exemplo, serviço de diretório X.500.

O DCOM também oferece todas as facilidades que vimos no COM e, aliado ao DCE RPC, pode oferecem estas facilidades remotamente, ou seja, os objetos poderão estar em máquinas diferentes.

O DCOM é baseado na filosofia de orientação a objeto, utiliza o conceito de interface, através de IDL; a chamada do cliente para o servidor; utilizando as invocações estáticas e dinâmicas; um repositório de interfaces destinado a invocar e localizar objeto semelhante ao CORBA, chamado de *Type Library.*

O DCOM usa, em adição ao IDL, o ODL (*Object Definition Language*) para definição de Metadados (objetos). No CORBA é usado a IDL tanto para criar IDL como ODL.

O DCOM utiliza a *Universal Unique ID* (UUID), baseado em OSF DCE, para localizar e invocar objetos. No CORBA, não é utilizado este mecanismo e sim objetos de referência e repositório para localizar e invocar objetos.

O DCOM usa o OSF DCE RPC como mecanismo de transporte básico entre objetos remotos. O CORBA usa várias opções tais como: IIOP(*Internet Inter-ORB Protocol)*, o qual usa *sockets* TCP/IP e ESIOP (*Environment Specific Inter-ORB Protocol*) que é executado sobre o DCE. Além do CORBA usar somente conexões baseada em TCP (*Transmition Control Protocol)*, enquanto o DCOM só suporta UDP (*User Datagram Protocol*).

Vale ressaltar que o DCOM é uma tecnologia proprietária Microsoft e que somente está disponível para os sistemas operacionais *Windows NT e Windows 95, (*o *Windows 95* fica prejudicado por falta de alguns serviços disponíveis para o Windows *NT.*). Isto significa dizer que a idéia de sistema heterogêneo fica prejudicada, e que a interoperabilidade entre máquinas existirá, mas somente se estiver utilizado um dos sistemas operacionais acima mencionados.

2.2.3.8. Considerações e conclusão

O Objetivo deste capítulo é tecer algumas considerações entre as tecnologias apresentadas, com ênfase para a comparação do CORBA com as outras arquiteturas. Dentre todas as tecnologias de objetos, apresentada neste capítulo, duas merecem maior destaque: o CORBA, pelo seu poderoso suporte a sistema heterogêneo; o JAVA pela sua grande portabilidade jamais alcançada em outra linguagem de programação. Todavia vale ressaltar, também, que CORBA é uma tecnologia de integração e que permite interoperabilidade de diferentes plataformas de máquina, de sistema operacional e de

linguagem de programação. Java é uma tecnologia de programação concreta, entretanto não oferece interoperabilidade para outros ambientes não-Java.

OLE/COM é uma tecnologia nova e, ainda, com muitas deficiências a serem sanadas. Isso se deve a sua origem voltada para as aplicações de edição de documentos. Além disso, OLE/COM é uma tecnologia proprietária o que dificulta os aspectos de sistemas abertos. Existe um trabalho em andamento, dentro da OMG, no sentido de integração OLE/COM com CORBA.

OpenDoc é uma tecnologia semelhante ao OLE/COM, contudo com muitas deficiências. O grupo CI Labs está preocupado com a integração do OpenDoc ao OLE/COM, embora, para o momento, não tenhamos material suficiente para afirmar que o grupo CI Labs esteja preocupados com a integração do OpenDoc a ambientes de computação distribuída, por exemplo, o CORBA.

DCOM é uma tecnologia proprietária desenvolvida pela *Microsoft* para objetos distribuídos, semelhante em alguns pontos ao CORBA da OMG, sendo que não permite interoperabilidade de diferentes plataformas de máquina e de sistema operacional, dificultando a sua aceitação e uso em ambientes não-*Microsoft*.

O ANSA serviu como uma primeira experiência em termos de padronização, em tecnologias de objetos distribuídos. Muitos dos conceitos de modelos adotados no ANSA tiveram reflexo nas especificações da OMG. Atualmente, o consórcio que especifica o sistema ANSA vem trabalhando no sentido de migrar suas tecnologias para torná-la compatível com o CORBA.

Em Relação ao DCE, a tentativa da OSF de incluir, na sua arquitetura suporte para objetos distribuídos [DIL93], não vingou, possivelmente, devido a popularidade alcançada pelo CORBA. No entanto, na OMG, já se encontra disponível a padronização CORBA 2.0, que permite a interoperabilidade do CORBA com outros ORBs, incluindo DCE. Como se pode notar, esses casos só enfatizam o CORBA como uma tecnologia de integração [CUR97] em todos os sentidos (*hardware e software*). Na figura 18, é mostrada algumas das possibilidades da integração CORBA.

A integração CORBA e Java/RMI

Ultimamente, têm surgido diversas propostas de padronização e implementações, principalmente dentro da OMG, no sentido de integrar CORBA e Java, trazendo a vantagens de ambas as tecnologias, principalmente para serem utilizadas em WWW. Esta integração promete ser a grande sacada nas tecnologias de objetos distribuídos deste final de século. Na OMG, existem dois grupos trabalhando no sentido de especificar esta integração. A primeira seria no sentido do mapeamento da IDL para Java, permitindo a geração do *stubs* e do *skeletons* escritos em Java [OMG96] e seguindo a mesma idéia do mapeamento IDL, para outras linguagens de programação já padronizadas pela OMG (C, C++, COBOL, Ada, Smalltalk, etc). A segunda é uma idéia não menos interessante, e consiste no mapeamento de Java para IDL [OMG97]. Neste caso, o programador teria uma forma de construir aplicações distribuídas diretamente em Java , que interagiria com o resto do ambiente heterogêneo, via IIOP. Pela geração de IDL, a partir do código Java, muitas linguagens de programação teriam acesso a esses componentes escritos em Java, através do IIOP. A própria *Sun* tem feito estudos de viabilidade para o Java/RMI adotar o protocolo IIOP e tudo indica que isso é bastante provável [CUR97].

HTTP e IIOP

Usando o protocolo IIOP como padrão torna-se possível descartar o uso do protocolo HTTP (*HiperText Transfer Protocol*), que é considerado uma tecnologia de baixo desempenho (figura 18). Com o HTTP, para cada requisição, é iniciada uma conexão e terminada quando os dados são recebidos, isto é, para cada requisição é necessário um processo de abertura de conexão. Já com o IIOP, o cliente abre uma conexão TCP/IP com o servidor e envia uma ou mais requisições IIOP ao servidor, na mesma conexão. Não há dúvida de que IIOP tornar-se-a padrão de fato. Para se ter uma idéia, a *Netscape* passa adotá-lo a partir da versão 4.0, do seu *Browser,* e nos seus outros produtos.

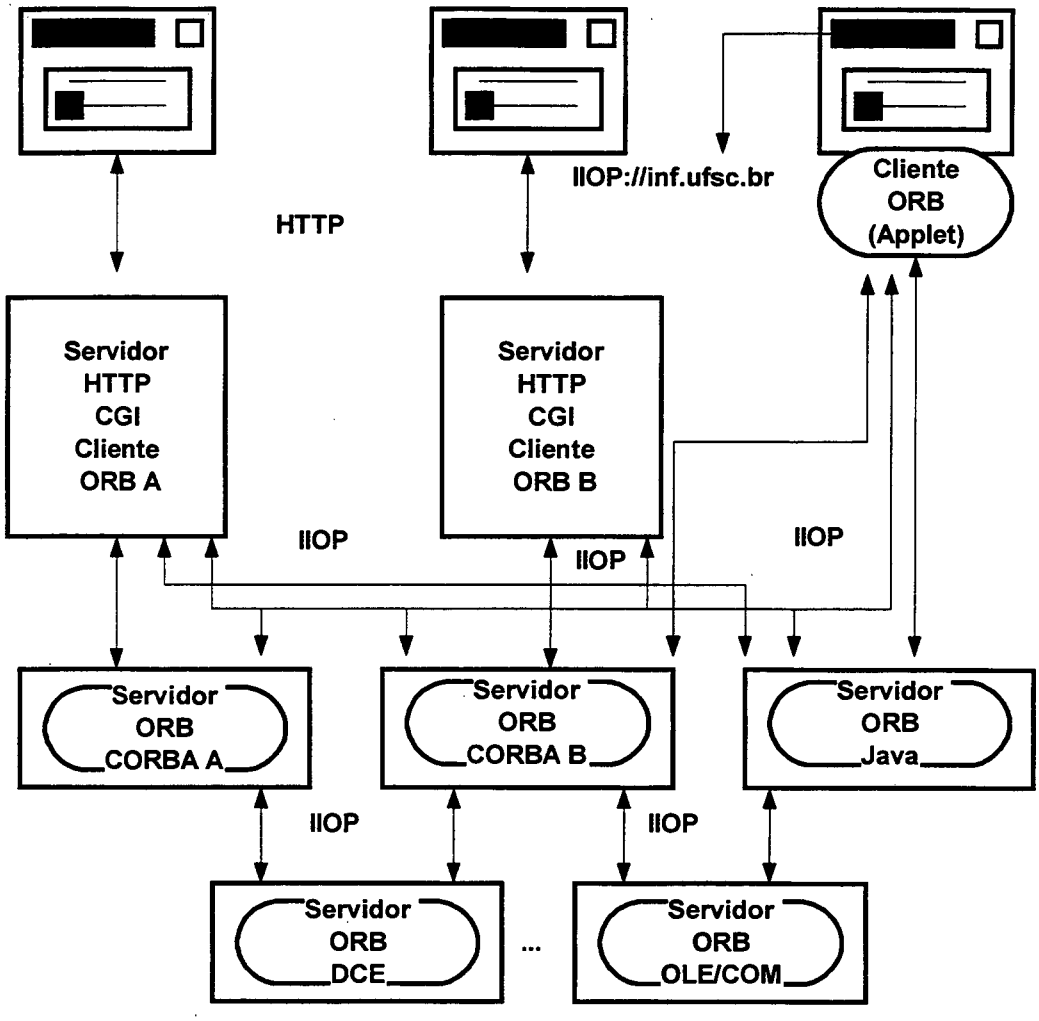


Figura 18. CORBA Tecnologia de Integração

Este capítulo tentou dar uma visão geral das principais tecnologias de objetos distribuídos do mercado, apresentando sua principais características e, como fazem para atender a uma gama de requisitos para programação orientada a objetos em ambientes distribuídos e em sistemas abertos. A grande tendência do mercado das tecnologias de objetos distribuídos, é o domínio da *Internet.* Quem conseguir impor seu padrão certamente dominará o mercado. Então, a necessidade de se ter um padrão aberto deve ser considerada como requisito primordial.

2.3. GERENCIAMENTO DE SERVIÇOS EM REDES DE TELECOMUNICAÇÕES

2.3.1. Introdução

Até recentemente, as prioridades do sistema TELEBRÁS foram a expansão e a modernização da rede de telecomunicações com a finalidade de atender à demanda reprimida de serviços de telecomunicações [BRI93].

Serviços, o que seria? Tentamos conceitua-lo como :

"um trabalho, ou conjunto de trabalhos, a ser feito e que tem por objetivo servir a uma pessoa, ou a grupo de pessoas".

Bem baseado na visão de futuro do sistema TELEBRÁS e nesta tentativa de conceituação de serviços, vemos que **Excelência em Serviços** já pode ser considerada como um dos desafios empresariais mais complexos e, ao mesmo tempo, desafiadores previsto para esta virada de século. Pode-se julgar uma certa dose de exagero no julgamento desta afirmação, porém, este conceito, é bastante coerente, considerando a situação atual e as tendências previsíveis nas relações Fornecedores-Clientes. Na seqüência temos, em contrapartida, o tempo disponível para as diversas ações no sentido das exigências do mercado consumidor. A complexidade, em se conquistar (ou perseguir) a excelência destacada na consideração inicial, tem sua justificativa frente ao notório despreparo de inúmeras (porque não dizer - a grande maioria) organizações de serviços, que insistem em tratar as relações de "conquista, fidelidade e manutenção de clientes" , como um mero resultante de preços baixos, ou de processos massificados de comercialização. Os prognósticos mais realistas nos levam a crer que o cliente, num futuro muito próximo, ocupará definitivamente a confortável posição de negociar sua condição de **consumidor fiel** somente a uma seleta classe de fornecedores, dos muitos que hoje disputam o mercado.

Os fornecedores por sua vez, ofertarão em troca:

⇒ **Qualidade no atendimento;**

⇒ **Disponibilidade de produtos e serviços;**

⇒ **Garantia pós-venda;**

⇒ **Além, obviamente, dos preços competitivos e atraentes, numa demonstração de esforço constante na manutenção do** cliente conquistado.

Se quisermos saber o motivo real de tais atitudes por parte dos fornecedores é muito simples - **a concorrência**.

O cenário em questão, previsível a curto prazo, é notoriamente diferente daquele que a história recente nos revela. Em épocas passadas, não muito distantes, clientes ficavam a mercê da boa vontade e do bom humor dos prestadores de serviços, cujo conceito, sobre esses mesmos clientes, resumia-se mais num "mal necessário" do que na responsabilidade da sobrevivência dessas empresas. Porém, há de se considerar alguns pontos que, invariavelmente, levaram, (não justificadamente) esses fornecedores ao desinteresse pelos investimentos na conquista de seus clientes.

Um motivo em destaque era o baixo nível de competitividade entre empresas prestadoras de serviços. Havia em competitividade branda que levava as organizações ao comodismo típico de quem desfrutava conscientemente de **zona de conforto.**

Outro motivo a considerar era a pouca interpelação por parte do cliente, no que tange à exigência da qualidade e outros atributos. A justificativa desta postura pairava no fato de que o grau de informação sobre produtos, a oferta de serviços, as especificações e mesmo do canal de acesso aos concorrentes eram baixos, obrigando o cliente a manter uma verdadeira "**fidelidade compulsória**", por não conhecer nem ter opção. Isto sem falar no comodismo das empresas que se auto-rotulavam de "tradicionais", "nobres", "antigas", "representante exclusiva" e que, por conta desses títulos, não tinham a menor preocupação com pesquisas de satisfação, nem com indicadores de desempenho, indicadores de qualidade, ou mesmo de desenvolver ações no sentido de aprimorar o atendimento a seus clientes. Seguramente, essa época já pode ser considerada uma página virada na história da **qualidade de serviços**. Para nossa satisfação, estamos vivenciando os primeiros passos de uma nova concepção nas relações Cliente-Fornecedor, em especial no campo de serviços. O argumento das modernas empresas é a agilidade, o empreendimento, a melhoria contínua, a luta incessante contra a obesidade administrativa e, até mesmo, o inconformismo com padrões de **qualidade** pouco desafiadores. Não haverá mais espaço para organizações que

insistam em agir em sentidos opostos às tendências da modernidade. Em suma, "empresa tradicional" não é mais sinônimo de sucesso ou de garantia de sobrevivência. [BAR96], [JUN96]

Cabe uma reflexão sobre o alerta transmitido pelo pensador Joe Cullen :

"Neste próximo milênio, as tendência indicam que permaneceram no mercado apenas dois tipos de empresa. Aquelas que investiram maciçamente em Qualidade, numa preocupação constante com a satisfação de seus clientes, e as outras que não existirão mais."

2.3.2. Órgãos de Padronização

Existem, no mundo, organizações que se destacam pela busca de padrões. Dentre esses órgãos, daremos ênfase aos dois mais importantes deles para ***tecnologia de informação***: a ISO e o ITU-T. A ISO cuida de padronizações em muitas áreas, e o ITU-T cuida, particularmente, das áreas de telecomunicações. Essas duas organizações têm sede em Genebra, na Suíça. A ISO, sendo a mais antiga teve a sua fundação em 23 de fevereiro de 1947, reconhecida já como organismo não-governamental e resultou da iniciativa de atendimento de diversos países no sentido de consensar um processo de padronização normativa, que tivesse o aceite contratual entre Fornecedores e Clientes.

De todas as séries de normas editadas, uma delas em especial - a série 9000, desenvolvida a partir de 1987 pelo TC 176 (Comitê Técnico da ISO específico para estudos normativos de sobre qualidade) nós interessa. A série ISO 9000 resulta num conjunto de normas assim denominadas : ISO 9000, ISO 9001; ISO 9002; ISO 9003; **ISO 9004.** Dentre essas, daremos maior atenção a **ISO 9004**(parte 2) - ***"Gestão da Qualidade e Elementos do Sistema da Qualidade" - Parte 2 - Diretrizes para Serviços - (1ª edição publicada em 1991).***

E, dentre às séries de normas publicadas pela ITU-T, daremos destaque as seguintes recomendações: a M-3020, que especifica metodologia de interface TMN, publicada em

outubro de 1992, e a M-3200, que especifica gerenciamento de serviços TMN, publicada em 1992.

2.3.3. Características dos Serviços de Telecomunicações

Em telecomunicações, serviços constituem uma série de recursos, oferecidos aos clientes das empresas de telecomunicações. Por exemplo, teleconferência, telecard, correio eletrônico, auxílio a lista, conta telefônica, despertador automático, etc.

Comparados aos elementos de redes, serviços de telecomunicações têm as seguintes características:

⇒ Serviços são distribuídos, já os elementos de rede não o são;

⇒ Serviços são implementados acima dos elementos de rede;

⇒ Sempre há menos serviços do que elementos de redes;

⇒ Serviços são heterogêneos, já os elementos de rede não;

⇒ Serviços são mais complicados quando comparados com elementos de rede. Em muitos casos os serviços envolvem pessoas como parte dos processos;

⇒ Serviços são dinâmicos enquanto que elementos de rede são estáticos. Novos serviços freqüentemente precisam ser criados, implantados e disponibilizados com rapidez [KON96].

2.3.4. Gerenciamento de Serviços

A administração das funções no gerenciamento de serviços têm diferentes orientações, e uma grande extensão, ou aplicabilidade, quando comparado com gerenciamento de redes.

O gerenciamento de serviço esta mais interessado em serviços relacionados com emissão, tais como:

⇒ Como fornecer um serviço ao assinante;

⇒ Quais funções precisam ser gerenciadas para que se tenha a qualidade de serviço prevista em contrato;

⇒ Como definir o modelo de informações, tal que este, preserve uma visão única e consistente para diferentes usuários de serviços;

⇒ Como é implementado e gerenciado um serviço;

⇒ Como gerenciar serviços, através de jurisdições de telecomunicações que podem ter sistemas de serviços oferecidos, políticas de preço, ambiente comercial, administrativo e estrutura tecnológica diferentes;

⇒ Como gerenciar força de trabalho e fluxo de tarefas necessárias para os serviços;

⇒ Qual a relação entre gerenciamento de serviços e de redes;

O gerenciamento de redes TMN baseada em OSI foi projetado para gerenciar um grande número de objetos simples (Figura 21). Isto pode não ser o suficiente para propiciar uma solução completa ao gerenciamento de serviços. Por exemplo, as facilidades técnicas de modelagem, de objetos usados atualmente, (como GDMO), não são suficientes para modelar os objetos e nem as funções de gerenciamento de serviços. Para que se consiga dominar rapidamente as mudanças nos serviços e a interoperabilidade, verificou-se a necessidade de novas tecnologias.

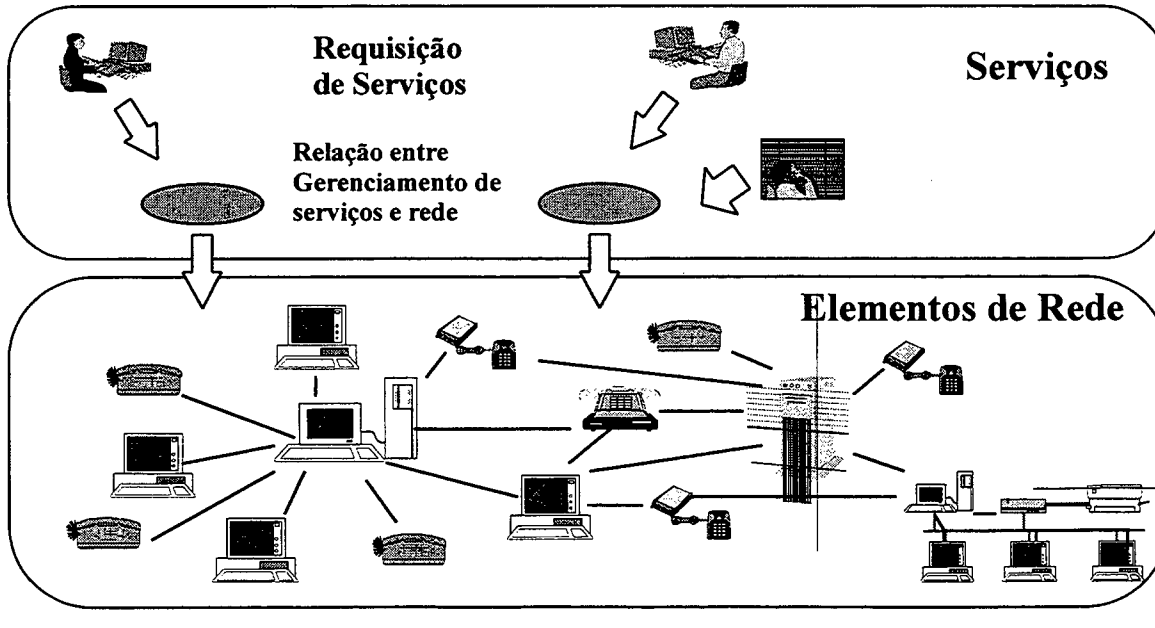


Figura 21: Número de elementos de rede bem maior que o número de serviços.

2.3.5. Necessidades do Gerenciamento de Serviços Distribuídos

Necessidades Comuns aos Sistemas Distribuídos:

⇒ Distribuição;

⇒ Extensibilidade;

⇒ Heterogeneidade;

⇒ Serviços Comuns.

Necessidades Especiais para o Gerenciamento de Serviços:

⇒ Consistência;

⇒ Suporte a Integração de Serviços;

⇒ Interoperabilidade.

2.3.6. Requisitos do Gerenciamento de Serviços

As aplicações de Gerenciamento de Serviços são necessárias para administrar serviços geograficamente dispersos, com computadores heterogêneos e recursos de telecomunicação muito complexos. O propósito do processo de gerenciamento é realizar acessos e modificações transparentes, consistentes e confiáveis destes recursos.

É importante ás indústrias desenvolverem novas tecnologias para que os recursos da rede global possam ser compartilhados e gerenciados por aplicações, de uma maneira eficiente e consistente. Empresas de telecomunicações distinguem-se, umas das outras, pelo oferecimento e gerenciamento dos serviços.

As maiores necessidades para gerenciamento de serviços são :

⇒ *Distribuição* - serviços de telecomunicações são geograficamente distribuídos, necessitando assim, serem gerenciados;

⇒ *Extensibilidade* - necessita de uma plataforma escalável, devido ao fato de que sempre aumentam os recursos, informações, serviços e redes a serem gerenciadas;

⇒ *Heterogeneidade* - A diversidade de recursos, em computação e processos humanos, requer suporte de sistema heterogêneo no domínio de gerenciamento;

⇒ *Consistência* - consistência nos dados é importante para o sucesso dos negócios das Empresas de Telecomunicações e necessários a um completo gerenciamento de serviço de telecomunicações distribuídos;

⇒ *Suporte na Integração de Serviços* - funções de gerenciamento de serviços são necessárias para que os novos serviços possam ser facilmente introduzidos e gerenciados e, existindo serviços, poderão ser integrados;

⇒ *Interoperabilidade* - Diferentes domínios administrativos e de negócios devem realizar a interoperabilidade, integrando gerenciamento de serviços. Este domínio de gerenciamento normalmente tem diferentes políticas empresariais, ambiente de negócios, arquitetura tecnológica subjacente e infra estrutura organizacional.

2.4. REFLEXÃO COMPUTACIONAL NO MODELO DE OBJETOS

A reflexão computacional define uma nova arquitetura de software. A arquitetura reflexiva é composta de um meta-nível, em cujo nível-base encontra-se as estruturas de dados e as ações a serem realizadas sobre o sistema objeto. Assim, em uma arquitetura reflexiva, o sistema computacional possui dois componentes interrelacionados:

⇒ um subsistema objeto, que faz computações sobre um domínio externo ao sistema, e

⇒ um subsistema reflexivo, que faz computações sobre o sistema objeto.

Em linguagens orientadas a objetos, a reflexão computacional é implementada através de meta-classes, que contém informações sobre a estrutura de classes da aplicação ou de meta-objetos que contém informações sobre a implementações e a interpretações de objetos da aplicação.

Nos últimos dez anos, e, com maior intensidade recentemente, a reflexão computacional tem atraído a atenção de pesquisadores, como forma de extensão de linguagens de programação e pela sua utilidade em diversos mecanismos, como por exemplo, mecanismos de programação distribuída, de forma simplificada e transparente à aplicação. A arquitetura reflexiva, ao estabelecer a separação de domínios, permite a construção de arquiteturas reusáveis de software. Podem ser reutilizados os componentes do programa de nível base e os componentes do programa de meta-nível.

A separação de domínios é o aspecto mais atraente da arquitetura reflexiva, não apenas por incentivar a reutilização mas, principalmente, por permitir ao programador da aplicação concentrar-se na solução do problema de programação, específico do domínio da aplicação. A reflexão tem sido adotada para expressar propriedades não funcionais do sistema, como confiabilidade e segurança, de forma independente do domínio da aplicação. [LIS97].

Os múltiplos significados do termo reflexão exigem um cunho especifico no contexto deste trabalho: tanto reflexão, como ato introspectivo de meditação, ou de conclusão dela advinda, seja reflexão, ou propriedade física de modificação da direção de propagação de luz, calor, som, desviando da primeira direção, adequam-se para definir o

significado de reflexão computacional. O primeiro significado traduz o objetivo de reflexão computacional, que é a capacidade de um sistema de atuar sobre si mesmo e não sobre o que o sistema deve produzir, realizando deduções e computações sobre dados internos do próprio sistema. Já, o segundo significado relaciona-se mais diretamente a uma forma de implementação de reflexão computacional no modelo de objetos (Ao ser enviada uma mensagem a um objeto, ela é desviada para o seu meta-objeto correspondente e, como resultado dessa reflexão, passa a realizar computações no meta-nível).

O conceito de reflexão computacional no modelo de objetos foi introduzido, em 1987, por Patti Maes [MAE87]. No conceito de Maes, reflexão computacional é a atividade executada por um sistema computacional quando faz computações sobre (e possivelmente afetando) suas próprias computações. Por sistema computacional entende-se um sistema baseado em um computador, com o objetivo de realizar computações e fornecer informações em um determinado domínio de aplicação. Neste conceito, a reflexão é definida como uma forma de introspeção, no qual o sistema tenta tirar conclusões sobre as suas próprias computações, as quais podem ser afetadas posteriormente. Em resumo, por reflexão computacional entende-se:

> **Definição 1**: **Reflexão computacional é** toda a atividade de um sistema computacional, realizada sobre si mesmo, e de forma separada das computações em curso, com o objetivo de resolver seus próprios problemas e de obter informações sobre suas computações.

Dotar um sistema computacional de tal capacidade significa possibilitar a realização de computações flexíveis, capazes de alterar e de alterar dinamicamente componentes do sistema. Uma definição, neste sentido, é dada por Steel [STE94]: a reflexão computacional é a capacidade de um sistema computacional de interromper o processo de execução (por exemplo, quando ocorre um erro), realizar computações, ou fazer deduções no meta-nível, e retornar ao nível de execução, traduzindo o impacto das decisões, para então retomar o processo de execução.

Nesta última definição é introduzido o conceito de níveis de computação, aproximando-se do significado físico de reflexão, visto que a ocorrência de um evento, (por exemplo, um erro) no processo de execução, reflete-se em um nível superior (meta-nível) de computação .

Neste sentido genérico de intervenção sobre o processo de execução, principalmente no que se refere ao fluxo de execução das ações de um programa, a reflexão computacional não é novidade. Na programação em linguagens do tipo *assembler,* era comum utilizar comandos inócuos, a exemplo do comando NOP - No Operation, para marcar pontos do programa em que deveriam ser feitas substituições de instruções, em tempo de execução. Substituindo-se um NOP por um comando de desvio, podia-se alterar dinamicamente o fluxo de execução. Outras práticas semelhantes, Como por exemplo o uso de sobreposições de código de instruções, eram comumente realizadas para contornar problemas de falta de memória, para acomodar todas as instruções de um programa.

Atualmente, a reflexão computacional possui um significado mais diretamente relacionado com o modelo de objetos, embora seja encontrada também no modelo de programação funcional e de programação em lógica. No modelo de objetos, refere-se à obtenção de dados sobre as propriedades de classes e de objetos de uma aplicação, e também, a possibilidade de modificar esses dados, modificando-se desta maneira as propriedades das classes e dos objetos da aplicação.

Arquitetura Reflexiva

Quando uma aplicação é executada, computações são realizadas sobre dados de seu próprio domínio e com o objetivo de atender os requisitos especificados para aquela particular aplicação, quando a computação é reflexiva, ela realiza computações sobre dados do próprio sistema, tendo por objetivo resolver problemas do sistema em si, ou atuar sobre a aplicação . A auto-representação de um sistema permite a alteração e a adaptação de determinados aspectos de sua estrutura e comportamento.

Entende-se por domínio de um sistema computacional todo conteúdo de informação, expresso por objetos e operações, relacionado com uma determinada área. Para efeito das

definições a seguir, designa-se de domínio *externo* do sistema ou de *domínio da aplicação* o conteúdo de informação relacionado com a aplicação, e designa-se de domínio *interno* do sistema, ou de auto-domínio, o conteúdo de informação sobre o próprio sistema, incluindo a representação interna do domínio da aplicação.

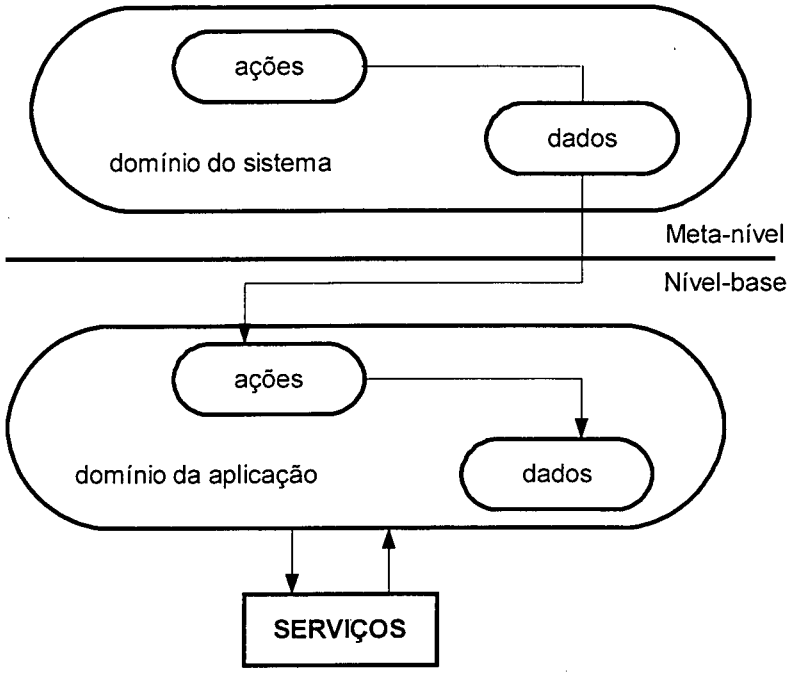


Figura 22. Arquitetura Reflexiva.

A reflexão computacional define uma arquitetura em níveis, denominada **arquitetura reflexiva,** composta de um meta-nível, em que se encontram as estruturas de dados e as ações a serem realizadas sobre o sistema objeto, localizado no nível base. A figura 22 esquematiza a arquitetura reflexiva, mostrando que as ações no meta-nível são executadas a vista de dados que representam informações do programa de nível base, enquanto que, por sua vez, o programa de nível base executa ações sobre os seus próprios dados, com a finalidade de atender aos usuários de seus serviços.

Assim, na arquitetura reflexiva, sistema computacional possui dois componentes interrelacionados:

⇒ um subsistema objeto, que faz computações no domínio externo do sistema, e ;

⇒ um subsistema reflexivo, que faz computações no sistema objeto.

Os dados do nível base são usados no meta-nível, para a realizações de computações reflexivas que podem interferir nas computações de nível base.

Definição 2: Sistema reflexivo é um sistema computacional que possui um componente capaz de realizar computações no domínio interno do sistema, e atuar nas computações do domínio externo.

O modo de atuação exige uma conexão entre o componente reflexivo e o sistema objeto. Esta conexão exige que o componente reflexivo (meta-nível) possua informações sobre o sistema objeto (nível base), tais como: quais são os componentes do sistema objeto, informações sobre situações (por exemplo, erros, eventos) que exigem uma atuação do meta-nível, entre outras. Estas informações variam de acordo com os objetivos da reflexão computacional: o componente reflexivo pode atuar sobre a estrutura, ou sobre o comportamento do sistema objeto.

Exemplo: Atuação sobre o comportamento do sistema objeto

Supondo que um objeto x receba uma mensagem x.ExibirValor, endereçada ao método ExibirValor, com o seguinte comportamento: um valor y é exibido no vídeo da estação de trabalho, onde o objeto x está sendo executado O seu meta-objeto pode alterar esse comportamento, fazendo que o valor de y seja exibido em diversas estações de trabalho, além daquela onde o objeto x encontra-se em execução.

No modelo de objetos, a computação reflexiva pode ser feita sobre classes ou instâncias de classes. No primeiro caso, o meta-nível é composto por meta-classes, as quais contêm informações sobre os aspectos estruturais de componentes do nível base tais como descrição de variáveis e de métodos que irão compor todas as suas instâncias. Esta estrutura pode ser alterada e, em decorrência disso, todas as instâncias também serão alteradas. No segundo caso, o meta-nível é composto de meta-objetos, que contém informações sobre os aspectos de comportamento dos objetos de nível base, como por exemplo, a maneira pela qual um determinado objeto trata uma mensagem. As seções seguintes descrevem mais detalhadamente estes conceitos.

O modelo de meta-classes

Algumas linguagens orientadas a objetos utilizam meta-classes para descrever a estrutura de todas as suas classes, a fim de que as informações necessárias a construção de uma classe, encontrem-se em sua meta-classe. Mais especificamente, meta-classes podem ser assim definidas [BOO94], [GRA89]:

Definição 3: Uma **meta-classe Mcx** é uma classe que descreve a estrutura de uma classe **x** e cujas instâncias também são classes.

Ou, simplesmente, uma meta-classe é a classe de uma classe; se a classe é obtida por instanciação de uma meta-classe, então uma classe é tratada como um objeto. A propriedade de ser instanciável como uma nova classe permite que classes possuam capacidade de realizar computações sobre seus próprios dados, fornecendo serviços a seus clientes. E também as distinguem de classes tradicionais (que descrevem a estrutura de objetos), visto que o seu domínio é a própria classe; seus dados referem-se a nomes de métodos, instâncias, heranças e seus métodos podem acessar esses dados. Existem diversas possibilidades de estruturação de linguagens com meta-classes:

⇒ **Classes são objetos :** todas as classes podem ser vistas como objetos e todos os objetos podem ser vistos como classes. Se um objeto pode ser visto como uma classe, isto significa que o próprio objeto contém informações sobre a sua estrutura, dispensando a existência de uma classe superior (a meta-classe) para descrever a estrutura dos objetos.

⇒ **Classes são instâncias de uma única meta-classe :** todos os objetos são instâncias de uma classe e todas as classes são instâncias de uma classe ancestral comum, denominada de meta-classe raiz. A meta-classe fornece uma interface para a estrutura de todas as classes. Isto permite que as classes sejam consideradas como objetos do programa que podem ser acessadas pelo programador.

⇒ **Todas as classes possuem meta-classes :** uma classe sempre possui uma meta-classe e todas as meta-classes possuem uma classe ancestral comum. Portanto, existem diversos níveis de classes: (a) a classe de um objeto; (b) a meta-classe da classe do objeto; (c) a meta-classe, da qual foi derivada a meta-classe; (d) a Meta-classe; (e) a classe da Meta-classe. .

Em linguagens que estruturam as suas classes a partir de meta-classes, estas propriedades são herdadas pelas classes, de forma que todas as classes são especializações de suas meta-classes. Na árvore de herança, a meta-classe é a super-classe de todas as classes e, conseqüentemente, as classes de todos os objetos são descendentes diretas de alguma meta-classe. Isto significa que a alteração em alguma meta-classe da árvore de herança pode ser percebida por todos os descendentes da árvore de herança .

A possibilidade de alterar a meta-classe, refletindo-se em todas as suas classes descendentes pode ser restringida na implementação da linguagem.

Reflexão estrutural de classes

Meta-classes, quando acessíveis aos usuário, permitem a realização de reflexão estrutural, que pode ser assim definida, com base em [FER89] :

Definição 4: Reflexão estrutural de uma classe x é toda a atividade realizada em uma meta-classe Mcx, com o objetivo de obter informações e realizar transformações sobre a estrutura estática da classe x.

Informações e transformações típicas de reflexão estrutural são: (a) obter informações sobre a classe x: sua classe ascendente, suas descendentes, suas instâncias, seus métodos e interfaces; (b) alterar a classe x: modificar seus atributos e seus métodos; e ainda, (c) atuar sobre classes: criar novas classes; eliminar classes existentes; renomear classes.

O modelo de meta-objetos

Neste modelo, a reflexão computacional realiza-se por associação de um objeto de nível base em um objeto de meta-nível. O objeto de meta-nível, denominado de meta-objeto, contém meta-informações sobre os objetos de nível base. Pode-se dizer que um meta-objeto, ao referir-se a um objeto, representa um outro objeto, denominado de referentes. Objetos e meta-objetos são interconectados, de tal forma que uma modificação, em qualquer um deles provoca efeitos no outro, de forma dinâmica. Esta associação causa-efeito é conhecida como **conexão causal**.

Genericamente, o conceito de meta-objeto é assim colocado, por Foote [FOO92]: meta-objetos são objetos que definem, implementam, dão suporte ou participam de alguma maneira da execução da aplicação, ou dos objetos de nível base.

O modelo de meta-objetos é distinguido do modelo de meta-classes principalmente por sua associação a objetos e não a classes de objetos. Qualquer objeto da aplicação pode ser associado a um, ou a mais meta-objetos, que definem, implementam ou participam de diferentes formas da execução dos objetos de nível base. Suas principais características são:

Individualidade: a cada objeto de nível base pode ser associado um meta-objeto. O objeto associado a um meta-objeto é aqui denominado de **objeto reflexivo.** Do ponto de vista de estruturação de aplicações, isto significa que podem ser selecionadas determinadas instâncias de classes para a realização de reflexão computacional, permanecendo não reflexivos outros objetos instanciados a partir das mesmas classes.

Separação de Classes: a classe do meta-objeto é distinta da classe de seu referente, permitindo que objetos de uma mesma classe sejam associados a diferentes meta-objetos; inversamente, meta-objetos, instanciados a partir de uma mesma classe podem ser associados a objetos de nível base de diferentes classes.

Reflexão comportamental: a classe de um meta-objeto contém informações sobre seu referente, incluindo seus atributos e seus métodos. Portanto, o comportamento de um objeto de nível base pode ser modificado pelo seu meta-objeto.

Associação dinâmica: a **associação** entre um meta-objeto e um objeto é feita em tempo de execução. Quando um objeto reflexivo é instanciado, ocorre também a instanciação de seu meta-objeto. Dependendo do protocolo de associação (MOP- Meta Object Protocol), é possível mudar o meta-objeto associado, de forma dinâmica. Esta característica permite que um objeto assuma diferentes comportamentos ao longo do processo de execução, dependendo das classes dos meta-objetos sucessivamente associados a ele.

Definição circular: um meta-objeto é um objeto, visto que é obtido por instanciação. Portanto, um meta-objeto pode ser tratado como um objeto comum e, inclusive, pode ter seu próprio meta-objeto. Esta característica permite a estruturação de uma torre de reflexão computacional.

Das colocações acima, destacam-se como importantes as seguintes características: um meta-objeto é um objeto instanciado a partir de uma classe, este objeto descreve alguns aspectos (não necessariamente todos) de outro objeto ao qual se refere, e de algum modo, participa do processo de execução de seu objeto referente. Em síntese, um meta-objeto pode ser assim definido:

Definição 5: Um **meta-objeto Mox é** um objeto que representa aspectos estruturais e comportamentais de um objeto **o**, a ele conectado. A estrutura de **o** *é* representada como atributos de **Mox** e os aspectos computacionais são descritos como métodos de **Mox.**

A interface entre objetos e meta-objetos é definida através de um MOP - Metaobject Protocol. O protocolo de meta-objetos preocupa-se com questões como [FER89]:

I) quais entidades devem ser transformadas em algo que possa sofrer operações no meta-nível;

II) como o relacionamento entre o nível base e o meta-nível é implementado;

III) quando o sistema passa para o meta-nível;

A primeira questão refere-se à perfeita carcterização dos dados que podem ser manipulados no meta-nível. A atividade computacional do nível base é transformada em dados para o nível superior, determinando quais computações podem ser realizadas. Esta operação de transformação é denominada de reificação ou de materialização, assim definida com base em [FER89]:

Definição 6: Reificação é a transformação de informações sobre a execução de um programa orientado a objetos, em dados disponíveis ao próprio programa.

Os objetos reificados constituem as **metas-informações** sobre as quais são realizadas as computações reflexivas. Aqui, novamente, dependendo do objetivo da reflexão, podem ser determinadas quais meta-informações são necessárias. Estas informações podem incluir as mensagens dirigidas a um objeto (nome do método e argumentos) ou, de forma mais intrusiva, o nome de um atributo encapsulado em um objeto e seu valor.

A reificação tem por objetivo revelar e tornar disponíveis ao programador da aplicação, informações a respeito dos componentes do programa que são conhecidas e tratadas normalmente apenas pelo compilador, ou interpretador da linguagem de programação, e pelo ambiente de execução. Essas meta-informações geralmente não são necessárias para a construção de programas de aplicação mas podem ser bastante úteis quando se trata de construir bibliotecas, ferramentas de desenvolvimento, frameworks e mecanismos de extensão de linguagens de programação.

As metas-informações podem ser estáticas, quando se tratam de informações estruturais, determinadas em tempo de compilação, ou dinâmicas, quando se tratam de informações sobre o processo de execução. As informações estruturais, mais comumente disponíveis são:

⇒ Classe de um objeto - fornece acesso ao nome da classe, identificando o tipo do objeto;

⇒ Herança - fornece informações a respeito de classes ascendentes e descendentes;

⇒ Propriedades - fornece informações sobre as estruturas de dados de uma classe e de seus métodos, podendo incluir informações sobre restrições de acesso aos membros da classe (públicos, privativos).

A segunda (II), dentre as questões acima diz respeito ao relacionamento entre o nível base e o meta-nível. Um meta-objeto possui uma meta-interface, através da qual são oferecidos os serviços genéricos da aplicação , e pode possuir também uma interface operacional, com os serviços que ele oferece ao nível base ou a outros meta-objetos. Por exemplo, um meta-objeto que fornece informações sobre a localização de componentes em sistemas distribuídos, possui funções para acessar o nome e o identificador do componente, informação sobre o que o contém, e o processador que o executa [BUS96]. Um outro meta-objeto pode usar estas informações para implementar transações distribuídas, coordenando os serviços de um grupo de objetos participantes da transação.

A terceira (III), questão acima indaga sobre quem e como inicia o processo de reflexão . Maes[MAE88] aponta duas soluções: A responsabilidade pode ser atribuída ao objeto de nível base que, neste caso, contém código mencionando seu meta-objeto, ou pode ser atribuída ao sistema. Neste **último** caso, o meta-objeto é ativado pelo sistema, quando ocorre algum evento envolvendo o objeto referente, como exemplo de uma mudança de estado de variáveis de instância (variáveis reflexivas) ou o recebimento de uma mensagem dirigida a um método (método reflexivo).

Na interceptação de mensagens, todas as mensagens enviadas ao objeto, ou mais especificamente, a algum método reflexivo do objeto, são delegadas ao seu meta-objeto, passando este a ser o responsável pelo tratamento da mensagem.

A realização da computação no meta-nível ocasionada pelo envio de uma mensagem a um objeto, depende de informações dinâmicas, obtidas durante o processo de execução, tais como:

⇒ sobre o método: nome do método da aplicação destinatário da mensagem;

⇒ sobre a chamada: argumentos que o cliente forneceu na mensagem.

A reificação destes componente possibilita ao meta-objeto reenviar a mensagem ao objeto da aplicação para a execução do método original, além de executar outras computações no meta-nível.

Reflexão comportamental de objetos

Na reflexão comportamental de objetos, também conhecida como reflexão computacional, a função básica do meta-objeto é explicitar como o objeto reage diante de uma mensagem, possibilitando a intervenção no estado da computação . Esta **intervenção** é a essência da computação reflexiva dinâmica, realizada de forma transparente no objeto e seus clientes. Busca coletar e registrar informações sobre o processo de execução, a exemplo de estatísticas sobre performance, informações para fins de depuração e de monitoração da execução, ou com a finalidade de modificar o curso do processo de execução, como no caso de, determinar qual computação deve ser feita a seguir, ou de ativar computações alternativas, ou ainda, prover outra infra-estrutura de execução, como computação distribuída.

Definição 7: **Reflexão comportamental** de um objeto **o** é toda a atividade realizada por um meta-objeto **Mox,** com o objetivo de obter informações e realizar transformações sobre o comportamento do objeto **o.**

Como o comportamento de um objeto é ditado por seus métodos, a reificação individualizada de uma mensagem que ativa um método e seu conseqüente tratamento pelo meta-objeto, faz com que a reflexão seja restrita ao método chamado, ou seja, torna-se uma reflexão computacional particular de cada método e não de todo o objeto.

Os métodos de um objeto reflexivo são divididos em dois grupos, os reflexivos e os não-reflexivos. Somente os primeiros têm a sua execução controlada pelo meta-objeto.

A propriedade de encapsulamento é essencialmente estática e assegura que o objeto seja visível apenas por suas interfaces, tornando invisível a sua implementação interna (suas variáveis de instância e seus métodos). Na reflexão comportamental realizada por interceptação de mensagens, esta propriedade não é alterada sob o ponto de vista do programa de aplicação, porque a estrutura interna do objeto permanece inviolada; apenas aspectos de seu comportamento podem ser substituídos externamente.

Ao contrário da reflexão comportamental, a reflexão estrutural viola o encapsulamento do objeto, visto que pode substituir métodos, e mesmo alterar a classe de uma instância já existente.

Torre de reflexão

A arquitetura reflexiva admite diversos meta-níveis, caracterizando uma torre de reflexão : cada meta-objeto pode ter um ou mais meta-objetos a ele associado. Meta-objetos, ao serem considerados objetos, podem enviar e receber mensagens de outros meta-objetos. Quando um meta-objeto recebe diretamente mensagens de outros meta-objetos, realiza-se uma computação reflexiva do sistema.

Cada nível da torre de reflexão constitui um domínio D_i, considerado como domínio base do domínio D_{i+1}, seu meta-domínio esta situado no meta-nível imediatamente superior. Cada domínio D_i é, ao mesmo tempo, o domínio base do domínio do nível superior D_{i+1}, e o meta-domínio do domínio D_{i-1}, situado no nível inferior, exceto por D_0, constituído por referentes que podem ser usados apenas no nível base .

Exemplo: Objetos da aplicação podem ser replicados e distribuídos no meta-nível, de forma transparente à aplicação. Objetos de nível base (D_0) são replicados e gerenciados por meta-objetos (D_1), sendo as réplicas executadas de forma distribuída, usando um protocolo de comunicação, fornecido por meta-meta-objetos (D_2).

Conceitualmente, a torre de reflexão é infinita, na qual cada meta-nível define o comportamento do nível base imediatamente inferior. Porém, devido à complexidade e ao custo computacional da torre reflexiva, o número de níveis não deve ultrapassar de três: o nível da aplicação;o meta-nível ; o meta-meta-nível [ANC95].

Resumo

A reflexão computacional permite fazer computações sobre uma computação, com o objetivo de alterar e adaptar sistemas de forma dinâmica. Define uma arquitetura em níveis, denominada arquitetura reflexiva, composta por um meta-nível, no qual se encontram as estruturas de dados e as ações a serem realizadas sobre o sistema objeto, localizado no nível base.

Em linguagens orientadas a objetos, a reflexão computacional é realizada através de meta-classes ou meta-objetos, que se distinguem por sua abrangência. A reflexão realizada, por meio de uma meta-classe, altera todas as instâncias da classe, enquanto que e reflexão realizada, por meio de um meta-objeto, refere-se a uma única instância. Em ambas as formas, o objetivo é fornecer informações sobre a representação dos métodos e sobre propriedades das classes ou dos objetos da aplicação.

Estas informações são fornecidas por um protocolo de meta-objetos MOP, que define quais informações e como serão reveladas ao programador. Não existe uma definição padrão de MOP. Por contrário, existem abordagens distintas para a definição destes protocolos. Alguns oferecem apenas informações estatísticas, como nome da classe e suas instâncias, nome de seus métodos, seus parâmetros de definição, nome e tipo de seus atributos. Outros fornecem meta-informações sobre o processo de execução, a exemplo de valores de variáveis de instância, chamadas de métodos e argumentos enviados. Também existem diferenças na forma de tratamento das meta-informações, visto que alguns protocolos permitem alterar e assim modificar dinamicamente o programa de nível base, enquanto outros permitem apenas introspeção, ou seja, as meta-informações servem como dados para o meta-nível examinar o nível base.

Componentes que podem ser observados, ou alterados durante o processo de execução, são denominados componentes abertos [KIC96], pois alguns detalhes de sua implementação são revelados a seus clientes. De forma oposta, componentes fechados (caixa-preta) revelam a suas funcionalidades através de interfaces bem definidas, e escondem detalhes de sua implementação. Ainda de acordo com Kiczales, [KIC96], componentes abertos oferecem ao programador da aplicação mais opções, permitindo:

⇒ usar apenas as funcionalidades básicas de um componente, quando a sua implementação for adequada;

⇒ controlar a estratégia de implementação do componente, quando necessário; decidir sobre a funcionalidade e a estratégia de implementação de forma completamente separada.

O paradigma de reflexão computacional tem atraído a atenção de pesquisadores como forma de extensão de linguagens de programação e também pela sua utilidade na implementação de diversos mecanismos, como por exemplo: mecanismos de programação distribuída, de forma simplificada e transparente à aplicação, algorítmos de gerenciamento de aplicações do tipo tempo-real, programação concorrente e tolerância a falhas. Tem-se mostrado uma solução bastante adequada para prover ao programador da aplicação flexibilidade nestas aplicações. Estas características permitem, por exemplo, que um sistema distribuído de gerência de serviços, separe o código fonte da aplicação, e o código fonte que contém, os aspectos de controle responsáveis pela implementação da gerência desejada.

3. FERRAMENTAS UTILIZADAS PARA O DESENVOLVIMENTO DO TRABALHO

3.1. O AMBIENTE DE *HARDWARE* E *SOFTWARE*

O trabalho foi implementado com a tecnologia Cliente/Servidor utilizando Java e CORBA, e com as seguintes característica.

Hardware:

⇒ processador Pentium de 133 e 233MHz;

⇒ com no mínimo 32 Mb de RAM;

⇒ disco rígido de no mínimo 1 Gb;

⇒ placa *Ethernet 10 Mbit/s*

Software:

Servidor

⇒ *Microsoft Windows NT 4.0* Servidor;

⇒ *Internet Information Server;*

⇒ *Symantec Café Visual JavaSoft e JDK* (inclusive RMI e JDBC);

⇒ *Visigenic VisiBroker* para *Java V3.2.*;

⇒ *Netscape FastTrack* Servidor;

⇒ *Netscape Navegator 4.0;*

⇒ *Microsoft Access-97.*

Cliente

⇒ *Microsoft Windows NT 4.0 Workstation ou Microsoft Windows 95*;

⇒ *Symantec Café Visual;*

⇒ *JavaSoft JDK* (inclusive RMI e JDBC);

⇒ *Visigenic VisiBroker* para Java V3.2.;

⇒ *Netscape Navegator 4.0.*

O *Visigenic VisiBroker* poderá ser substituído por outro *software* CORBA, como os demais *software.*

Além das características acima, o Sistema Operacional deve permitir operações distribuídas, que seja tolerante a falhas, e se possível, controladores de segurança do tipo RAID (*Redundance Array Inexpensive Disc*), placa de rede, modem, ou ligação à um servidor WEB.

4. ANÁLISE DO MODELO PARA GERENCIAMENTO DE SERVIÇOS DE TELECOMUNICAÇÕES

4.1. Introdução

O crescimento observado nos últimos anos, tanto em número, quanto em tamanho e complexidade, e a popularização de redes de computadores, veio acompanhado da necessidade de interconectar redes anteriormente isoladas para permitir que computadores em diferentes redes compartilhem informações e trabalhem de forma cooperativa.

Para tornar possíveis tais procedimentos, tem se demonstrado a necessidade de sistemas abertos, onde seja viável a comunicação com máquinas de fabricantes diferentes e/ou que utilizem protocolos distintos, afim de possibilitarem a comunicação, a sincronização, o compartilhamento de recursos e a nomeação global, sem perda da segurança do sistema, sem sobrecarregar a administração da rede, e sem penalizar o usuário comum, pela imposição de tecnologias totalmente novas e incompatíveis, com as ferramentas com as quais este estava acostumado a lidar.

Tem se mostrado necessário também um sistema de processamento distribuído adequado à capacidade das novas máquinas disponíveis a preços acessíveis, através de novas redes de fibra ótica de alta velocidade e capacidade de transmissão. Com a popularização dos serviços de hipertexto via rede, o *voice mail,* as teleconferências, e outros serviços baseados em redes de computadores, tem crescido o interesse, tanto por parte dos usuários como das grandes empresas de *hardware* e *software*, por um sistema que seja adequado a este novo cenário no qual a informática ocupe todos os espaços possíveis, e que possua as características descritas anteriormente.

Com o intuito de propor padrões para sistemas abertos, com arquitetura de suporte ao desenvolvimento de aplicações distribuídas em ambientes heterogêneos, várias empresas

tem trabalhado, para impor um padrão para este ambiente e ser aceito pelas organizações internacionais e/ou ter boa aceitação no mercado (padrão de fato).

Dentre as arquiteturas e padrões projetados até o momento temos: DCE (*Distributed Computing Environment*), ANSA (*Advanced Network System Architecture)*, CORBA (*Common Object Request Broker Architecture*), OLE/COM, *OpenDoc*, DCOM e a linguagem JAVA. Em ordem cronológica, DCE e ANSA são tecnologias mais antigas e as demais são as mais recentes, às quais, utilizam a maior parte dos conceitos e modelos de ambas para implementarem suas novas propostas.

Os recentes avanços na área **objetos distribuídos** - têm permitido muitas expectativas positivas para o desenvolvimento de sistemas cliente/servidor, que utilizam, como se fossem locais, recursos disponíveis através da rede, utilizando toda a conceituação de objetos e componentes de software.

A *Internet* permite um alcance global de recursos, a linguagem de programação Java quebrou os limites impostos pela heterogeneidade dos sistemas conectados à rede, o paradigma de reflexão computacional utiliza um modelo de programação em que um sistema pode analisar seu próprio comportamento e atuar sobre ele próprio. Um padrão para acesso a objetos distribuídos está sendo estabelecido por CORBA e implementado por recursos de acesso a *WEB* (*OrbixWeb, IIOP*), permitindo transparência de acesso e de localização nas chamadas remotas, a métodos de objetos distribuídos na rede.

O CORBA, no atual cenário da computação distribuída, desponta com uma das principais tecnologias candidatas a plataforma de suporte à distribuição de objetos.

O CORBA é constituído por um conjunto de objetos básicos e de rede os quais são conhecidos e confiáveis, atuando como uma camada de *software* intermediário, permitindo que aplicações de gerência distribuídas sejam constituídas em ambientes heterogêneos, independente dos detalhes e dos problemas característicos da distribuição. O CORBA é uma tendência mundial, e já está disponível para as principais plataformas existentes no mercado e provavelmente, num futuro próximo, ocorrerá sua disseminação nas principais organizações acadêmicas e empresariais.

Observa-se que até o momento, as empresas de telecomunicação, oferecem seus serviços aos assinantes, utilizando tecnologia com sistemas analógicos, pois, a bem pouco

tempo, os serviços digitais foram disponibilizados para este setor. Além de utilizarem equipamentos complexos e de fabricação proprietária, como por exemplo o equipamento SIDATA que implementa o serviço de despertador automático. E muitos dos serviços oferecidos, ainda são semi-automatizados.

Hoje as Empresas de telecomunicações, já despertaram para o grande problema da globalização e das privatizações, que impõe que as mesmas, se tornem mais competitivas e competentes para conquistar o maior número de clientes com seus serviços e produtos. Com este novo paradigma, a evolução e o baixo custo das **Tecnologia de Informação**, as empresas encontraram um novo caminho que engloba em grande parte a solução para o problema. A **Internet** e a **Extranet.**

4.2. Objetivos e Metas

A meta inicial deste trabalho, consiste em simular serviços oferecidos por empresas de telecomunicações na rede *Internet*, gerando bases de informações gerenciáveis (MIBs) que disponibilizará as informações para que o sistema de gerência de serviços implemente as funcionalidades de gerência , através da linguagem JAVA, da arquitetura CORBA, do paradigma de Reflexão Computacional e também permita que através destas bases, disponibilize as informações para diversos segmentos administrativos da empresa, consultarem, gerarem gráficos, estatísticas, fluxo de trabalho para tomada de decisões e utilizações variadas destas informações através de ferramentas computacionais disponíveis.

Este trabalho procura responder as seguintes questões: 1.Como fornecer um serviço ao assinante? - 2. Quais funções precisam ser gerenciadas para que se tenha a qualidade de serviço prevista em contrato? - 3. Como definir o modelo de informações, tal que este, preserve uma visão única e consistente para diferentes usuários de serviços? - 4. Como é implementado e gerenciado um serviço? - 5. Como gerenciar serviços através de jurisdições de telecomunicações que podem ter sistemas de serviços oferecidos, políticas de preço, ambiente comercial, administrativo e estrutura tecnológica diferentes? - 6. Como gerenciar força de trabalho e o fluxo de tarefas necessárias para os serviços? - 7. Qual é a relação

entre gerenciamento de serviços e de redes? - 8. Por que a utilização de JAVA, CORBA e Reflexão Computacional para gerenciar serviços?

4.3. Metodologia Utilizada

Devemos lembrar que o desenvolvimento de um sistema computadorizado não pode ser entendido apenas como sinônimo de codificar programas, como acontecia há 25 ou 30 anos atrás. Hoje em dia, sabe-se que esta não é uma boa conduta. Para haver sucesso no desenvolvimento de sistemas, torna-se necessária a utilização de uma metodologia de trabalho ou especificações em Métodos Formais. Como os Métodos Formais pesquisados não atendem por completo a especificação do trabalho proposto, partimos para escolha de uma metodologia que assegure o desenvolvimento correto da trabalho.

Uma **metodologia** pode ser entendida como uma dissertação sobre a maneira de se utilizar um conjunto coerente e coordenado de métodos para atingir um objetivo, de modo que se evite, tanto quanto possível, a subjetividade na execução do trabalho.

Um **método** pode ser entendido como um procedimento a ser adotado para se atingir um objetivo. Para tanto, o método se vale de um conjunto de técnicas.

Uma **técnica** pode ser entendida como sendo um modo apropriado de se investigar sistematicamente um determinado universo de interesse ou domínio de um problema. Para se expressar, uma técnica faz uso de uma notação.

Uma **notação** é um conjunto de caracteres, símbolos e sinais formando um sistema convencionado de representação ou designação.

O uso de metodologias que proponham a modelagem dos sistemas é a melhor maneira de se resolver os problemas da atividade de desenvolvimento. Deve ser definido o ciclo de vida do sistema adotado, ou seja, quais as fases de trabalho a serem executadas. Uma boa metodologia deve também apresentar definição clara de **"quem"** faz **"o quê"**, **"quando"**, **"como"**, e até mesmo **"onde"**, para todos que estejam envolvidos diretamente ou não com o desenvolvimento do sistema.

Uma metodologia deve definir quais as fases de trabalho prevista no desenvolvimento de sistemas. Para cada fase, quais as técnicas adotadas. São exemplos de técnicas: Análise

e Projeto Estruturado, Análise Essencial, Análise e Projeto Orientado a Objeto. A metodologia deve ainda, para cada técnica adotada, definir quais as ferramentas utilizadas. São exemplos de **ferramentas**: Diagrama de Fluxo de Dados, Diagrama de Entidade-Relacionamento, Diagrama de Eventos, Diagrama de Transição de Estados. Cada ferramenta irá produzir um tipo de modelo. São exemplos de modelos: Modelo Funcional, Modelo Conceitual de Dados e Modelo de Controle. A metodologia deve estabelecer ainda quais os pontos de controle e padrões de qualidade, além das responsabilidades do pessoal envolvido nos controles e métricas.

Conforme se pode notar, é inegável a utilidade de se dispor de uma metodologia que seja adotada no desenvolvimento de sistemas. Em função disso, desde meados da década de 70, vários autores publicaram propostas de técnicas que, quando reunidas em uma metodologia podem produzir ótimos resultados. Tais técnicas fazem uso de ferramentas que dão origem a modelos que representam as principais perspectivas de um sistema.

O quadro a seguir, mostra as principais ferramentas de modelagem de sistemas utilizadas por quatro técnicas.

TÉCNICAS	ABORDAGEM	FERRAMENTAS
ANÁLISE TRADICIONAL	• FUNCIONAL	• TEXTOS • FLUXOGRAMAS
ANÁLISE ESTRUTURADA	• FUNCIONAL • DADOS	• DIAGRAMA DE FLUXO DE DADOS • DIAGRAMA DE ESTRUTURA DE DADOS • MINIESPECIFICAÇÕES • NORMALIZAÇÃO • DICIONÁRIO DE DADOS
ANÁLISE ESSENCIAL	• FUNCIONAL • DADOS • CONTROLE	• TABELAS DE EVENTOS • DIAGRAMA DE FLUXO DE DADOS • DIAGRAMA DE ENTIDADE-RELACIONAMENTO • DIAGRAMA DE TRANSIÇÃO DE ESTADOS • DIAGRAMA DE ESTRUTURA DE DADOS • NORMALIZAÇÃO • MINIESPECIFICAÇÕES • DICIONÁRIO DE DADOS
ANÁLISE BASEADA EM OBJETOS	• CONSISTENTE, UNINDO FUNCIONAL DADOS E CONTROLE	• DIAGRAMA DE OBJETOS • TABELAS DE EVENTOS • DIAGRAMA DE FLUXO DE DADOS • DIAGRAMA DE OBJETO-RELACIONAMENTO • DIAGRAMA DE TRANSIÇÃO DE ESTADOS • DIAGRAMA DE ESTRUTURA DE DADOS • PROJETO DA ESTRUTURAS DAS CLASSES • PROJETO DOS MÉTODOS

Dentre as técnicas de modelagem apresentadas, utilizaremos a **Análise Baseada em Objetos** para desenvolvimento desse trabalho. Complemento sobre o assunto encontram-se em [POM95] e [MAR94].

4.4. Análise da Implementação

Este novo paradigma em relação a tratamento de dados internos (Internet/Extranet) e acesso a serviço das grandes empresas em nosso caso as telecomunicações , se tornou, um grande desafio. É neste cenário que a Internet representa um novo paradigma para empresas, não só em termos de recuperação e divulgação de informações, mas principalmente em termos de comunicação e gerenciamento. Uma das principais características da Internet é a sua capacidade de facilitar a comunicação interativa dentro da organização, com os seus clientes, fornecedores e com o mundo externo, isto é, certamente um diferencial competitivo entre as empresas.

Esta mudança de paradigma tem dois grandes impactos sobre os negócios:

Primeiro os Benefícios: Melhoria da eficiência e a eficácia na comunicação empresarial; Maior penetração de mercado; Melhoria da produtividade; Diminuição da distância com o cliente; Ampliação da comunicação com o cliente; Novos serviços.

Segundo a Redução de Custos: Redução de quadro de funcionários; Redução de Viagens; Redução dos Custos de Telecomunicações; Redução do Custo de Distribuição; Redução do Tempo de Execução dos Processos da Empresa; Desintermediação.

Dentro deste paradigma, quando analisamos a questão dos objetos que compõem os serviços e seu gerenciamento, vimos que existe uma divisão de responsabilidades e autorizações entre diferentes grupos de objetos. Dentre estas divisões destacamos a interconectividade entre a gerência da rede propriamente dita, com suas funções de configuração, desempenho, falhas, segurança e contabilização e **a camada de aplicações** na qual destacamos os serviços e ainda o planejamento estratégico e operacional da empresa (Figura 23). Com esta análise montamos um domínio com o propósito de gerenciar serviços, onde conjunto de objetos se relacionam para devidos fins.

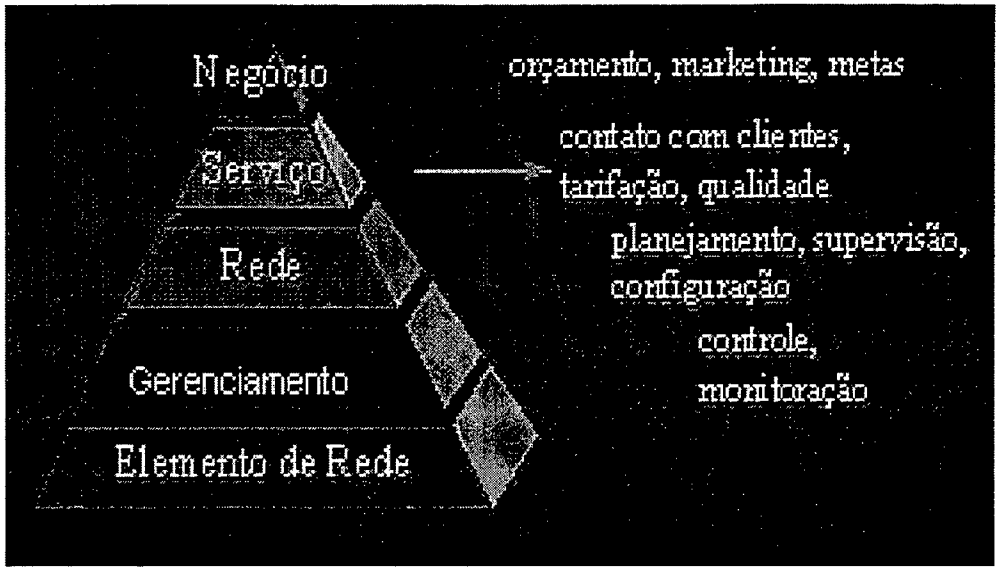


Figura 23. Camadas de Gerenciamento de Telecomunicações

Para implementarmos tais objetos na WEB, levamos em consideração alguns componentes tais como protocolos, linguagens, sistemas operacionais em plataforma distribuída.

O paradigma de orientação a objeto tem se mostrado uma ferramenta útil para o desenvolvimento de sistemas distribuídos, além de sua aplicabilidade ao desenvolvimento de componentes de *softwares* já existentes e testados. A idéia de modelar sistemas distribuídos como um conjunto de objetos interagindo tem sido considerada como apropriada para integrar recursos em ambientes heterogêneos e distribuídos.

Objetos oferecem um modelo natural para sistemas distribuídos abertos, pois os componentes destes sistemas se comunicam usando mensagens através de interfaces bem definidas adequando-se às características de desacoplamento e extensibilidade. Neste contexto, uma forma de garantir a interoperabilidade entre objetos distribuídos é através de padrões como o Modelo de Referência ODP (Open Distributed Processing) (ITU-T/ISO, 1995 a) dos organismos internacionais ISO (Internacional Organization for Standardization) e ITU-T(Internacional Telecommunication Union) e como a arquitetura CORBA (Common

Object Request Arquitecture)(OMG-1997 d) do consórcio internacional OMG (Object Management Group), resultado da união de mais de 700 empresas, centros de pesquisa e universidades.

Suas propostas de "middleware" surgiram da necessidade de romper com o paradigma da arquitetura de distribuição "Three Tier", em que a primeira camada delega as estações de trabalho a função de interface como usuário, a segunda camada o acesso e a lógica do negócio da Empresa (Banco de Dados) e a terceira camada com aplicações servidoras em máquinas mais robustas.

Tais propostas têm despertado o interesse em soluções de gerência tanto de elementos de redes quanto de sistemas de suporte e de aplicações distribuídas, motivados pelas restrições dos modelos atualmente existentes, SNMP (Simple Network Management Protocol) para a Internet e CMIP (Common Network Information Protocol) para um ambiente OSI (Open System Interconnection) para ter-se acesso a recursos gerenciados.

O SNMP é usado para monitorar e controlar elementos de rede, como "gateway", "hub", "switch" e roteadores, através de aplicações em estações de gerência. Embora as suas necessidades de processamento e memória sejam baixos, as funções de "pooling" (de tempos em tempos o gerente pergunta a cada agente se tem algo a informar, isto causa um grande *overhead*, pois na maioria das vezes não existe nada a ser dito) e "trap" (quando o agente tem alguma coisa a informar ao gerente, ele manda a informação, utilizando a rede somente se necessário), não são adequadas para a gerência de aplicações distribuídas.

O modelo de gerência OSI/CMIP oferece uma abordagem mais sofisticada, usando conceitos de orientação a objetos. Embora a arquitetura TMN (Telecomunications Management Network) ofereça um modelo mais rico, sobre as mesmas restrições por ter incorporado o CMIP como protocolo de comunicação... Como exemplo deste interesse, o RM-ODP tem sido adotado como meta-padrão para o desenvolvimento de padrões de gerências em redes e em telecomunicações.

Por outro lado , o crescimento e o sucesso do modelo integrados de objetos CORBA como solução de interoperabilidade tem motivado o mapeamento das especificações dos padrões OSI/CMIP e SNMP para CORBA IDL (Interface Definition Language), através de

adaptadores baseados na possibilidade de CORBA se tornar um padrão integrado de gerência.

O modelo de Informação oferecido por CORBA IDL é comparável ao GDMO (Guidelines For The Definition of Management Objects) do modelo OSI/CMIP e superior ao do SNMP para definição de MIB (Management Information Base), permitindo a clara separação de interface e implementação. CORBA oferece o paradigma da orientação a objetos da mesma forma que os modelos OSI/CMIP e TMN com um maior grau de abstração. Da mesma forma que GDMO apresenta tal característica de separação. CORBA oferece uma solução muito mais simples e não requer que os desenvolvedores conheçam notações de sintaxe abstrata e esquema de codificação e decodificação.

A linguagem IDL é usada para descrever as interfaces dos objetos CORBA, sendo puramente declarativa. Uma definição IDL especifica cada atributo e parâmetros das operações de uma interface definida. IDL, obedece às normas léxicas de C++, apesar de que algumas novas palavras tenham sido introduzidas para dar suporte à distribuição. O IDL foi especificado para dar a facilidade de seu mapeamento para linguagens de implementação como C, C++, *smalltalk* e Java. Uma outra motivação para a sua utilização é a possibilidade de usar ferramentas de desenvolvimento que suportam a tradução de modelos de objeto sem interfaces IDL com compiladores que geram automaticamente os "stubs" e "skeletons" dos objetos da aplicação.

Do ponto de vista de comunicação , CORBA oferece mais transparência e uma arquitetura mais elegante, com o protocolo GIOP (General Inter-ORB Protocol) e o IIOP (Internet Inter-ORB Protocol), sem a necessidade do agente intermediados existente nos modelos SNMP/CMIP. O uso do CORBA como plataforma de gerência resulta em uma arquitetura mais simples, onde as entidades gerentes e os objetos gerenciados com interfaces IDL continuam a desempenhar o mesmo papel, porém já não é mais necessário o agente. Uma referência de objetos interoperável (IOR - Interoperable Object Reference) identifica unicamente um objeto CORBA.

Uma plataforma ORB (Object Request Broker) se encarrega de localizar a implementação de objetos remotamente chamada pelo cliente, de sua ativação, do envio da invocação de métodos e das respetivas respostas. Uma vez conhecida a interface de um

objeto servidor, um cliente CORBA pode ter acesso a seus métodos através das interfaces estáticas e dinâmicas. As interfaces chamadas estáticas usam os "stubs" gerados em tempo de compilação, a partir de descrições IDL dos objetos gerenciados a serem invocados. As interfaces chamadas dinâmicas utilizam a Interface de Invocação Dinâmica, que permite a um gerente construir e realizar chamadas a objetos gerenciados em tempo de execução através do repositório de interfaces. Dentre vários aspectos citados acima que motivam a utilização do CORBA como um método de gerência um dos aspectos que abordarei com mais ênfase em meu trabalho são as implementações CORBA em Java e protocolo IIOP, já embutido nas últimas versões do navegador da "Netscape", permitindo mais a tecnologia WWW (Word Wide Web), pois o HTTP (HiperText Transfer Protocol) criado em 1990 por TIM Berners-Lee, Roy Fielding e Henreck Frystyk Nielsen, encontrando-se na sexta versão; desde sua criação até o momento houveram poucas mudanças, sendo que para entendermos melhor o seu funcionamento, ele é como se fosse um RPC para WEB, ou seja, ele é um processo *stateless*. O cliente estabelece uma conexão para um servidor remoto, em seguida existe uma solicitação , o servidor processa a solicitação, retorna a resposta e então fecha a conexão, isto faz com que o protocolo seja ineficiente, pois a cada solicitação tem que ser montado uma conexão TCP, causando com isto uma grande quantidade de *overhead*, além de pouca segurança em aplicações distribuídas na WEB.

Pensando nesta facilidade de integração com objetos distribuídos os navegadores estão implementando novas opções para acesso a objetos através do ORB.

O CORBA ainda oferece os seus componentes, acessíveis por operações ORB como protocolo de gerência, facilitando o desenvolvimento de aplicações descentralizadas e distribuídas e seu gerenciamento, isto é oferecido através do CORBA *services* e CORBA *facilities*.

Para demonstrar o trabalho, ou seja, a proposta de um modelo flexível para o gerenciamento de serviço, utilizamos como exemplo o serviço despertador ou despertador automático, um dos mais populares serviços em redes de telefonia. Dividimos o trabalho da seguinte forma: uma Descrição Informal; o Diagrama Estrutural; o Diagrama de Objetos, que nos dá uma visão mais ampla da gerência do serviço, sendo descrito os objetos com

seus atributos e métodos; a Implementação do Modelo usando CORBA e JAVA e finalmente conclusões e pespectivas do trabalho.

4.5. DESCRIÇÃO INFORMAL

4.5.1. Despertador Automático

O serviço despertador ou despertador automático é um dos mais populares em redes de telefonia. Através dele o assinante dispõem de uma espécie de agenda eletrônica para recordá-lo dos compromissos e/ou acordá-lo na hora desejada. Para usar o despertador automático o assinante deve discar o código do serviço - 134, seguido pelo código de programação, o qual consiste de um digito que indica a opção escolhida e, ainda mais quatro dígitos que indicam o horário para a realização do serviço.

134 + Código de programação + Hora

Há oito opões para a programação do despertador Automático, senão elas as seguintes:

0 - para o cancelar a programação do serviço;

1 - para fazer a chamada no mesmo dia;

2 - para fazer a chamada no dia seguinte;

3 - para fazer a chamada dois dias após o dia da programação;

4 - para fazer a chamada três dias após o dia da programação;

7 - para fazer a chamada em caráter permanente de segunda a sexta feira;

8 - para fazer a chamada em caráter permanente de segunda a sábado;

9 - para fazer a chamada em caráter permanente de segunda a domingo.

Por exemplo, se o assinante discar "13471200" será feito uma programação para fazer chamadas de segunda a sexta feira sempre ao meio-dia, ou melhor, entre 12: 00 horas e 12:10 horas dependerão da demanda do serviço para este horário.

O equipamento do ambiente de telecomunicações responsável pelo despertador automático é o SIDATA ou "Sistema Despertador e Agenda Teleprogramável". Basicamente o SIDATA deve receber as solicitações dos assinantes e gerar as chamadas nos horários programados. Além destas atribuições ele é responsável também pelo faturamento do uso do serviço.

Em localidades onde o Despertador Automático não está disponível, ele deve ser solicitado via telefonista. O SIDATA pode ser programado via terminal, através de um técnico, quando os assinantes têm acesso apenas via telefonista ao serviço, ou seja, sem poder programar direto pelo telefone como ocorre nas capitais e grandes centros litorâneos.

Durante a programação das solicitações, o SIDATA interage com os assinantes enviando-lhes mensagens sobre a condição de atendimento da sua solicitação. O SIDATA solicita a CTP (Central de Telefonia Pública) o número do assinante, a sua categoria e o número que foi discado, analisa o código de programação e o horário solicitado. Após análise, o SIDATA envia mensagem comunicando se a solicitação foi aceita ou não. As mensagens enviadas pelo SIDATA aos assinantes são as seguintes:

"Despertador Automático - sua programação foi aceita e será atendida".

"Despertador Automático - sua programação esta incorreta - favor consultar a Lista e tentar novamente.

"Despertador Automático - horário sobrecarregado. Favor tentar outro horário, dez minutos antes ou depois".

"Despertador Automático - Programação Cancelada".

Tanto a programação como o cancelamento devem ser feitos com pelo menos trinta minutos de antecedência.

Um SIDATA possui oito juntores de entrada para receber as solicitações dos assinantes e trinta e dois geradores que executam os serviços através da realização de

chamadas para os assinantes. Uma solicitação demora trinta segundos e a execução do mesmo demora em média um minuto. Assim, um SIDATA pode gerar trinta e duas chamadas e receber dezesseis programações por minuto. O limite de pedidos programáveis em um SIDATA é da ordem de seis mil diárias e a limitação por faixa de dez minutos é de duzentos e quarenta pedidos.

O SIDATA, quando atua como um sistema de agenda teleprogramável, funciona mediante a carga de solicitações através de disquetes. Os números a serem chamados são programados através da leitura dos dados destes disquetes.

Componentes do SIDATA:

⇒ Módulo principal - recebe e agenda as solicitações e gera as chamadas. Também é responsável pela supervisão geral do sistema e pelo interfaceamento com os outros módulos. A recepção de chamadas é feita através de juntores que estão interligados à CTP. À medida em que as solicitações são programadas, elas são agendadas na memória de solicitações e um programa de análise garante que as chamadas sejam geradas nos horários e dias programados. Quando é atingido o horário programado pelo assinante, um gerador discará o número. O SIDATA fará três tentativas para fazer a chamada. Entre 23:55 e 0:05 horas o SIDATA não aceita solicitações pois estará fazendo análise dos pedidos (troca de código).

⇒ Módulo bilhetador - armazena dados referentes as solicitações recebidas e atendidas em unidades de disquete. Também armazena periodicamente em discos os dados contidos na memória de pedidos. Este módulo também é usado para fazer a carga de dados à partir destes disquetes. Os dados referentes às solicitações recebidas e geradas são armazenadas em um arquivo denominado de arquivo "Tarifa" para serem utilizados para fazer o faturamento do serviço. A memória de pedidos é armazenada em um arquivo denominado de arquivo "Memória" e utilizado para recarregar os

pedidos quando o SIDATA é desligado ou então para permitir que as solicitações possam ser programadas em outros SIDATAs através de digitação ou através da carga direta para a memória. Este módulo é responsável pela carga de solicitações através de disquetes para a realização do serviço através da empresa de telecomunicação. Pode haver uma impressora conectado ao módulo bilhetador que permite que sejam listadas ordenadamente o conteúdo dos arquivos "Memória" e "Tarifa".

⇒ Módulo gerente - responsável pela interligação dos diversos SIDATAs, transfere pedidos de um SIDATA para outros, elimina pedidos duplicados e o cancelamento simultâneo de pedidos, além de supervisionar os SIDATAs.

⇒ Módulo máquina de mensagens - fornece as mensagens utilizadas pelo SIDATA para informar os assinantes.

⇒ Módulo inversor estático - fornece alimentação para os acionadores do disco do bilhetador e para os ventiladores do sistema.

⇒ Impressora on-line - imprime as solicitações recebidas e geradas pelo SIDATA, alarmes disparados e operações realizadas no sistema. A impressora do módulo bilhetador permite listagens ordenadas do conteúdo dos arquivos "Memória" e "Tarifa".

Basicamente há três formas para a programação das solicitações: juntores de entrada, carga solicitações através dos discos flexíveis e digitação das solicitações.

O SIDATA pode ser operado a partir de um terminal de computador através de direti conversacionais.

4.5.2. Definição do Cenário

A tarefa inicial deste trabalho consiste no desenvolvimento de um software capaz de gerenciar falhas de equipamentos simulados e, além disso, disponibilizar informações simples e significativas, relativas ao comportamento dos mesmos ao longo do tempo. Um administrador de rede e administradores de negócios devem ter condições de acessar essas informações, e através delas, poder inferir estatísticas, auditorias e observar a evolução do sistema dado em um determinado período de tempo. Tais equipamentos e a interação entre eles devem ser simuladas por um outro software (sistema gerenciado) o qual deverá ter uma atuação independente e ainda prover condições para a gerência do software gerenciador, quando este está presente.

O sistema gerenciado deve simular equipamentos de telefone, além do equipamento SIDATA, conectados a uma central telefônica, que deve prover conexões entre eles. Os serviços de agenda teleprográmavel não devem ser contemplados na simulação. As linhas telefônicas não devem ser consideradas, porém deve haver conexão entre os equipamentos através dos meios computacionais disponíveis.

A interação do usuário com o equipamento de telefone deve ser obtida através de interface gráfica. À central de telefonia cabe a comutação das ligações e ao SIDATA o provimento do serviço despertador. A aplicação gerente deve ser capaz de perceber falhas dos equipamentos, selecionar técnicos e abrir ordens de reparos. Entre a aplicação gerente e os equipamentos devem haver agentes cuja função é promover uma interface padronizada e transparente. A aplicação gerente deve ter capacidade para perceber a queda dos agentes e iniciativa própria para tentar repará-los e inicializá-los quando possível, ou em caso contrário, abrir ordens de reparo para os técnicos.

Deve ser possível para uma pessoa responsável pelos técnicos (supervisor dos técnicos) entrar com informações pertinentes aos técnicos no sistema como: a disponibilidade, a especialidade e a localização deles. Tais informações devem possibilitar a seleção do técnico mais adequado para os problemas que possam surgir. Essa pessoa também deve ter condições para solicitar a desativação de agentes e equipamentos quando houver necessidade de atividades de manutenção. Além destas características, a parte

gerenciadora deve ter uma implementação distribuída, obtida através da plataforma CORBA-JAVA.

Devem ser estudadas as possibilidades de comunicação entre a aplicação gerente (ambiente CORBA) e os agentes devem ser implementados através dos recursos providos pelo CORBA-JAVA. A implementação da parte gerenciada bem como a comunicação entre as suas partes (componentes) deve ser obtida através dos meios computacionais disponíveis.

A aplicação gerenciada deve procurar simular características reais do serviço despertador, no entanto, as atividades de gerenciamento executadas sobre ela não têm a pretensão de prover uma solução definitiva para as atividades de gerência de aplicações gerentes do ambiente real de telefonia. Enquanto a aplicação gerenciada procura abstrair e simular o provimento do serviço despertador, a aplicação gerente concentra-se apenas em algumas atividades de gerência as quais devem ser executadas sobre a aplicação simuladora. Em resumo, a aplicação gerente deve apenas prover algumas atividades de gerência sobre a aplicação gerenciada a qual procura abstrair e simular o serviço despertador.

4.5.3. Diagrama Estrutural da Proposta

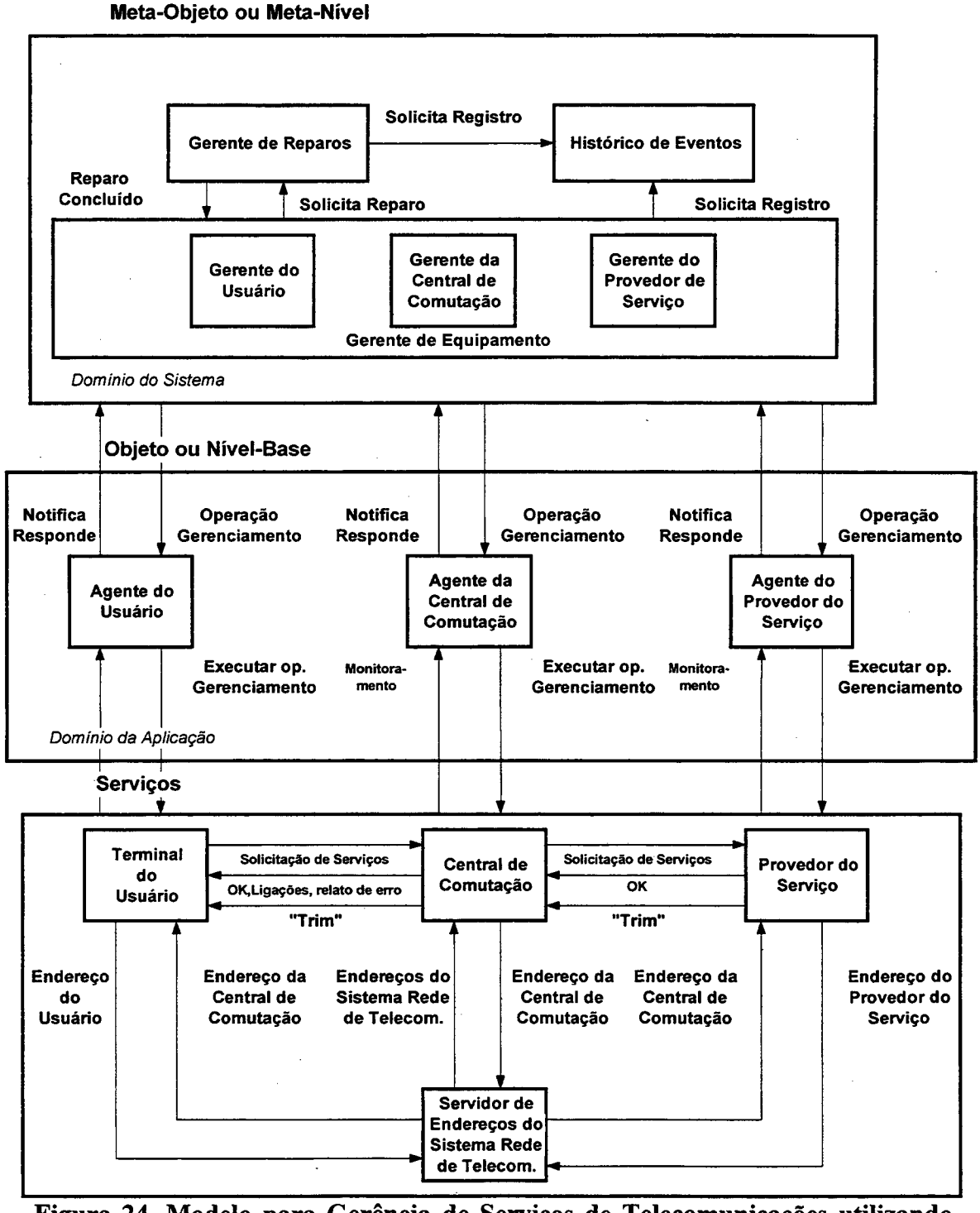


Figura 24. Modelo para Gerência de Serviços de Telecomunicações utilizando uma Arquitetura Reflexiva.

4.5.4. Simulação da Rede de Telecomunicações (RT)

A parte gerenciada da aplicação, como mencionado anteriormente, compõem-se de uma simulação de equipamentos de telecomunicação, consistindo de: Aparelhos de Telefone denominados na aplicação de **Terminal do Usuário,** uma Central de Telefonia denominada de **Central de Comutação** e um Equipamento SIDATA, sendo nesta aplicação denominado de **Provedor do Serviço Despertador.**

Descrição dos Componentes da Simulação da RT

O terminal do usuário consiste de um processo com uma interface gráfica (ver figura 25) a qual interage diretamente com o usuário permitindo a este fazer ligações para outros usuários além de utilizar os serviços fornecidos pelas operadoras de telecomunicações, de maneira análoga ao que acontece aos aparelhos de telefone no mundo real. Neste cenário o único serviço disponível, por razões já mencionadas é o serviço de despertador. Esta interface deve consistir de duas janelas: uma para receber e outra para enviar informações para o destinatário. A janela de recepção também informa ao usuário sobre eventos terminal do usuário (destinatário) inexistente, ocupado, ligação finalizada e central fora do ar. O usuário poderá finalizar ligações e solicitar serviços do provedor do serviço despertador. O usuário solicitará ligações através de identificadores conhecidos semelhantes aos números dos aparelhos de telefone. O usuário poderá informar um identificador ao sistema através de botões representando números similarmente ao que acontece ao um teclado comum de telefone.

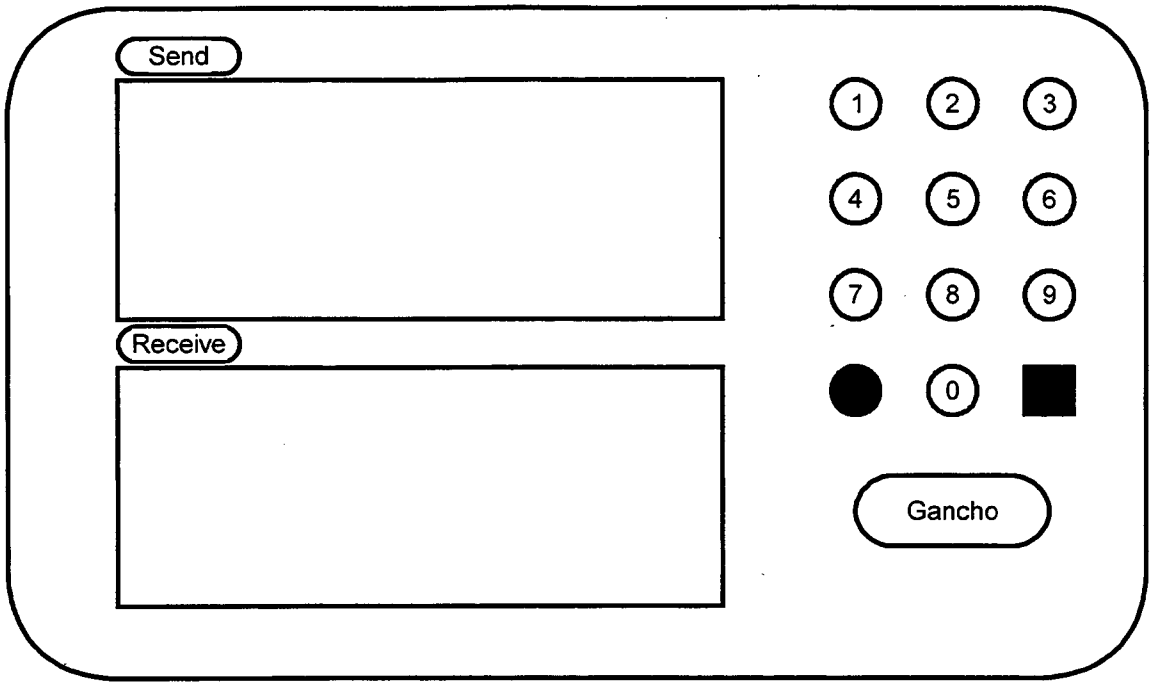


Figura 25 : Modelo de interface do terminal do usuário

A central de comutação tem o papel de conectar temporariamente equipamentos ligados à mesma (terminais telefônicos e SIDATA). Estas conexões são sempre realizadas de um para um (terminal para terminal ou terminal para SIDATA). No caso do destinatário ser inexistente ou estar ocupado esta deve retornar mensagens de erro ou aviso.

O provedor do serviço despertador - deve receber as requisições dos usuários, verificar disponibilidade dos horários para a execução dos serviços, confirmar ou recusar as requisições executar os serviços confirmados. Como em um SIDATA real o provedor do serviço despertador também deve ter um limite de requisições para tratar ao mesmo tempo e outro para o número de requisições que podem ser atendidas em um mesmo horário e data.

O provedor de endereços dos equipamentos - além dos componentes citados que simulam equipamentos reais da rede de telefonia há um quarto componentes neste cenário: o provedor de endereços dos equipamentos. A sua única incumbência é a de solucionar os problemas advindos com a necessidade de conhecimento mútuo dos endereços dos equipamentos que são simulados neste cenário através de processos. Cada terminal do usuário e o provedor do serviço despertador devem conhecer o endereço da central de

comutação e esta deve conhecer os endereços de todos os componentes da aplicação gerenciada.

Todos os componentes do subsistema gerenciado são simulados através de processos, e embora a linha de transmissão não seja considerada e por conseqüência não modelada, os terminais dos usuários e o provedor do serviço despertador estão conectados à central de comutação. A comunicação entre os processos que simulam os componentes da aplicação gerenciada é obtida, neste cenário, através do protocolo IIOP.

4.5.5. Aplicação Gerente

A aplicação de gerência deve ter uma concepção simples e independente da aplicação gerenciada. A presença ou a ausência da aplicação gerenciadora não deve interferir no funcionamento normal da aplicação gerenciada, apenas trocar informações de gerenciamento com ela e eventualmente atuar no seu funcionamento. É desejável que o funcionamento da aplicação gerente seja semelhante a uma peça plugável que se encaixa ao sistema gerenciado, apenas gerenciando e não dominando.

Devido a sua natureza distribuída a aplicação gerente deve comportar componentes que se integrem através da troca de informações de gerenciamento e do provimento de serviços.

A aplicação do gerente é composta por três entidades principais, responsáveis diretamente pelas atividade de gerência: os gerentes de equipamento, o gerente de reparos e o histórico de eventos, conforme demonstrado na figura 23 apresentada na seção 4.2. Além destes há ainda os agentes, tela de interação com o técnico, tela de interação com o supervisor dos técnicos e a tela de interagir com o operador da rede e com administradores de negócios.

As atividades de gerenciamento executadas sobre a aplicação gerenciada são basicamente a detecção e a solução das falhas ou panes dos equipamentos da aplicação gerenciada e o registro de todos os fatos pertinentes ao funcionamento deles para consultas posteriores.

Na verdade os gerentes de equipamento, o gerente de reparos e o histórico de eventos são componentes essenciais para a aplicação gerente e sempre devem estar presentes. Os demais componentes são partes externas que apenas consomem ou atuam nas informações de gerenciamento produzidas pela aplicação gerente.

4.5.6. Descrição dos Componentes da Aplicação Gerente

O gerente de equipamento é a parte responsável pela gerência direta dos equipamentos. Há um gerente de equipamento para cada tipo de equipamento e existe um agente para cada equipamento da aplicação gerenciada.

O gerente de equipamento recebe respostas e notificações dos agentes e então encaminha solicitações para o histórico de eventos para que este registre estes eventos pertinentes ao funcionamento da aplicação gerenciada.

Quando um gerente qualquer envia uma operação de gerenciamento para um agente, e se houver retomo do agente para o gerente de alguma informação especifica aquela operação, este retomo se chama resposta. Se, por outro lado, o agente tomar iniciativa de informar o gerente sobre algum evento do equipamento, que não tenha sido requisitado pelo gerente, esta informação se chama notificação.

No caso da ocorrência de uma notificação de queda do equipamento, o gerente de equipamento solicita ao gerente de reparos a solução do problema, assim como ocorrerá no caso de qualquer outro tipo de problema que venha a ocorrer no equipamento.

Se em um determinado período de tempo, o gerente de equipamento não receber nada do agente, ou seja um notificação de "estou vivo", ele assume que ocorreu falha com o agente e então solicita o reparo do mesmo ao gerente de reparos.

O gerente de equipamento pode receber solicitações do gerente de reparos para a manutenção de um equipamento da aplicação gerenciada. Neste caso ele envia uma operação de gerenciamento para o agente. Se a solicitação for destinada ao agente, o gerente de equipamento deve atualizar sua lista de agentes como não contendo mais tal agente.

Quando um agente é reparado e inicializado, o gerente de reparo avisa o gerente de equipamento responsável e este deve então restabelecer contato com o agente.

O gerente de reparos tem a função de solucionar as falhas detectadas pelos gerentes de equipamento. Ele deve criar um relato de problema no evento da queda de um equipamento ou queda de um agente.

O gerente de reparos cria um relato de problema, escala um técnico e armazena o relato em um repositório.

Os técnicos devem consultar periodicamente o repositório a fim de achar solicitações encaminhadas para eles.

A medida que um técnico evolui na solução de um problema ele deve atualizar o estado do mesmo no relato de problema correspondente, assim o gerente de reparo poderá enviar mensagens a respeito do andamento da solução do problema para o histórico de eventos registrá-las.

Quando o relato de problemas é fechado pelo técnico, ou seja, resolvido, o gerente de reparos avisa o gerente de equipamentos sobre o reparo do equipamento ou do agente e remove o relato de problema do repositório, além de enviar uma mensagem para o histórico de eventos.

Caso o técnico escalado não consiga resolver o problema, o gerente de reparos seleciona outro técnico. Mediante informações obtidas através da tela de interação com o supervisor dos técnicos, o gerente de reparos pode selecionar o técnico mais qualificado para a solução do problema.

Através da tela de interação, o supervisor dos técnicos pode requisitar manutenção de agentes e de equipamentos para o gerente de reparo que deve escalar um técnico e abrir um relato de problema.

O histórico de eventos recebe solicitações dos gerentes de equipamentos e do gerente de reparos para registrar os eventos pertinentes ao funcionamento dos agentes e dos equipamentos da aplicação gerenciada.

Como mencionado anteriormente existe um agente para cada equipamento em operação na aplicação gerenciada.

O principal papel de um agente é atuar como uma interface entre a aplicação gerente e a gerenciada. Ele está em contanto direto com o equipamento, e na maioria dos casos em um ambiente real, é embutido no próprio equipamento.

A principal função do agente é esconder as diferenças e as particularidades de cada equipamento gerenciado, da aplicação gerente, ou seja, desacoplar as aplicações gerentes dos objetos gerenciados. Assim a comunicação entre os agentes e a aplicação gerente pode ser simplificada através da padronização de interfaces e protocolos.

Neste cenário tal comunicação será experimentada através de interações da aplicação gerente, que deverá ser desenvolvida no ambiente CORBA, com os agentes que deverá usar recursos disponíveis do ambiente CORBA-JAVA.

Os agentes devem obter dos equipamentos sinais de monitoramento como equipamento ativo ("estou vivo"), receberão ou transmitirão dados, entre outros, poderão processar e converter estes sinais quando necessários para em seguida enviar notificações sobre estes eventos.

Se em um determinado período de tempo o agente não receber nada do equipamento, ele assume a queda do mesmo e envia uma notificação para o seu gerente de equipamento.

As notificações que os agentes poderão enviar para os seus respectivos gerentes consistem em:

- Agente do terminal:

  ⇒ tentou efetuar ligação (tempo de ligação, para quem ligou, sucesso ou fracasso);

  ⇒ estou vivo;

  ⇒ Agente da central:

  ⇒ comutou ligação (quem com quem, identificar unicamente a ligação);

  ⇒ descomutou ligação (identificador da ligação);

  ⇒ capacidade física da central esgotada; estou vivo;

- Agente do provedor de serviço despertador:

  ⇒ sobrecarga (de solicitação ou de atendimento);

  ⇒ recebeu ligação (tempo de ligação, quem ligou);

  ⇒ efetuou ligação (tempo de ligação , para quem ligou, sucesso ou fracasso);

  ⇒ estou vivo;

O agente pode receber operações de gerenciamento do gerente de equipamento e executá-las sobre o objeto gerenciado. Neste caso ele deve enviar uma resposta sobre o resultado da execução da operação de gerenciamento.

Inicialmente será implementada apenas uma operação de gerenciamento possível: solicitar o desligamento do equipamento para manutenção. Neste caso o agente tenta desativar o equipamento que esta monitorando e envia uma resposta ao gerente. Então o agente se auto-desliga. Após a manutenção o técnico responsável deve colocar o equipamento e seu respectivo agente novamente no ar.

A tela de interação com o técnico permite que um técnico possa acessar os relatos de problemas atribuídos a ele e, na medida que a solução do problema evolui, possa atualizar o estado ou a situação do problema no relato de problema correspondente.

A tela de interação com o supervisor dos técnicos permite que o supervisor dos técnicos possa fornecer dados sobre os técnicos para o gerente de reparo, além de requisitar manutenção. Quando uma solicitação de manutenção é criada o gerente de reparos deve escalar um técnico e abrir uma ordem de reparo, na forma de relato de problema (para cada solicitação de manutenção).

A tela de interação com o operador da rede ou monitor permite ao operador da rede ou o monitor, acessar informações sobre os eventos que ocorreram durante a operação da rede de telefonia simulada neste cenário.

4.6. ESPECIFICAÇÃO DOS MÓDULOS

4.6.1. Descrição dos Objetos

Provedor de endereço dos equipamentos de telecomunicação: Este componente tem como objetivo implementar um provedor de endereços dos equipamentos de telecomunicações que permita aos equipamentos saber onde estão o(s) equipamento(s) ao(s) qual(is) devem se conectar. Este artifício é necessário devido à necessidade de se endereçar explicitamente processos que estão rodando em um ambiente de rede, para que se possa acessá-los. A localização deste provedor é única e conhecida por todos os equipamentos.

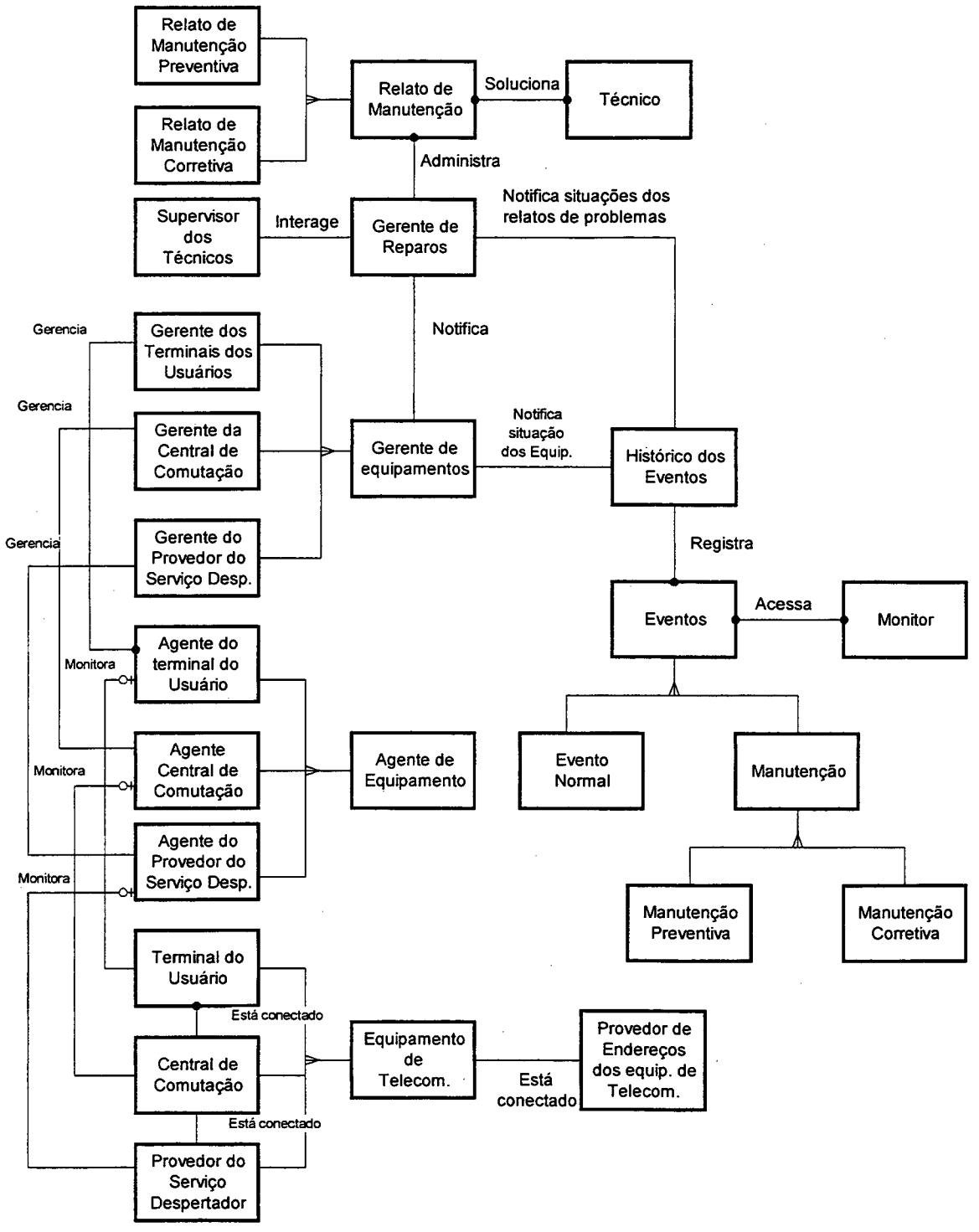


Figura 26 . Diagrama de Objetos do Sistema

Equipamentos de telecomunicação: Esta é a superclasse (meta-classe) da qual herdam as classes que constituem os três tipos de equipamentos de telecomunicação que serão simulados no sistema:

⇒ **Terminal do usuário:** Componente responsável pela simulação de um terminal telefônico, ou aparelho telefônico. Através deste componentes um usuário pode se conectar e trocar mensagens com outros terminais ou efetuar a programação do serviço despertador. Este componente apresentará uma interface que lembra o modo de operação de um telefone convencional, conforme mostrado na figura 25 da seção 4.5. Haverão **n** terminais do usuário no sistema.

⇒ **Central de comutação:** Componente responsável pela efetivação de conexão entre equipamentos que desejam se comunicar. Todos os equipamentos (terminais do usuário e provedor de serviço despertador) se conectam à central de comutação, cuja função é estabelecer circuitos temporários entre 2 terminais ou entre 1 terminal e o provedor de serviço despertador. Haverá apenas uma central de comutação no sistema.

⇒ **Provedor do serviço despertador:** Este objeto do sistema simula o SIDATA, ou seja, simula um equipamento que provê o serviço despertador, conforme descrito anteriormente. Ele será programado por usuários através dos terminais do usuário e então deverá executar os serviços "ligando" para os solicitantes na hora marcada. Haverá apenas um provedor de serviço despertador no sistema.

Além do funcionamento básico particular de cada equipamento, que foi descrito acima, os equipamentos terão em comum o fato de enviarem constantemente sinais de monitoramento para seus respectivos agentes. Estes sinais informarão os agentes sobre eventos normais de operação , assim como ocorrência de erros (ausência do sinal "estou vivo").

Agente de equipamento: Esta é a superclasse (meta-classe) da qual herdam os agentes específicos de cada um dos equipamentos da simulação de rede de telecomunicações: **agente do terminal do usuário, agente da central de comutação** e **agente do provedor de serviço despertador.**

O agente é responsável pela comunicação e ações entre a aplicação gerente e os equipamentos. Eles serão responsáveis por receber sinais de monitoramento dos equipamentos e enviá-los à aplicação gerente na forma de notificação.

Os agentes também podem receber operações de gerenciamento, executá-las sobre os equipamentos e enviar respostas à aplicação gerente.

Gerente de equipamentos: Esta é a superclasse (meta-classe) da qual herdam os gerentes específicos de cada um dos equipamentos da simulação de rede de telecomunicações: **gerente dos terminais dos usuários, gerente da central de comutação** e **gerente do provedor de serviço despertador.**

Um gerente de equipamento é responsável pelo recebimento de notificações do(s) agente(s) e por solicitar registro destas no histórico de eventos. Caso a notificação seja causada por uma situação de erro, o gerente de equipamento solicita um manutenção ao gerente de reparos, ao invés de solicitar registro da notificação no histórico de eventos.

O gerente de equipamento é, também, responsável pelo encaminhamento de operações de gerenciamento que possam ser solicitadas. Neste caso este deve aguardar resposta da operação e em seguida enviar uma resposta ao solicitante.

Gerente de reparos: Responsável pela manutenção do sistema. Caso ocorra um evento de erro, indicado por um dos gerentes de equipamentos, ou uma solicitação de manutenção, indicada pelo supervisor dos técnicos, o gerente de reparos deve abrir um **relato de manutenção** (superclasse (meta-classe)), Este consiste em uma entidade que contém informações relevantes ao andamento da manutenção de um dos equipamentos da simulação da rede de telecomunicações.

Um relato de manutenção pode ser de dois tipos (classes hereditárias):

⇒ **relato de manutenção preventiva,** que consiste em uma "ficha" que registra o andamento de uma manutenção solicitada para prevenção de defeitos em um equipamento e;

⇒ **relato de manutenção corretiva,** que consiste em uma "ficha" que registra o andamento de uma manutenção solicitada devido à defeito ocorrido em um equipamento.

Além do gerenciamento de relatos de manutenção, o gerente de reparos deve escalar um técnico que deverá efetuar a manutenção de um equipamento. Esta escalação ocorre através de informações obtidas de uma base de dados que contém informações sobre técnicos disponíveis.

Estas informações são obtidas do supervisor dos técnicos.

Finalmente, o gerente de reparos deve solicitar registro ao histórico de eventos sobre a evolução dos relatos de problema.

Técnico: Esta entidade é responsável pela interação de um técnico humano com o sistema. Através deste componente um técnico pode verificar relatos de manutenção escalados para ele e atualizá-los, conforme evolui-se a manutenção do equipamento. o técnico pode, também, requisitar uma operação de gerenciamento através deste componente.

Supervisor dos técnicos: Componente responsável pela interação do supervisor dos técnicos humano com o sistema. Através deste, o supervisor dos técnicos pode solicitar manutenção de um equipamento e entrar com dados sobre os técnicos disponíveis para manutenção do sistema.

Histórico de eventos: Este componente recebe solicitações de registro de eventos pertinentes ao funcionamento do sistema. Um **evento** (superclasse (meta-classe)) pode ser do tipo (classes hereditária):

⇒ **evento normal,** que são obtidos pelos gerentes de equipamentos através de notificações ou;

⇒ **evento manutenção,** que são obtidos de gerente de equipamentos. Um evento **manutenção** pode ser do tipo: **manutenção preventiva** ou **manutenção corretiva,** analogamente ao descrito anteriormente para as entidades relato de manutenção preventiva e corretiva.

Monitor: Entidade responsável pela interação de um monitor ou operador de rede humano com o sistema. Através deste componente o monitor pode obter informações e estatísticas sobre o funcionamento do sistema.

4.6.2. Atributos dos Objetos

Equipamento de telecomunicação:
identificador do equipamento
modelo do equipamento
fabricante do equipamento
tempo de vida útil previsto
localização do equipamento (host)
endereço (processo) do equipamento
data de inicio de operação
data da última manutenção
técnico da último manutenção

Terminal do usuário:
identificação do assinante
identificador (número e prefixo) do terminal do usuário

Central de Comutação:
capacidade física máxima
capacidade física utilizada

Provedor do serviço despertador:
capacidade para atendimento de requisições ao mesmo tempo
capacidade para execução das requisições ao mesmo tempo

Provedor de endereços dos equipamentos de telecomunicações:

Agente de equipamentos:
identificador do equipamento monitorado
localização do agente (host)
endereço do agente (processo)
data da última manutenção
técnico da última manutenção

Agente de terminal do usuário

Agente de central de comutação:

Agente de provedor do serviço despertador:

Gerente de equipamentos:

Gerente dos terminais dos usuários:

Gerente da central de comutação:

Gerente do provedor do serviço despertador:

Gerente de reparos:

Histórico de eventos:
data de início do log

Supervisor dos técnicos:
identificação do supervisor dos técnicos

Monitor:
identificação do operador de rede

Técnico:
identificação do técnico
telefone para contato
endereço para contato

Relato de manutenção:
data de criação do relato de problema
horário de criação do relato de problema
data de início do reparo/manutenção do relato de problema
horário de início do reparo/manutenção do relato de problema
data de término do reparo/manutenção do relato de problema
horário de término do reparo/manutenção do relato de problema
grau de prioridade
identificação do técnico responsável
estado do relato do problema (enfileirado, aberto, deferrido, fechado)
localização do equipamento (host)

Relato de manutenção preventiva:
identificação do requisitante
justificativa da atividade de manutenção
descrição da atividade de manutenção
data prevista para início da manutenção
horário previsto para inicio da manutenção
data prevista para término da manutenção
horário previsto para término da manutenção

Relato de manutenção corretiva:
entidade que identificou o problema
tipo da falha
descrição da falha
causa provável da falha

descrição da atividade de reparo

Eventos:
data e hora de inicio da operação
data e hora de término da operação
equipamento

Evento normal:
tipo de operação (recepção, transmissão, inativo)
equipamento origem/destino

Evento manutenção:
identificação do técnico responsável
estado do relato de problema

Evento manutenção corretiva:
identificador da entidade que percebeu o problema

Evento manutenção preventiva:
identificação do solicitante da atividade de manutenção

4.6.3. Operações dos Objetos

Equipamentos de telecomunicações:
incializar (estabelecer contato com prov. end. e inicializar os processos)
enviar sinais de monitoramento para o agente

Terminal do usuário:
solicita ligação
recebe ligações
recebe mensagens da central de comutação
exibe mensagens para o usuário
recebe mensagens do usuário
envia mensagens para a central de comutação
finaliza ligação

Central de Comutação:
estabelece ligação
retorna erro (destinatário inexistente ou ocupado)
Provedor do serviço despertador:
aceita/recusa serviço
programa o serviço
monitora os horários solicitados

executa o serviço despertador

Provedor de endereços dos equipamentos de telecomunicações:
inicializar provedor de endereços
recebe endereços
fornece (distribui) endereços para os equipamentos

Agente de equipamento:
enviar notificação para o gerente
recebe operação de gerenciamento do gerente e enviar resposta
executar operação de gerenciamento

Agente de terminal do usuário:
monitorar terminal do usuário

Agente de central de comutação:
monitorar central de comutação

Agente de provedor do serviço despertador:
monitorar provedor do serviço despertador

Agente de provedor de endereços dos equipamentos de telecomunicações:

Gerente de equipamentos:
enviar operação de gerenciamento para o agente
solicita registro de evento para o histórico de eventos (notifica situação dos equipamentos)
solicita reparo para o gerente de reparos
monitora agente

Gerente dos terminais dos usuários:

Gerente da central de comutação:

Gerente do provedor do serviço despertador:

Gerente do provedor de endereços dos equipamentos de telecomunicações:

Gerente de reparos:
escalar técnico
criar relato de problema
administrar (monitorar) repositório de relato de problemas
solicita registro de evento para o histórico de eventos (notifica estado do relato de problemas)
notifica o gerente de equipamento responsável sobre a conclusão de reparo

consulta cadastro de técnicos

Histórico de eventos:
registra evento

Supervisor dos técnicos:
recebe/atualiza informações dos técnicos
recebe solicitação de manutenção

Monitor:
acessa repositório de eventos
apresenta informações e estatísticas

Técnico:
atualiza o relato de problema
exibe relatos de problemas atribuídos para o técnico

Relato de manutenção:

Relato de manutenção preventiva:

Relato de manutenção corretiva:

Eventos:

Evento Normal:

Evento Manutenção:

Evento de manutenção preventiva:

Evento de manutenção corretiva:

4.6.4. Modelagem Dinâmica

4.6.4.1. Diagrama de Eventos

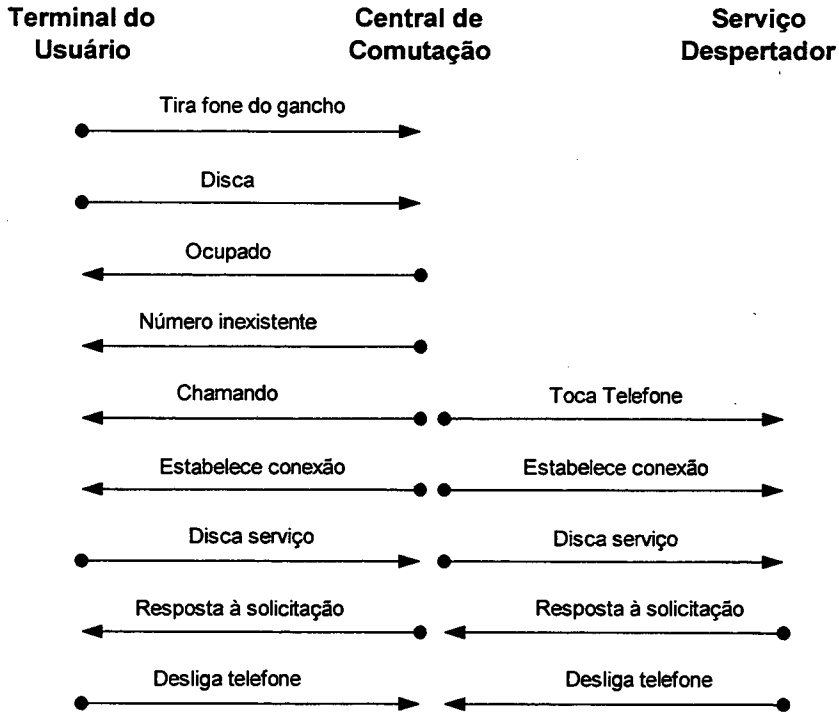


Figura 27. Troca de eventos do term. do usuário - chamador, central de comutação e serviço despertador - receptor.

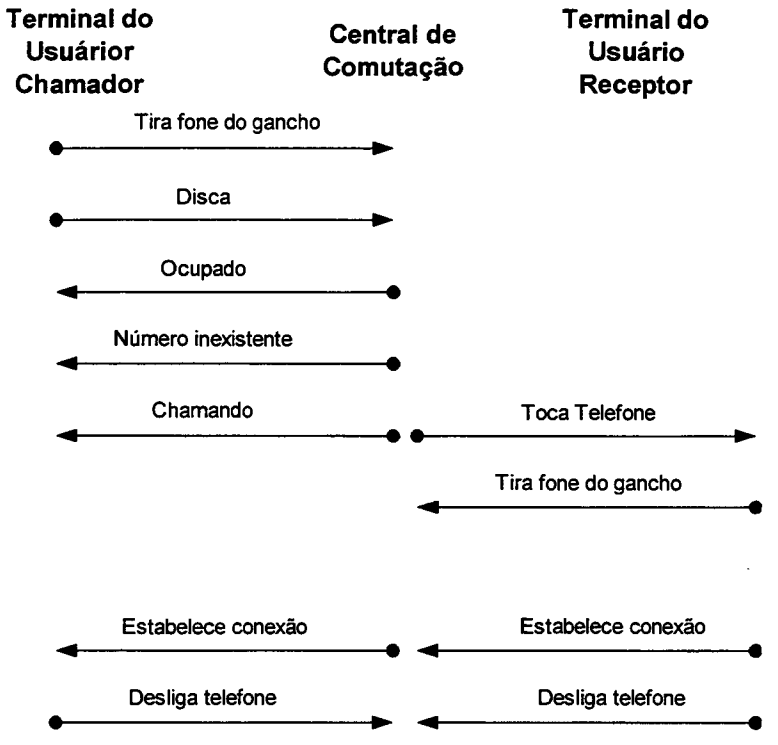


Figura 28. Troca de eventos entre dois terminais do usuário e central de comutação

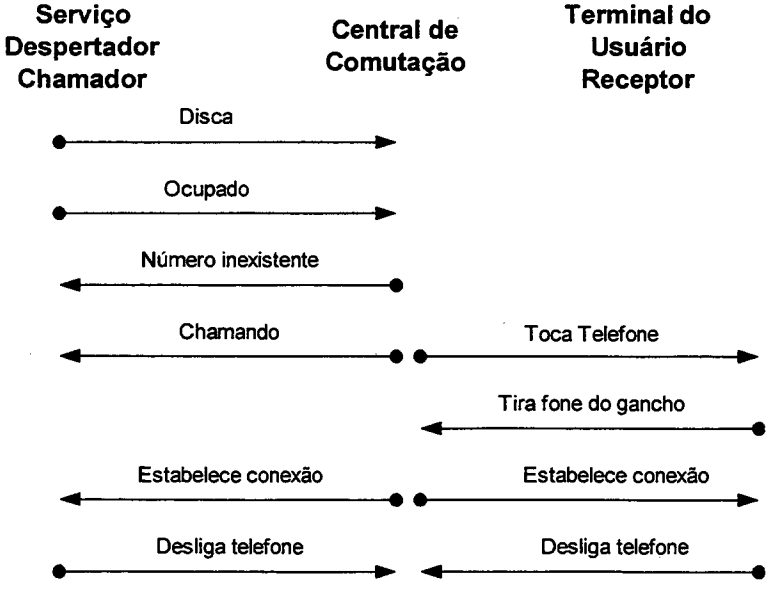


Figura 29. Troca de eventos entre serviço despertador - chamador,central de comutação e terminal usuário - receptor.

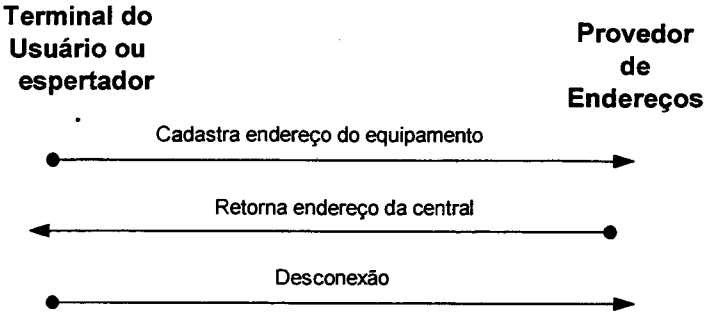


Figura 30. Troca de eventos entre terminal do usuário ou serviço despertador e provedor de endereço.

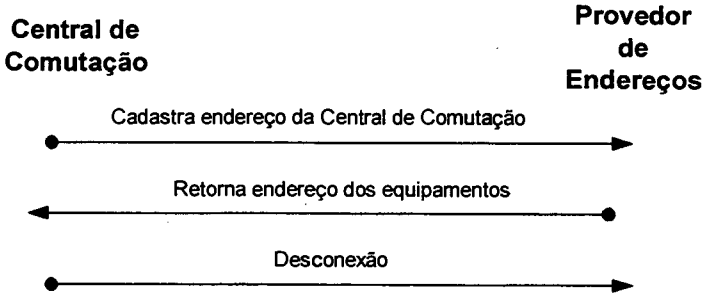


Figura 31. Troca de eventos entre central de comutação e provedor de endereços.

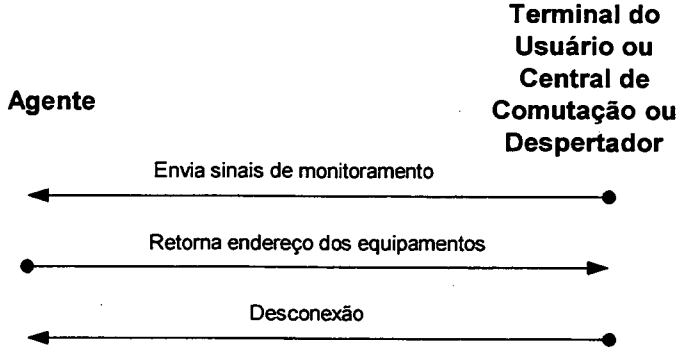


Figura 32. Troca de eventos entre agente e equipamentos.

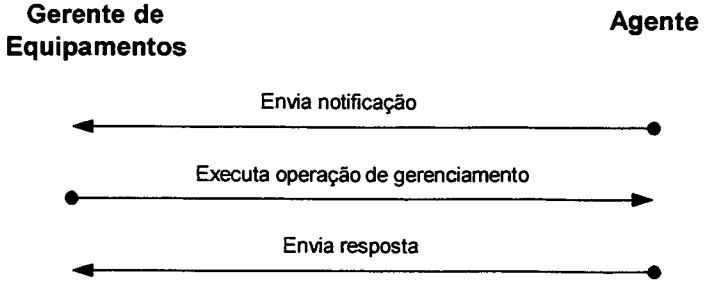


Figura 33. Troca de eventos entre gerente de equipamentos e agente.

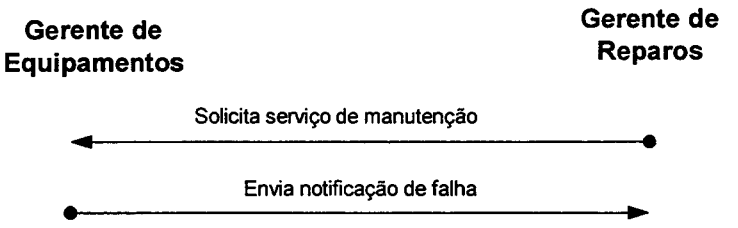


Figura 34. Troca de eventos entre gerente de equipamentos e gerente de reparos.

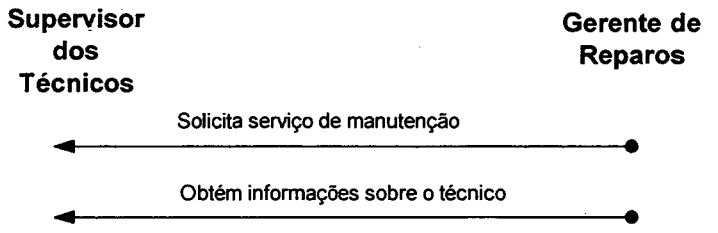


Figura 35. Troca de eventos entre supervisor dos técnicos e gerente de reparos.

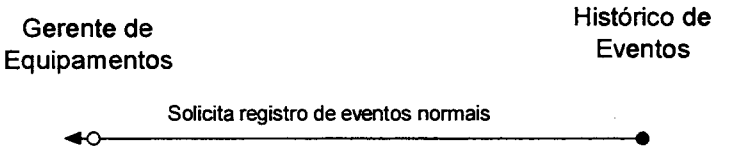


Figura 36. Troca de eventos entre gerente de equipamentos e histórico de eventos.

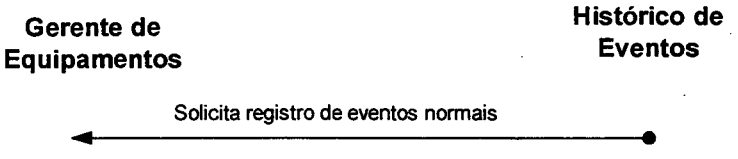


Figura 37. Troca de eventos entre gerente de reparos e histórico de eventos.

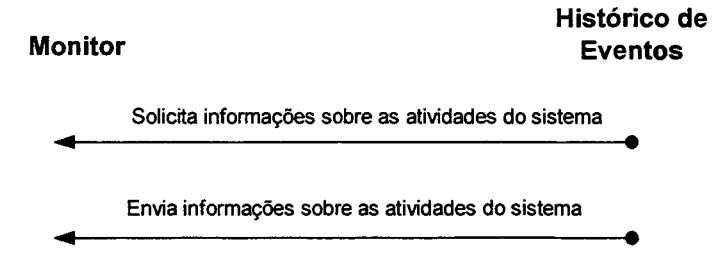


Figura 38. Troca de eventos entre monitor e histórico de eventos.

4.6.4.2. Diagramas de Estados

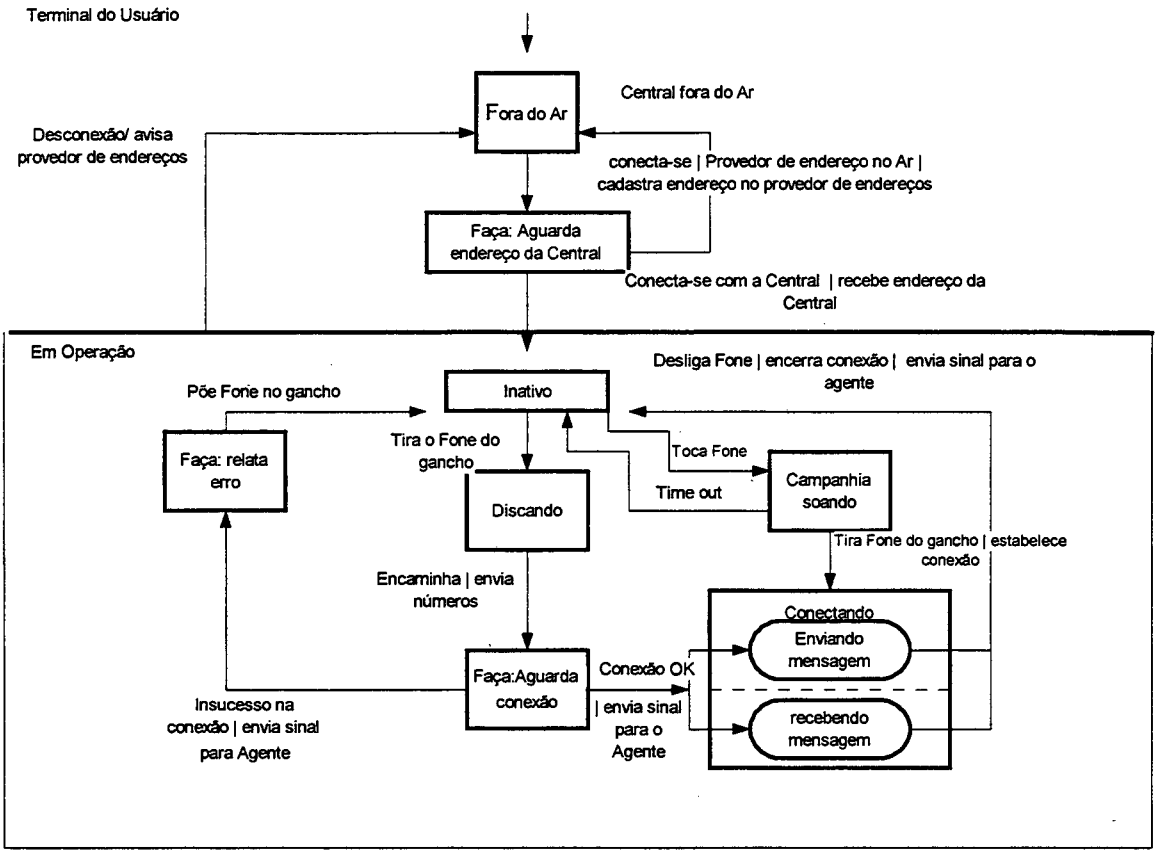


Figura 39. Diagrama de Estados do Terminal do Usuário.

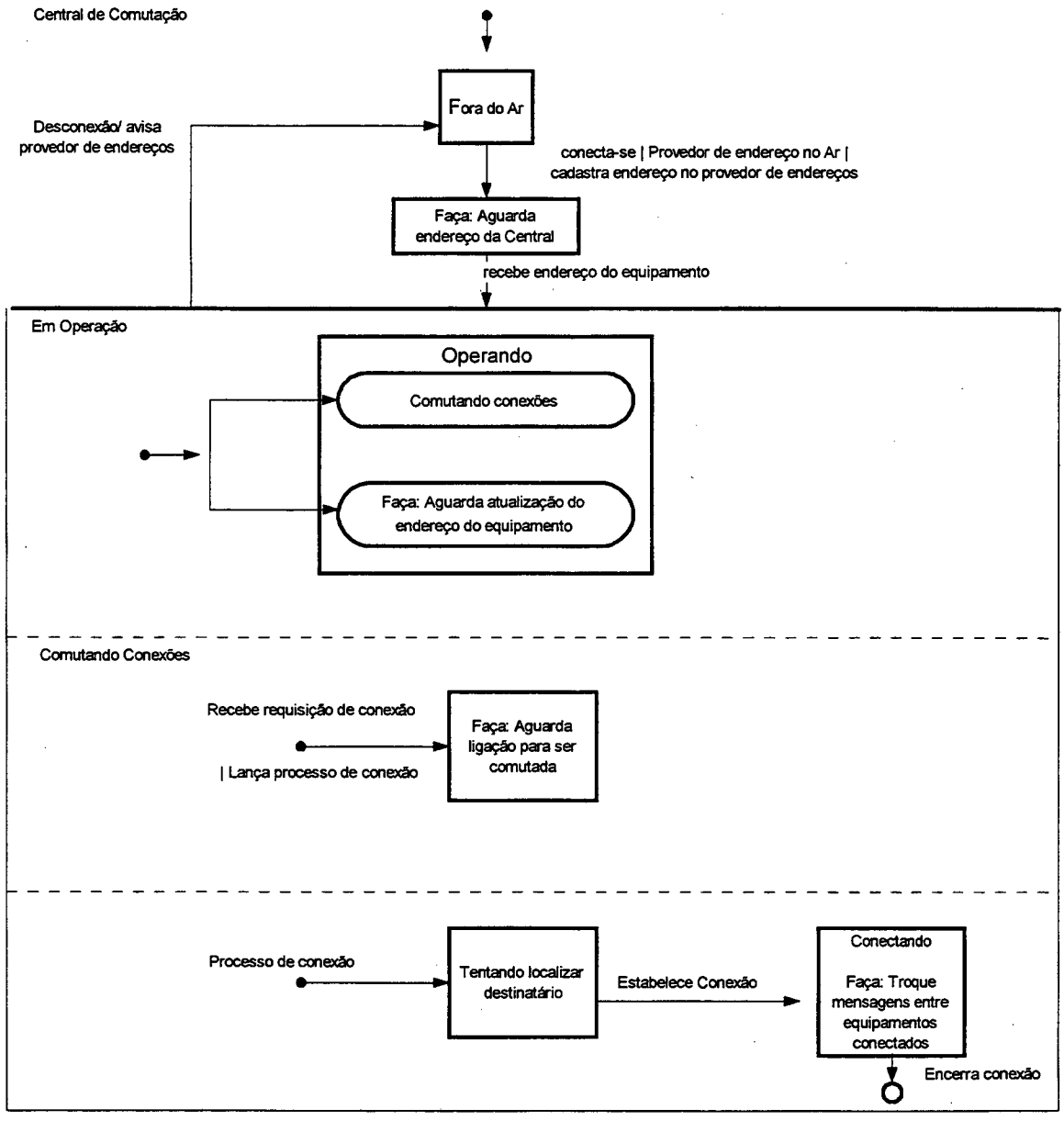


Figura 40. Diagrama de Estados da Central de Comutação.

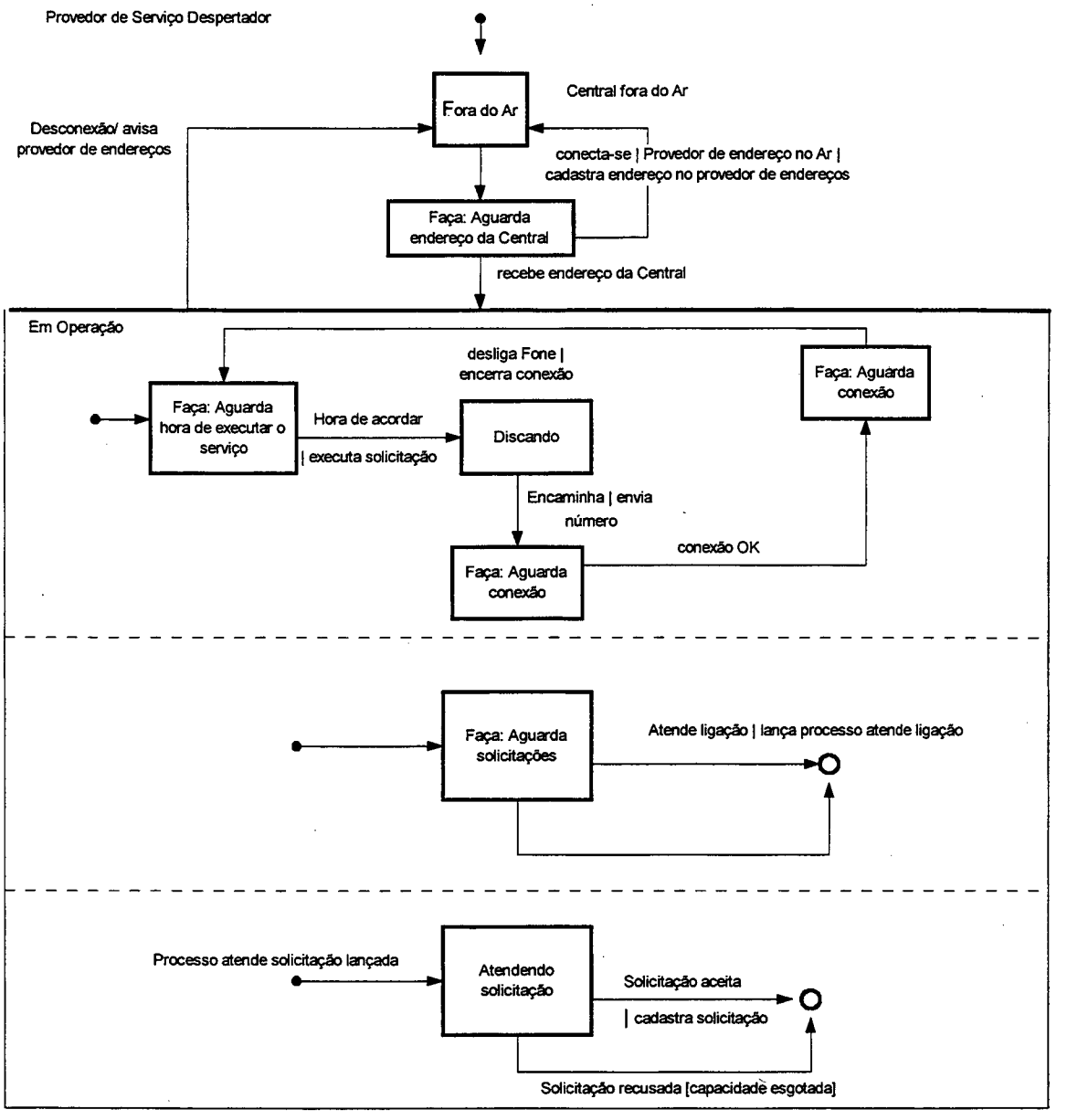


Figura 41. Diagrama de Estados do Provedor de Serviço Despertador

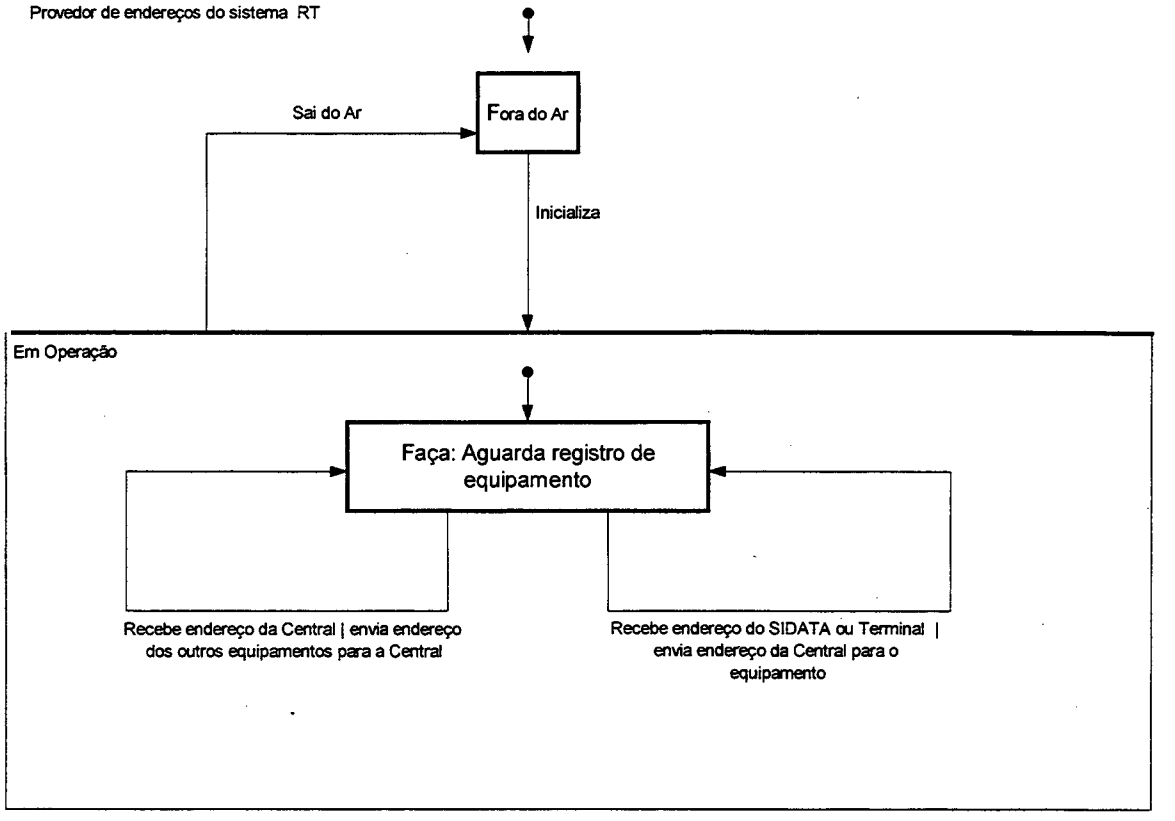


Figura 42. Diagrama de Estados do Provedor de Endereços do Sistema RT.

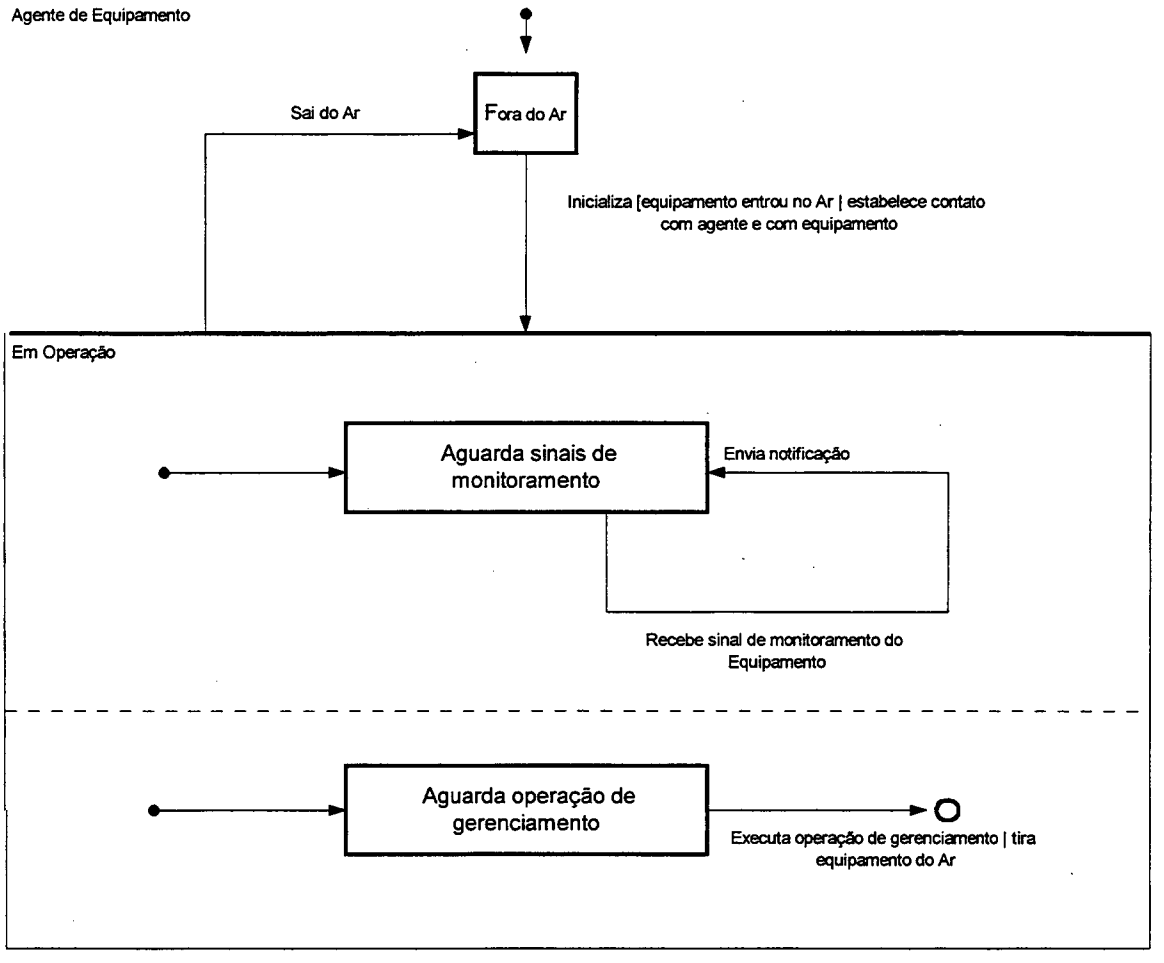


Figura 44. Diagrama de Estados do Agente de Equipamento.

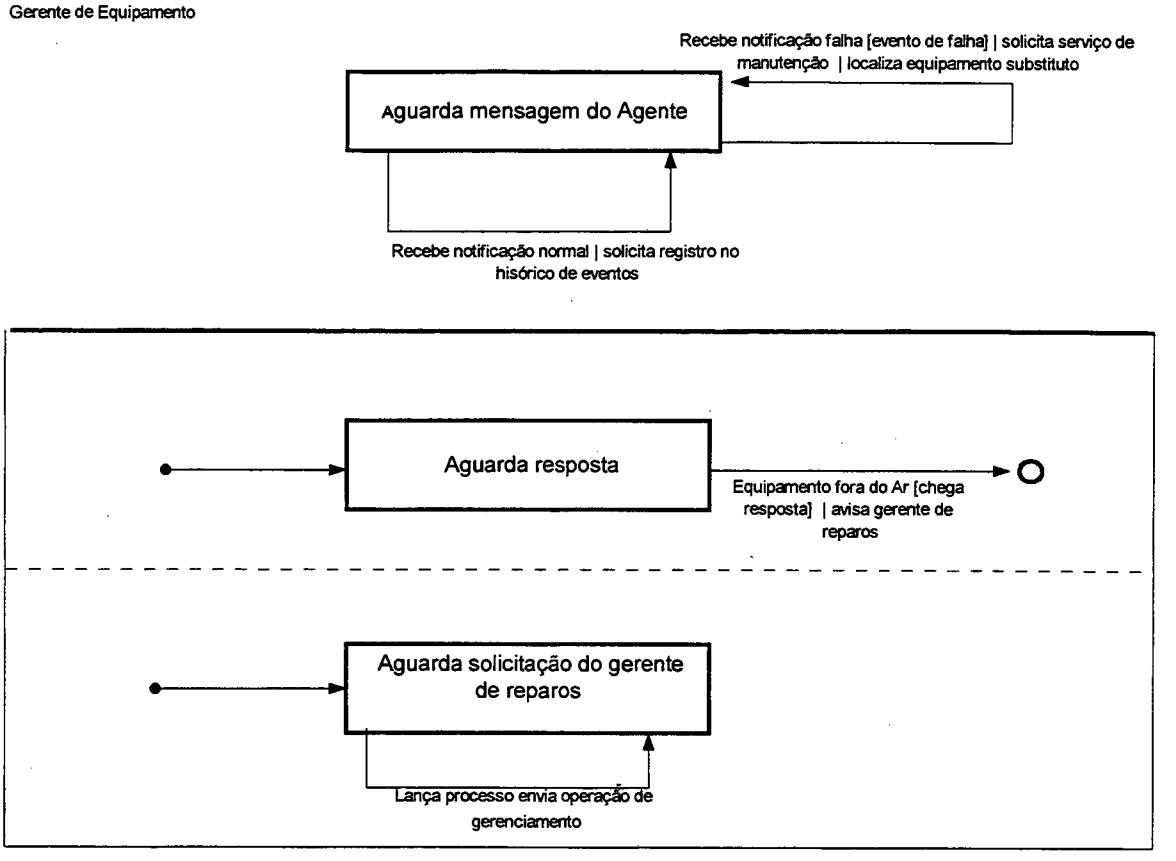


Figura 44. Diagrama de Estados do Gerente de Equipamentos.

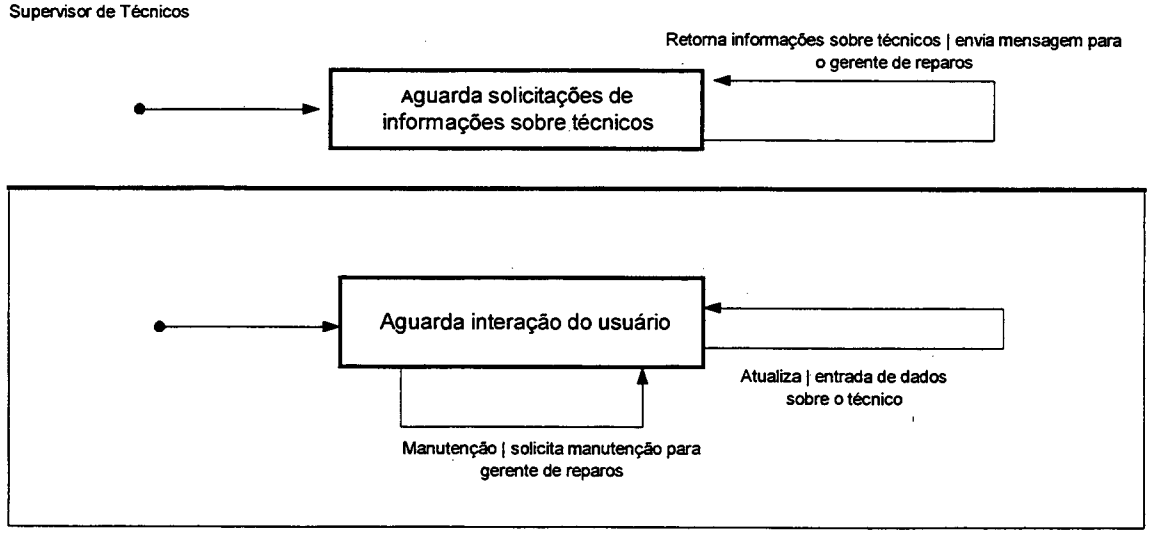


Figura 45. Diagrama de Estados do Supervisor dos Técnicos.

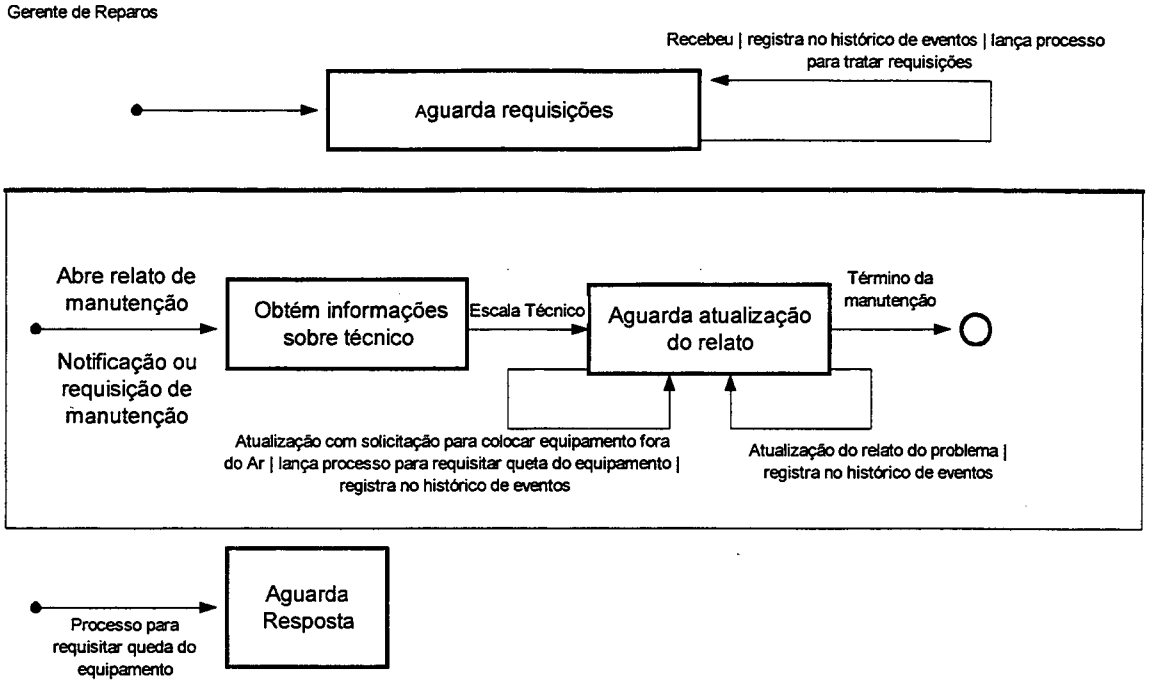


Figura 46. Diagrama de Estados do Gerente de Reparos.

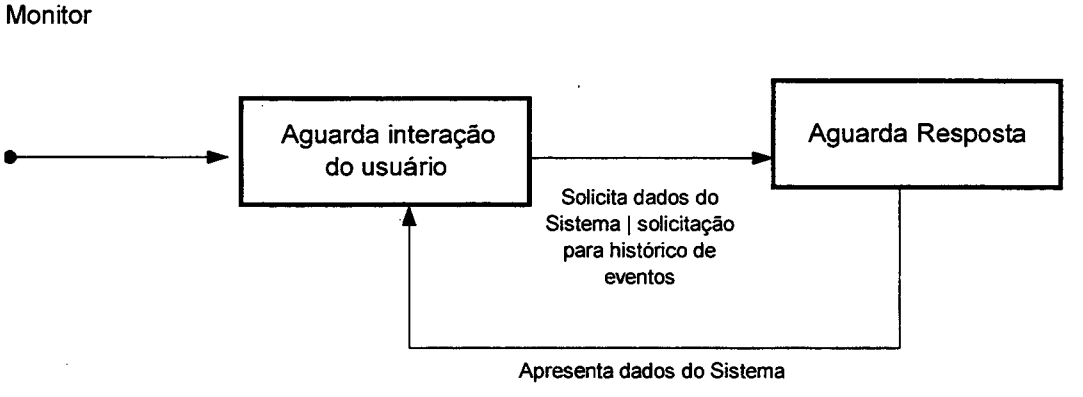


Figura 47. Diagrama de Estados do Monitor.

4.6.4.3. Modelagem Funcional

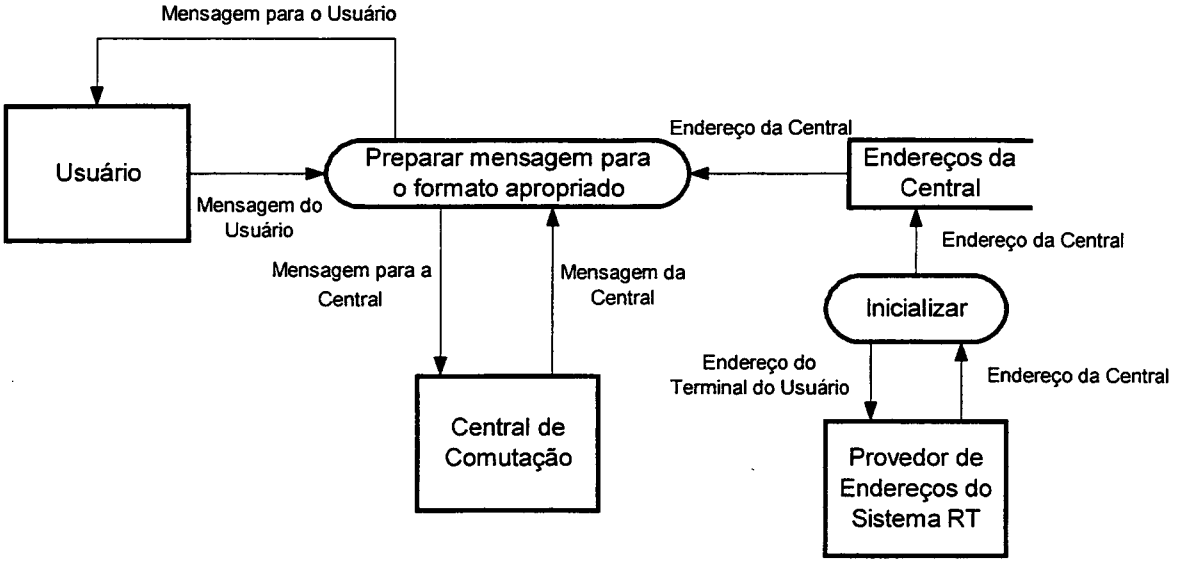


Figura 48. Diagrama de Fluxo de Dados do Terminal do Usuário.

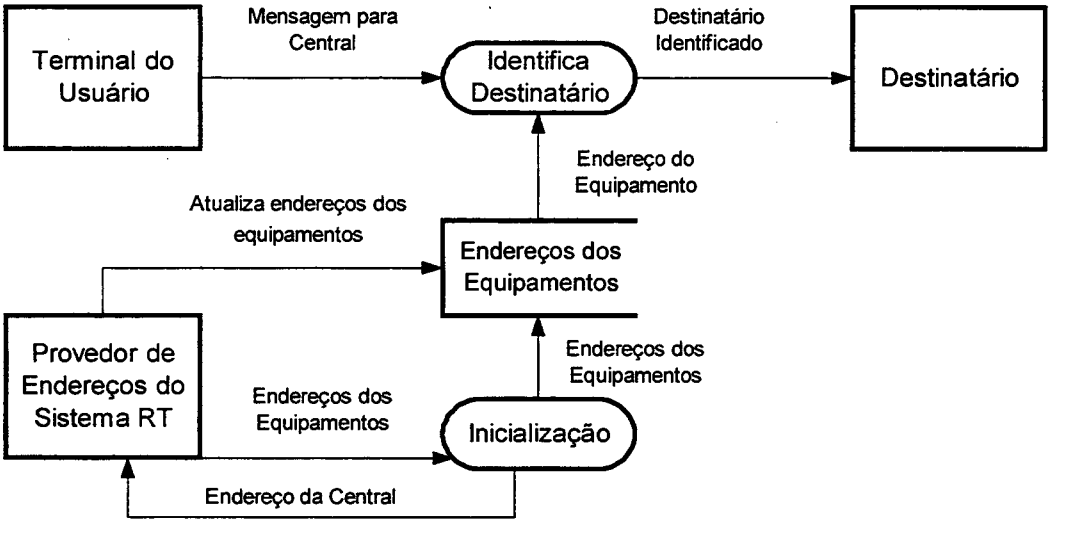


Figura 49. Diagrama de Fluxo de Dados da Central de Comutação.

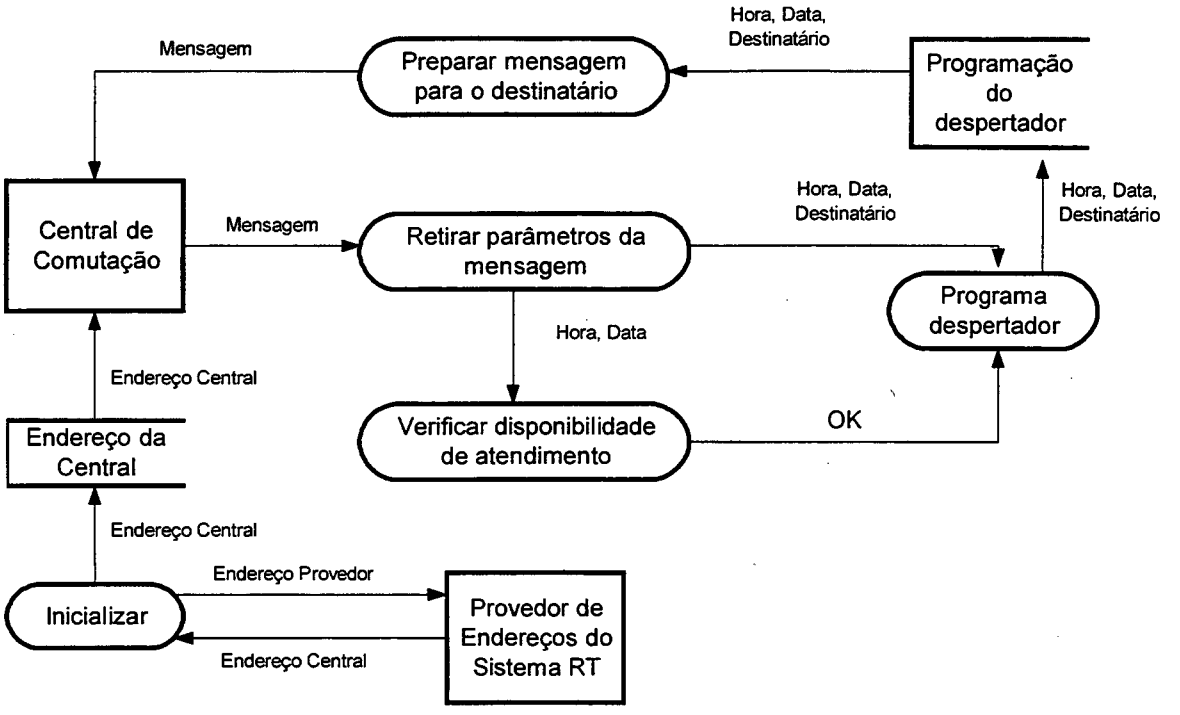


Figura 50. Diagrama de Fluxo de Dados do Provedor do Serviço Despertador.

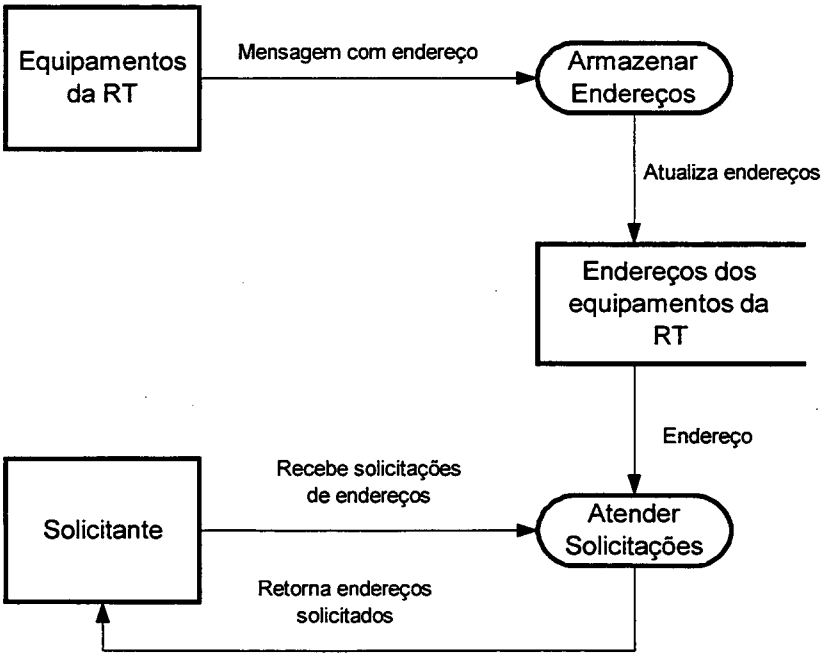


Figura 51. Diagrama de Fluxo de Dados do Provedor de End. do Sistema RT.

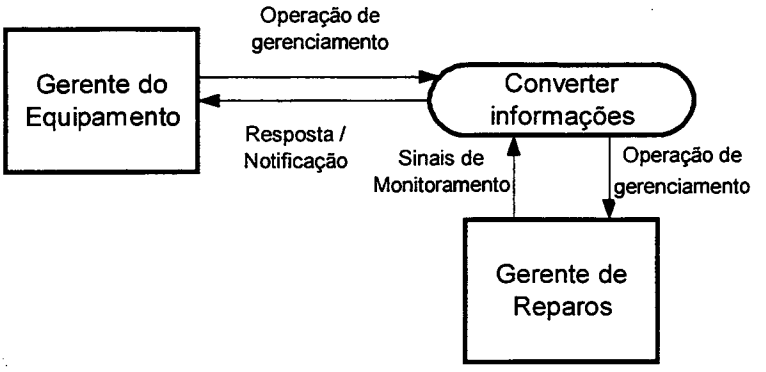


Figura 52. Diagrama de Fluxo de Dados do Provedor do Agente de Equipamentos.

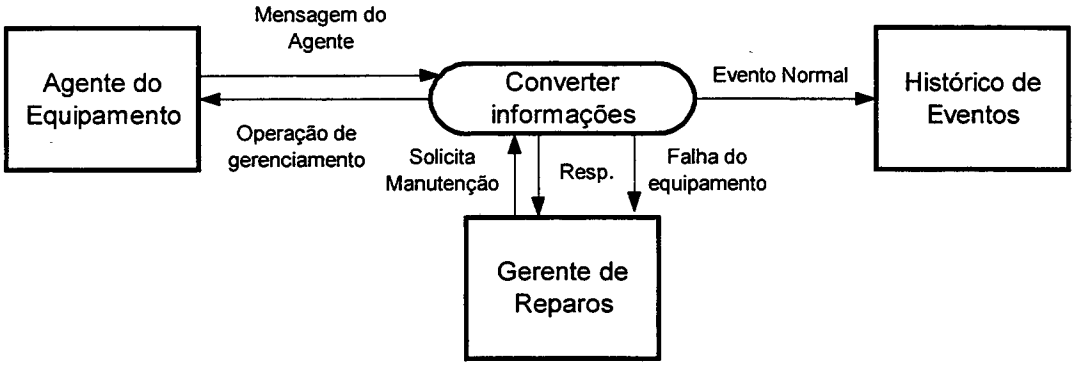


Figura 53. Diagrama de Fluxo de Dados do Gerente de Equipamento.

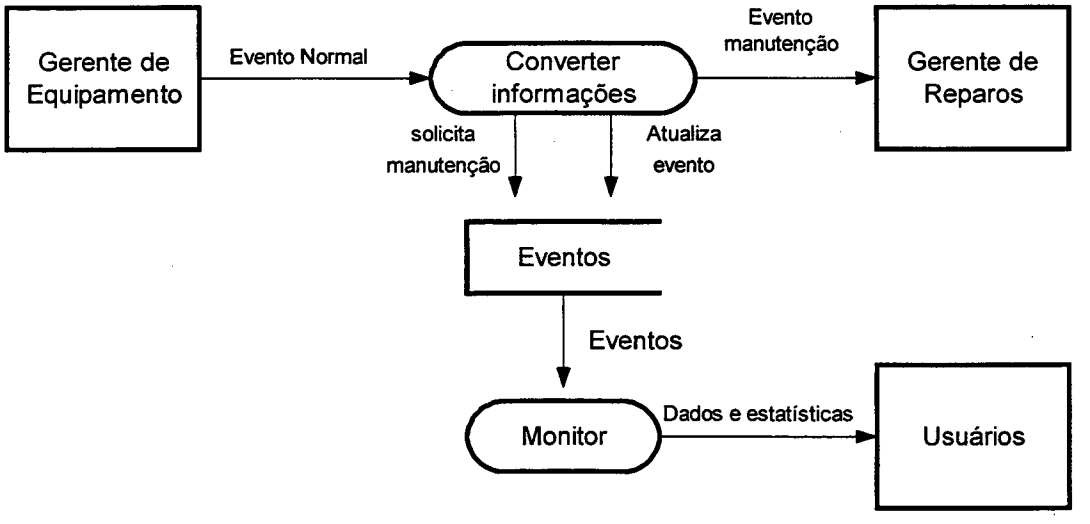


Figura 54. Diagrama de Fluxo de Dados do Histórico de Eventos e Monitor.

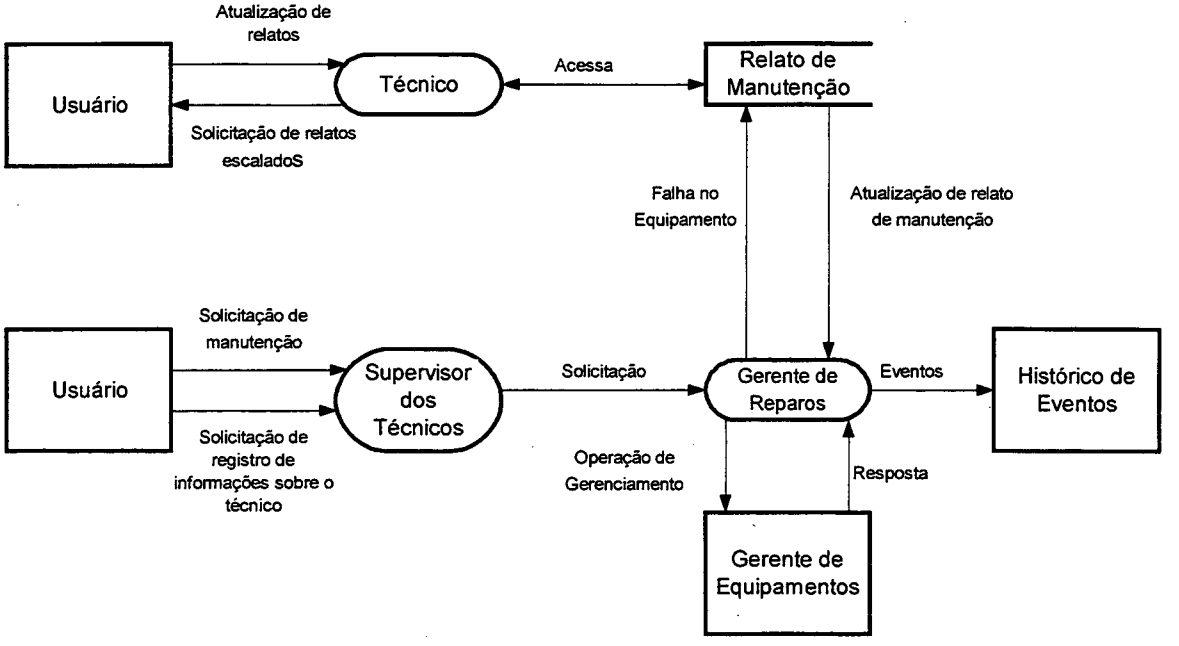


Figura 55. Diagrama de Fluxo de Dados do Gerente de Reparos, Técnicos e Supervisor dos Técnicos.

5. IMPLEMENTAÇÃO DO MODELO UTILIZANDO CORBA, JAVA E REFLEXÃO COMPUTACIONAL

5.1. Introdução

Este capitulo implementa o modelo de gerenciamento de serviços de telecomunicações, utilizando a linguagem JAVA sobre uma plataforma CORBA, com utilização do paradigma de Reflexão Computacional sobre a WEB. Como exemplo, é utilizado o serviço despertador automático.

5.2. Por que JAVA, CORBA e Reflexão Computacional ?

As principais razões para utilização de JAVA são:

- **Portabilidade de aplicações independente de plataforma**: as aplicações JAVA são altamente portáveis, devido a representação padronizada de byte-code, geradas pelos compiladores JAVA, independente de máquina e de sistemas operacionais. Isto significa, um avanço sobre as outras linguagens de programação, em particular para aplicações cliente, uma vez que um único conjunto de código fonte ou de *byte-code* compilado, será utilizado em qualquer plataforma. Consequentemente, o custo de manutenção e desenvolvimento pode ser significativamente reduzido.
- **Programação na *Internet***: as ligações da Linguagem JAVA permite implementação dos clientes CORBA como *applets*. Isto habilita acesso a objetos CORBA, e potencialidade para herdar aplicações encapsuladas em objetos, utilizando os *Browsers* populares. Na realidade os *Browsers* que oferecem o JAVA estão tornando-se uma *Universal Graphical User Interface* (GUI). Em uma empresa a mesma tecnologia pode ser utilizada em *Intranets*, uma vez que o protocolo TCP/IP é utilizado. Embora seja útil possuir *applets* que sejam somente

clientes para objetos CORBA, *applets* também podem implementar objetos CORBA. Esta abordagem é limitada devido ao modelo de *applet sandbox,* o qual desabilita o acesso das *applets* a recursos da máquina onde elas são executadas, contudo, as *applets* podem fornecer interfaces **CallBack** (uma chamada de um servidor para um cliente, invertendo a regra Cliente/Servidor).

- **Linguagem de programação orientada a objeto**: Java oferece as mesmas facilidades que as principais linguagens C, C++, SmallTalk, oferecem. Java oferece construções que as linguagens C e C++ não oferecem, ou seja coletor de lixo automático, manipulação de exceção e suporte a *Threads.* Estas características são muito usadas na programação de sistemas distribuídos.

As principais razões para utilização do CORBA são:

- **Definir interfaces independente de implementação:** A OMG IDL oferece um meio de separar a interface, da implementação dos objetos. Esta separação é muito usada em Engenharia de Software. Sistemas baseados em metodologia e ferramentas Orientadas a Objetos, bem como Técnicas de Modelagem de Objetos (OMT), e a Linguagem de Modelagem Unificada (UML), podem ser escrita com OMG IDL. Uma vez que interfaces são especificadas em IDL, um desenvolvedor ou um grupo de desenvolvedores pode implementar partes diferente do sistema. A separação da interface da implementação também é bastante usada para gerenciar componentes de software. A IDL permite acesso a múltiplas implementações, podendo ainda ser estendida por herança, onde são substituídas por interfaces-base.

- **Independência de linguagem de Programação:** O CORBA fez o mapeamento da IDL para várias linguagens, de maneira que parte do sistema ou de aplicações podem ser implementadas em diferentes linguagens desde de que a mesma esteja mapeada, pois, toda interação em uma aplicação acorre através da interface que é especificada independente da linguagem de programação que a implementará.

Anteriormente, aplicações distribuídas eram implementadas em uma linguagem particular por causa da disponibilização de bibliotecas de acesso remoto da linguagem. Com CORBA escolhe-se a linguagem mais apropriada para implementar o objeto, baseando-se na integração, na experiência e no domínio da equipe de desenvolvedor com a linguagem, ou a linguagem conveniente para implementar a semântica do objeto.

- **Acesso a Objetos independente de sua localização:** Aplicações distribuídas são baseadas em URL ou *Socket*, pois, precisam endereçar o Servidor especificando um *host name* e o número de uma porta de comunicação. Em contraste, CORBA disponibiliza localização transparente, dos métodos independentes do local físico a onde estejam, podendo ser trocado de local sem interromper a aplicação que o utiliza. CORBA ainda disponibiliza mecanismo para recomeçar o serviço solicitado pelo Cliente. Isto pode ser controlado pelo policiamento de vários sistemas servidores.

- **Geração automática de código tratando invocações remotas:** Sistemas distribuídos exigem um esforço de programação de baixo nível incluindo abertura, controle e fechamento de conexão em rede; *marshaling e unmarshaling* ; e um esforço para estabelecer escuta de requisições pela portas de *sockets,* além de solicitar a implementação do objeto. Compiladores IDL e sistemas que utilizam o ORB, deixam o programador livre destas tarefas. Eles criam representações de definições elaboradas em IDL, tais como, constantes, *data types*, e interfaces em uma particular linguagem de ligação, por exemplo, C++ ou JAVA. Também criando o código para o *marshaling e unmarshaling* para os tipos de dados definidos pelo desenvolvedor. Bibliotecas são disponibilizadas para suportarem os tipos predefinidos pelo CORBA.

  O código gerado para o cliente, isto é, o código que invoca uma operação sobre um objeto, é conhecido como *stub*. O código gerado para o servidor, o qual invoca os métodos da implementação que se quer operar, é chamado de código *skeleton.*

O código *skeleton* em conjunção com o ORB fornece um mecanismo transparente em tempo de execução do aplicativo, manuseando as solicitações que chegam e administrando as conexões de rede associadas aos objetos.

- **Acesso aos serviços e facilidades do modelo CORBA:** O ORB oferece um meio para chamada distribuída e transparente a métodos de objetos remotos. Normalmente, aplicações distribuídas complexas, necessitam funcionalidades adicionais. Dentro da OMG estas necessidades tem sido analisadas e conduzidas para a especificação de serviços fundamentais. Estes serviços são publicados com o produto *CORBAService*. Por exemplo:
  - ***Naming Service*** - serviços nomeação ou referência para objetos distribuídos;
  - ***Trading Service*** - serviços negociação para objetos distribuídos;
  - ***Event Service*** – serviços assíncronos, de mensagem baseados em assinatura;
  - ***Transaction Service*** – processamento de transações para objetos distribuídos;
  - ***Security Service*** - disponibilização de autenticação, autorização, criptografia e outras características de segurança.

A principal razão para utilização de Relexão Computacional :

Este paradigma facilita a replicação e o controle dos objetos utilizados pelos múltiplos clientes dos serviços oferecidos em uma plataforma CORBA. Cada serviço solicitado será mapeado sob a forma de um objeto-base. Os protocolos de coordenação são implementados na forma de meta-objetos. Estas entidades na verdade, nada mais são do que o conjunto de *stubs* no cliente e no servidor, *stubs* para comunicação dos objetos e o BOA (Basic Object Adapter) com suporte para o gerenciamento do grupo de objetos (bibliotecas de objetos, repositório de implementações), gerados todos a partir da tradução da especificação IDL da interface de um objeto-servidor. O servidor é implementado através de um conjunto de métodos da classe servidor, gerada na compilação da IDL, ficando o programador responsável pela inserção dos códigos de cada método da classe. No processo

de compilação das interfaces, a plataforma CORBA gera automaticamente todo o suporte para a comunicação (*stubs*) entre as entidades envolvidas, incluindo também as funcionalidades para gerenciamento de grupos de objetos. O compilador também gera o arquivo contendo o "esqueleto" do código do servidor (declaração de variáveis e métodos). O programador então, fica responsável pela descrição dos objetos-base da aplicação e dos protocolos de coordenação, preenchendo os corpos dos métodos definidos nas interfaces dos meta-controladores. Com este esquema de implementação, os objetos-base cliente e servidor permanecem isentos de quaisquer atividades que não estejam ligadas à aplicação propriamente dita. Todos os aspectos relativos ao gerenciamento do objeto e às interações no contexto CORBA ficam concentrados ao nível de meta-controladores. Neste contexto, o cliente apresenta-se estruturado como um cliente-base que representa o comportamento da aplicação, e um meta-cliente, que não possui função ativa, mas que pode ser usado no gerenciamento de um determinado número arbitrário de clientes, ou para implementar mecanismos de tratamento de exceções no cliente. A estrutura do servidor é semelhante a do cliente, sendo que o serviço, ou seja o objeto que se deseja replicar, é um meta-controlador ou meta-servidor, responsável pelo protocolo de coordenação.

Como exemplo veja a Figura 56, que implementa uma parte do gerenciamento de serviços, o qual controla todos os clientes e suas solicitações e também mostra, através de referência (CORBA *objectreference*), que um cliente pode passar a ser servidor, dependendo da necessidade, fazendo com que o cliente haja sobre ele mesmo. Neste contexto o **Client** apresentar-se-á estruturado em um cliente-base que representa o comportamento da aplicação, e um meta-cliente, que não possui função ativa, mas que pode ser usado no gerenciamento de um número determinado de clientes e o objeto **MultiCoordinator** será um meta-controlador (meta-servidor), responsável pelo protocolo de coordenação. O código a seguir está escrito em IDL e retrata o modulo *Client* e *MultiCoordinator*, mostrando suas interfaces com suas respectivas operações.

```
module Client
{ interface ClientControl
  { // start and stop operations for controlling client counting
    boolean start();
    string  stop();
  };
};
module MultiCoordinator
{ interface Coordinator
  { // object for registering and controlling client counting
    boolean register(in string clientNumber,
                     in Client::ClientControl clientObjRef);
    boolean start();
    string  stop();
  };
};
```

5.3. Implementação dos módulos

Os serviços serão objetos CORBA, ou seja, terão interface definida em IDL de modo a serem acessíveis por qualquer objeto e também acessam qualquer outro objeto CORBA, independentemente da linguagem de programação. Os serviços serão identificados por sua referência a objetos e assim localizados pelo *brokers*. Ver figura 56.

Nesta etapa, foram analisados alguns requisitos para implementação de sistemas na WEB:

Os **objetos gerenciáveis** foram modelados utilizando-se técnicas de orientação a objetos. Quanto a gerência destes objetos, por serem serviços processados em tempo real, ou seja, OLTP (*OnLine Transaction Processing*), executam simultaneamente e repetidamente um conjunto predefinido de declarações SQL (*Structured Query Language*).

Os clientes devem requisitar um serviço, fazer suas coisas, e então partirem. A função básica da OLTP é a reutilização de recursos. Não é permitido a ninguém monopolizar recursos escassos. Todo mundo deve aprender como compartilhar.

Nas últimas três décadas, os projetistas de OLTP dominaram a arte de comprimir ao máximo a performance fora de seus sistemas. Eles sabiam como escrever aplicações que escalonavam. A medida importante para os projetistas de aplicações OLTP eram *thoughput*, tempo de resposta , e o número de usuários concorrentes.

A seguir está a lista do que possibilitamos medir na aplicação de Gerência de Serviços de Telecomunicações:

- ***Benchmark* dos serviços,** o *Benchmark* dos serviços é um modelo hipotético para uma aplicação de serviços de telecomunicações oferecidos na WEB. Ele define o padrão de transação chamado *User*, uma base de dados padrão chamada **Mib_Servico**, um método escalonável para grandes sistemas, e uma medida de *throughput* e de preço por performance;

- **Atualizações em tabelas** da base de dados, para melhorar o atendimento e a *performance* do servidor de serviços;

- **Relatório de Eventos** para tomada de decisões, caso os eventos sejam críticos e para ativar técnicos no sentido de realizarem manutenções corretivas ou preventivas a equipamentos que apresentam falhas (a ativação de técnicos para manutenção não foi implementada).

Algumas definições são destacadas:

1. *Throughput*: esta é a taxa na qual o sistema OLTP pode processar transações. Medimos o *throughput* em transações por segundo (tps). A medida que adicionamos usuários, o sistema alcança o máximo *throughput* e então começa a cair, a medida que mais usuários entrarem no sistema. Um bom sistema OLTP deve ser capaz de manter o *throughput* estável, quando adicionarmos mais usuários.

2. *Response Time*: este é o tempo transcorrido desde o momento em que você submete a transação até o momento em que você receber a resposta. O tempo de resposta deve crescer linearmente, não exponencialmente, a medida que acrescentamos mais usuários ao sistema.

3. *Concurrent users*: o número de usuários concorrentes é um fator importante nos sistemas OLTP. *Throughput* e tempo de resposta são significativos quando os

medimos em função do número de usuários. Devemos também medir a *performance* do sistema em momentos de pico. O gerente certamente estará interessado em números de *performance* de pico .

4. A definição de transação: para a unidade tps ser significativo é importante que concordemos o que exatamente é uma transação. Uma transação varia de aplicação para aplicação. Uma transação que reserva um lugar numa linha aérea é obviamente diferente de uma transação de chão de fábrica. Nosso *beachmark* deve ser capaz de especificar exatamente o que é uma transação. Por exemplo, devemos descrever ambos, os comandos reais SQL e também como aplicamos, *commits*, *logs*, e *locks*.

5. *System steady_state verification* (Verificação com o sistema estabilizado): Um sistema alcança o *steady state* quando sua *performance* fica constante de um determinado ponto em diante. A *performance* do sistema pode variar 20% ou mais antes de atingir o *steady_state*. Podemos atribuir esta variação a diversos fatores, incluindo *buffers* que enchem e ao *swapping* de páginas da memória para disco, a medida que a carga do sistema aumenta. Devemos relatar os resultados dos *beachmarks* quando o sistema estiver nas condições de *steady_state* na tabela de eventos.

6. *Cost per transaction* (custo por transação): o custo do sistema deve ser usado para normalizar resultados de testes de tps através de sistemas amplamente diferentes, por exemplo, um PC de $3000 versos um *Mainframe*. Isto significa que devemos mensurar o custo por *tps*.

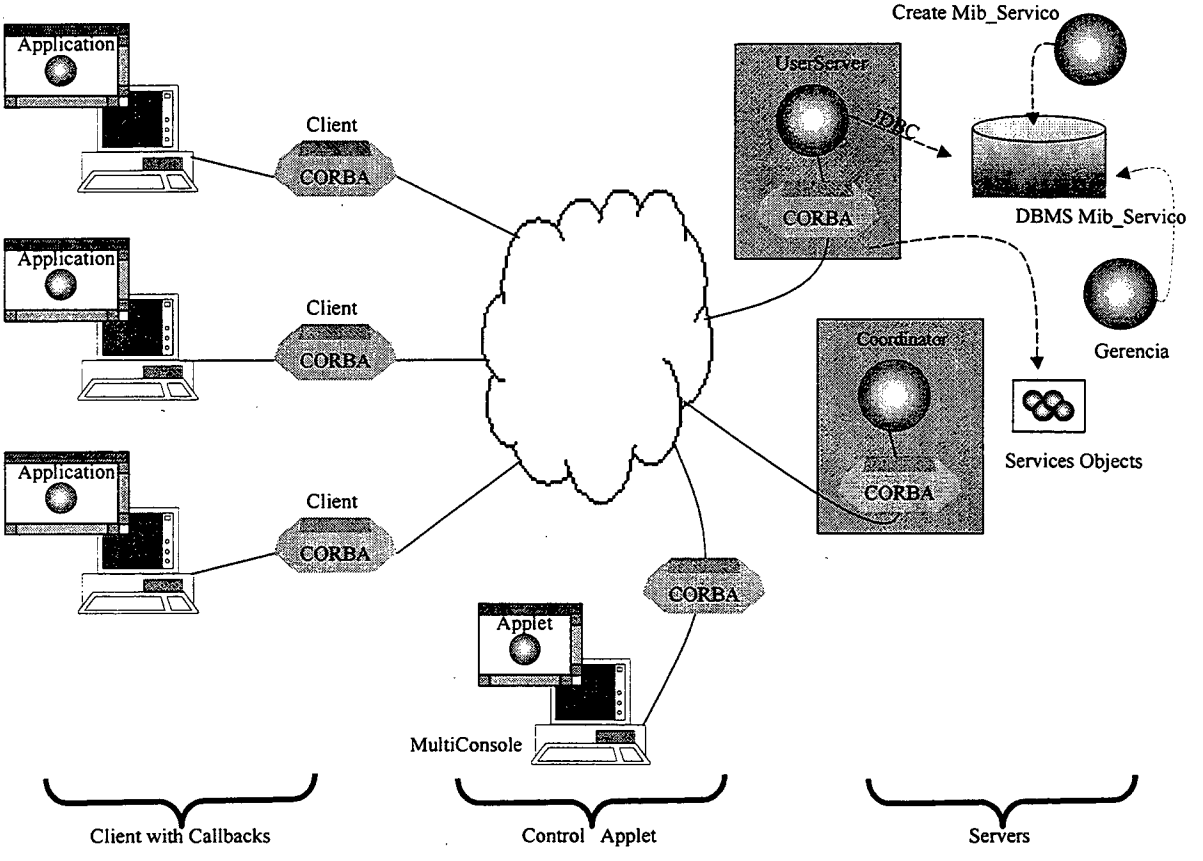
7. The test environment(o ambiente de teste): Nosso *benchmark* deve especificar o *hardware* do sistema e as configurações de *softwares*. A configuração de *hardware* deve incluir a velocidade do processador, o montante de memória, o

tamanho do *buffer pool,* e a velocidade dos discos. Devemos também especificar o ambiente de comunicação entre o cliente e o servidor.

8. ***Full disclosure*** (declaração ampla): Devemos fornecer informações suficientes para permitir a um auditor, replicar os resultados dos nossos testes.

Para alcançar estes objetivos de gerência, utilizei como base a figura 56 e estruturei o trabalho da seguinte forma:

- ✓ Descrição da IDL – para declaração formal do modelo de gerência;
- ✓ Criação de uma base de dados gerenciável, com base nos atributos definidos no modelo de gerência;
- ✓ Criação de um ambiente computacional compatível com as necessidades do modelo de gerência;
- ✓ Criação de páginas em HTML, para escolha e solicitação do serviço a ser executado;
- ✓ Implementação dos objetos (Cliente e Servidor) .




```
module service
{
 exception BankException
  { string reason;
  };
interface User
 {
 string        receiveservice(in string usr, in string prog);
 string        Consultausr(in string usr, in string pwd);
 void          AddCentralComut();
 string        consContChamada (in string usr, in string prog);
 void          decCentralComut();
 void          addServico (in string usr, in string prog);
 void          UpServico (in string usr, in string prog);
};
// Distribuidor de objetos (serviços solicitados)
 interface Dispenser
 {
   User       reserveUserObject() raises (BankException);
   void       releaseUserObject(in User userObject)
                   raises (BankException);
 };
};
```

```
        // Coordinator.idl
        module Client
        { interface ClientControl
          { // start and stop operations for controlling
client counting
            boolean start();
            string  stop();
          };
        };
        module MultiCoordinator
        { interface Coordinator
          { // object for registering and controlling client
counting
            boolean register(in string clientNumber,
                        in          Client::ClientControl
clientObjRef);
            boolean start();
            string  stop();
          };
        };
```

Figura 56. Cenário da aplicação de Gerenciamento de Serviços com sua IDL.

Os atributos dos objetos na MIB, foram implementados em um Sistema Gerenciador de Banco de Dados Relacional, o qual dispõe suas informações para qualquer aplicação que necessite dessas. Este SGDB, deve ter todas as características de um banco de dados de "primeira linha" (ORACLE, SYBASE, SQL-SERVER,DB2,...), e que suporte aplicações Cliente/Servidor. Como nosso exemplo é meramente acadêmico usamos o *MS-ACCESS-97 da Microsoft,* por ser um banco de dados que atende as necessidades deste trabalho, porém não suporta grandes capacidades de informações. O modelo desta MIB está descrito na Figura 57.

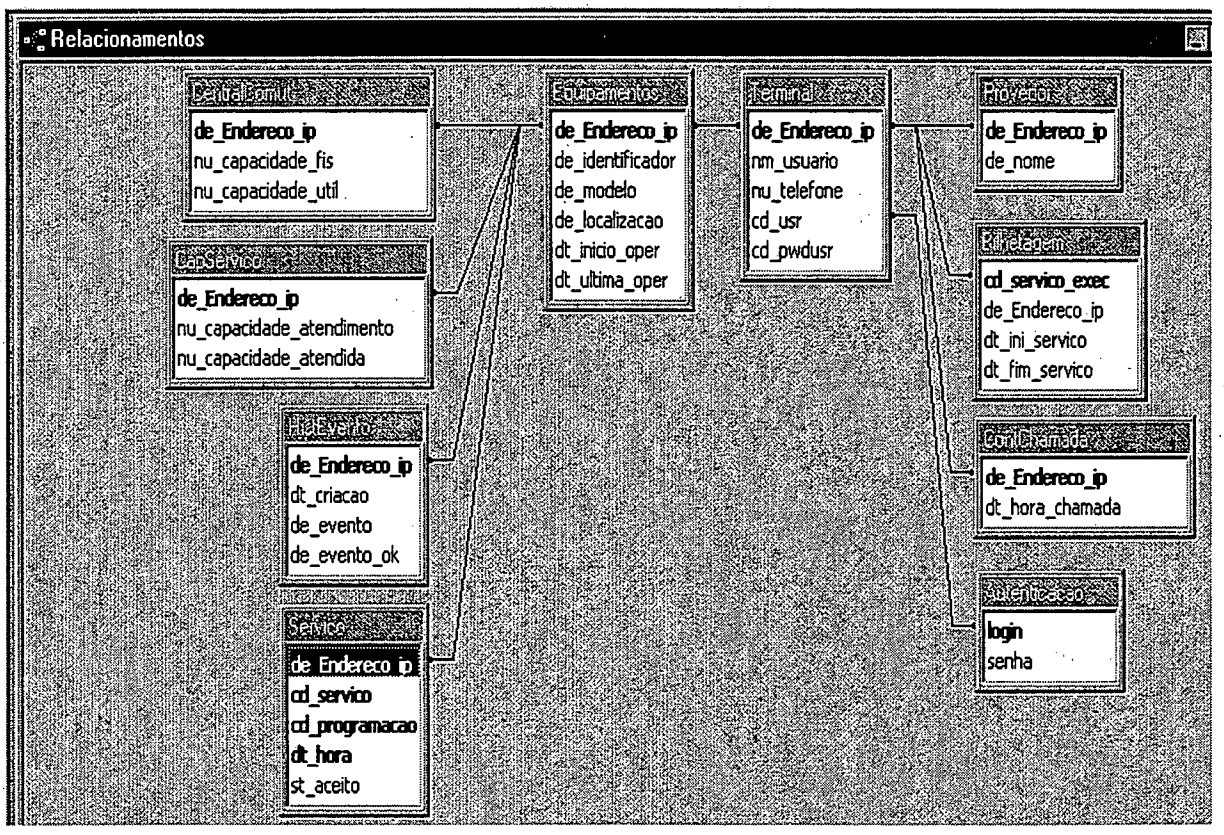


Figura 57. Base de Informações de Serviços – MIB-Serviço

Com a utilização da Internet para o gerenciamento, temos que dar a devida atenção a questão de segurança, com a utilização de canais seguros como HTTPS e SSL para a autenticação, bem como na manutenção das informações, pois nem todos os *Browsers* estão aptos com o protocolo IIOP, que viabiliza IOR através do *Gatekeeper* (Figura 58), o qual funcionará como um *firewall*.

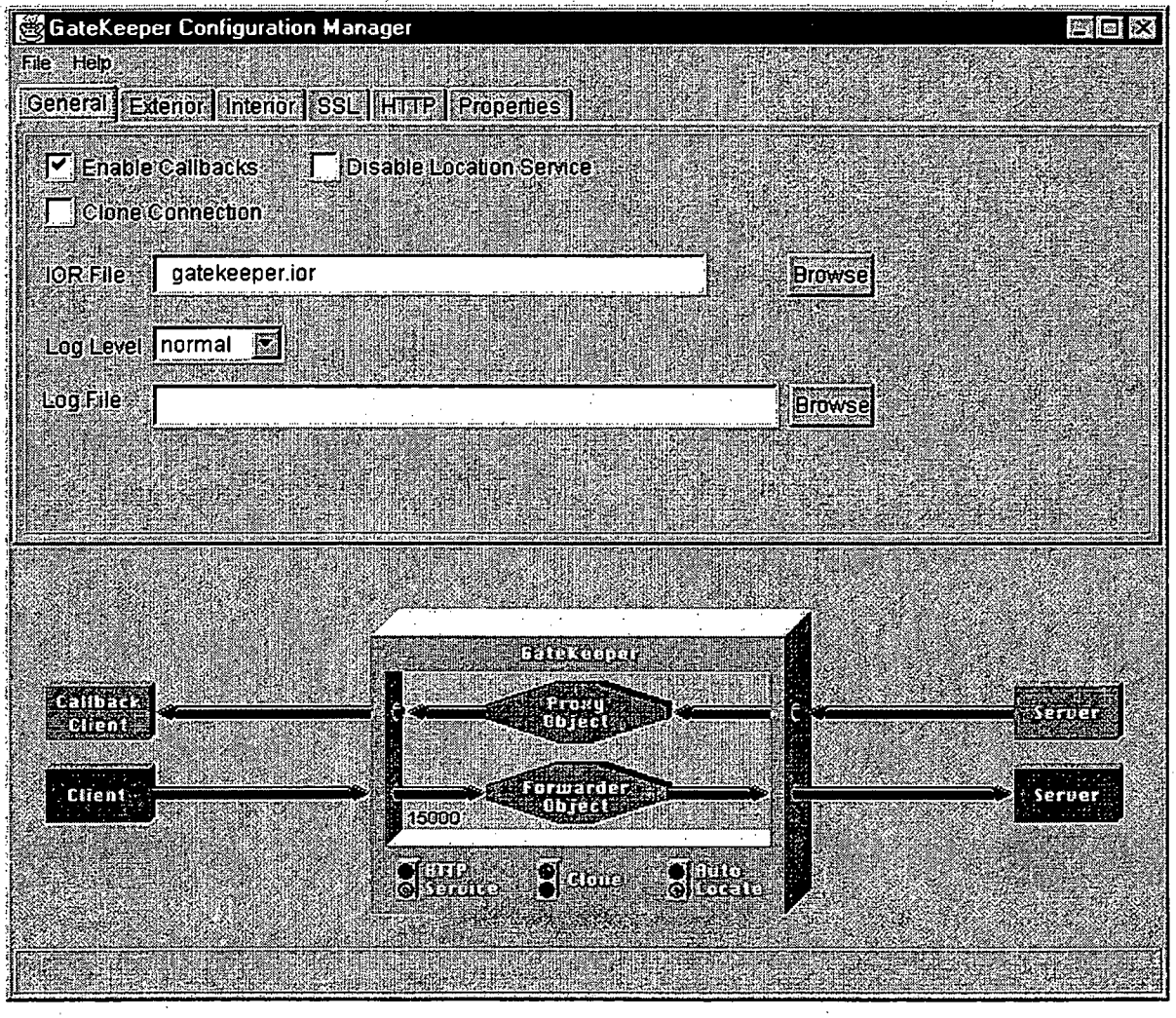


Figura 58. Ferramenta GateKeeper, controle de segurança.

Na Figura 59, o acesso às bases de dados remotas são feitas através de aplicações CORBA, dando mais segurança às transações remotas, com o auxílio de seus serviços e protocolo (CORBA*services* e CORBA IIOP). Isto causa um *overhead* menor, embora perca *performance*, pois, se utilizássemos para implementação um acesso direto às bases de dados com JDBC, seria bem mais rápido o acesso aos dados, porém, teríamos que implementar vários controles na aplicação-cliente e no ambiente Cliente/Servidor, como por exemplo: *firewall, HTTP-S, SSL* e outros. Isto não é uma boa solução para sistemas distribuídos na WEB. Por este motivo, foi criado um servidor WEB, com o Windows-NT *Server*, utilizando a ferramenta *Internet Information Server* que possibilita a criação de um *Site*, permitindo acesso as página HTML, *applets* e outros componentes da aplicação proposta, a partir do servidor WEB.

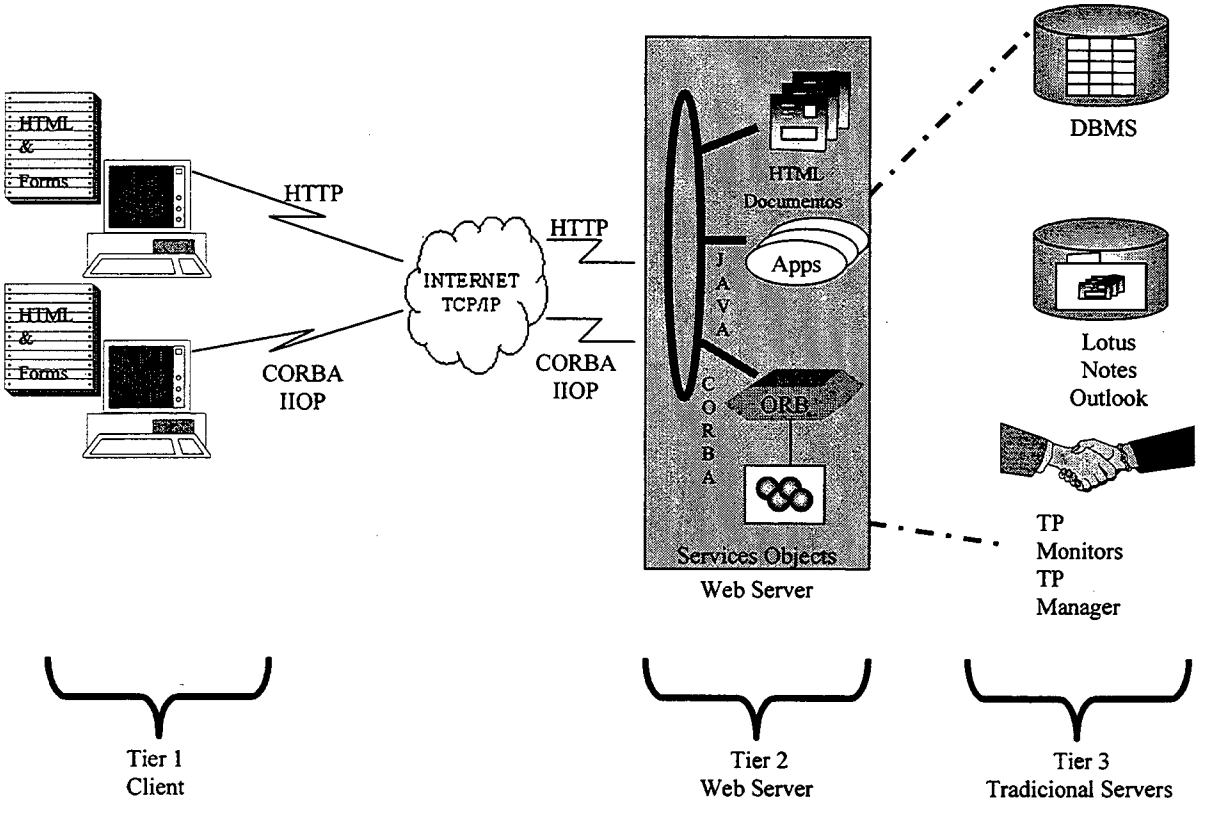


Figura 59. Acesso aos serviços sobre a WEB.

O coordenador de objetos (**CoordinatorImpl**) será um meta-objeto que contém informações sobre todos os outros objetos, o qual trata qualquer número de serviços, de qualquer tipo, em *threads* independentes, e com igual prioridade. Este meta-objeto também é conhecido como Reflexão Computacional, que tem como função básica explicitar como o objeto reage diante de uma mensagem, possibilitando a intervenção no estado de sua computação. Em nossa implementação usamos a linguagem Java para implementar o coordenador com *threads* e o *Callback* (figura 60), o tornando Cliente e/ou Servidor, dependendo do estado da computação.

O Coordenador (figura 60) permitirá capturar o estado dos serviços ativando e desativando-os, através de comandos externos ao mesmo.

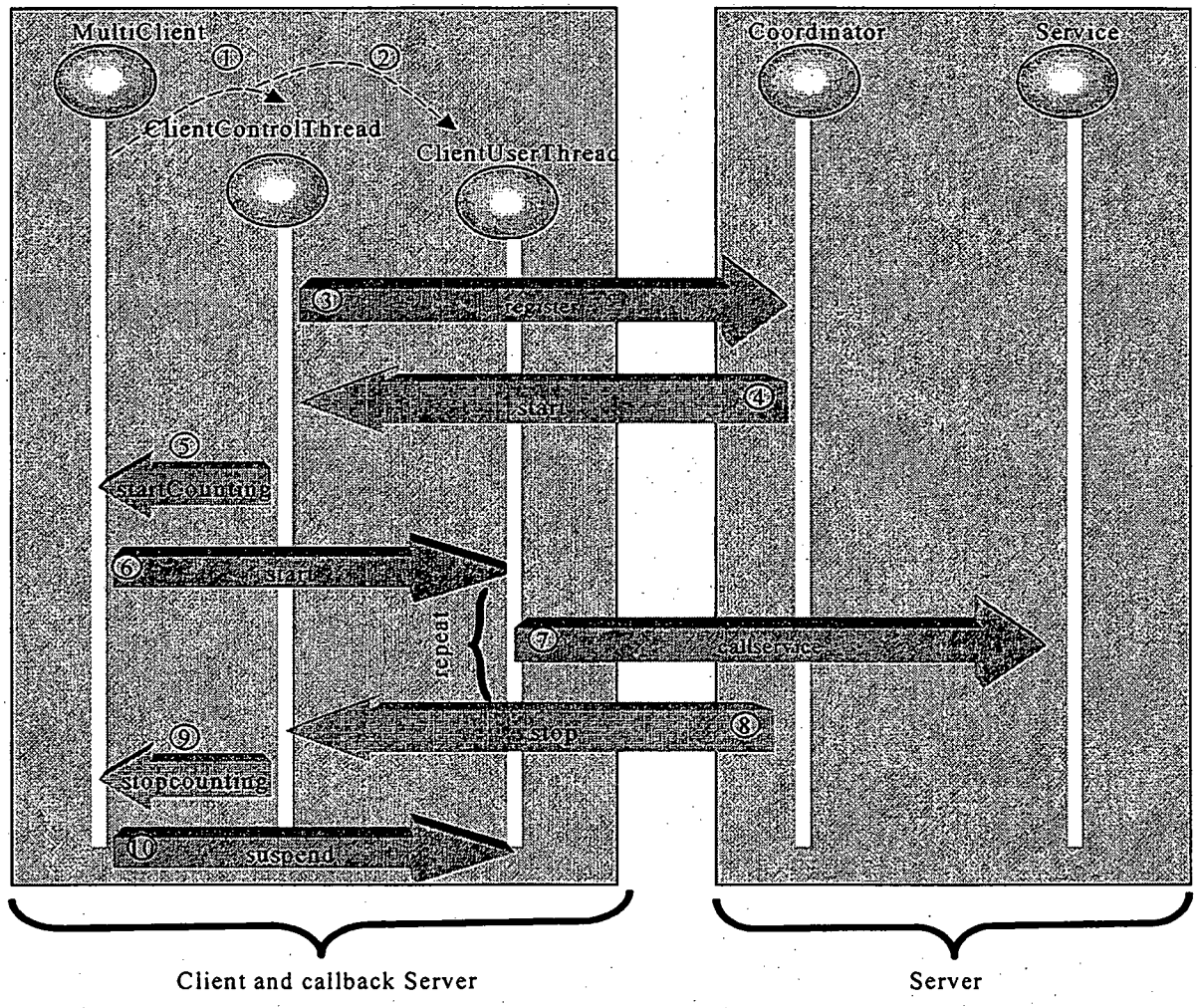


Figura 60. Cenário de execução interna do objeto cliente com Callback.

Acompanhamento passo-a-passo a figura 60.:

1. Com a aplicação *MultiClient* (vários Client), são criados dois objetos, conforme mostrado na figura 60, o primeiro deles é o que será utilizado para realização do *callback* e é chamado de *ClientControlThread*, Uma *thread,* é executada com alta prioridade sobre as outras o *ClientControlImpl* é a sua implementação e controle;
2. A segundo objeto criado é o *ClientUserThread*, objeto que é executado com uma *thread,* para qualquer novo serviço solicitado por um cliente;
3. O *ClientUserThread* chama o método *register*, implementado sobre o objeto *Coordinator* e passa para ele, a referência do objeto(serviço) solicitado pelo cliente o qual possibilita o controle de execução do *callback*;
4. O objeto *Coordinator* inicializa todos os objetos (clientes) que foram referenciados, através do método *start* implementado no objeto *ClientControlThread*, por meio de *callback;*
5. O *ClientControlThread* faz uma chamada ao método *StartingCounting* sobre o objeto *MultiClient;*
6. O Comando *thread* principal inicializa o *ClientUserThread*, através de seu método *start;*
7. *ClientUserThread* após ser ativado, fica a executar o serviço (*Call*Service) em um *loop* contínuo, sobre o servidor , até que esse serviço seja completado;
8. A parada ou suspensão do objeto *ClientUserThread* ocorre pelo

Implementação em Linguagem Java dos objetos da figura 60.

```
class CoordinatorImpl CoordinatorImplBase
    implements MultiCoordinator.Coordinator
....
    extends MultiCoordinator._
public boolean register(String clientNumber,
                Client.ClientControl clientObjRef)
            throws org.omg.CORBA.SystemException
 {
  try
  { System.out.println("Client " + clientNumber + "
Registering");
   thisClient[Integer.parseInt(clientNumber)]          =
clientObjRef;
   System.out.println("Client " + clientNumber + "
Registered");
   return true;
  } catch(Exception e)
  { System.out.println("Exception" + e);
   return false;
  }
```

```
import org.omg.CORBA.*;
public class MultiClient extends Applet
{
....
clientNumber = IP;
 controlThread = new ClientControlThread(this);
 controlThread.start();

controlThread.setPriority(Thread.NORM_PRIORITY + 1);
  userThread = new ClientUserThread(this);
  userThread.start();
....
}
```

```
// ClientControlImpl.java

class ClientControlImpl extends Client._ClientControlImplBase
implements Client.ClientControl
{ private MultiClient thisClient;

 ClientControlImpl(Object client)
 {
  super();
  thisClient = (MultiClient)client;
  System.out.println("Client Control Object Created as client :
"
          + thisClient.clientNumber);
 }

 public          boolean          start()          throws
org.omg.CORBA.SystemException
 { return thisClient.startCounting();
 }

 public          String          stop()          throws
org.omg.CORBA.SystemException
 { return thisClient.stopCounting();}
```

Tanto os objetos-clientes, quanto os objetos-servidores foram programados utilizando-se a linguagem de programação Java.

As seguintes classes foram implementadas para o Cliente: **MultiClient, ClientControlImpl, ClientUserThread** e **ClientControlThread.** E as seguintes classes constituem o Servidor: **CoordinatorServer, UserServer DispenserImpl, UserStatus, UserImpl** e **ManagerMib**. No caso do **ManagerMib,** que é a classe que gerencia os acessos a base de dados, essa classe está junto ao servidor, porque, qualquer negociação com a MIB (*Manager Information Base*) será feita através do CORBA no servidor, não sendo diretamente, por questão de segurança de acesso aos dados.

A seguir são descritas cada uma das classes utilizadas pela aplicação cliente:

- ✓ **MultiClient** - É uma aplicação *multithread* que permiti ao cliente receber *callback* (chamada de um servidor para um cliente), para cada *thread* específica. Uma *thread* chama o método *callservice* remotamente no servidor **UserServer,** enquanto uma outra aguarda o **CoordinatorServer,** avisando quando deve parar seu processamento, através do método remoto *receiveservice*; desta forma nós criamos um objeto híbrido, o qual numa determinada hora é cliente e em outra é um servidor.
- ✓ Neste modelo, vários usuários podem solicitar serviços simultaneamente. A aplicação **MultiClient** tem uma interface GUI (*Graphic User Interface*) ver figura 61, que permite ao usuário, informar alguns argumentos, para serem passados como parâmetro para o método do servidor, através de um CORBA ORB.

- ✓ Os procedimentos utilizados foram: uma Interface GUI; o ORB; o objeto servidor **Dispenser** (o qual possui uma referência); foram criados e inicializados dois objetos: **ClientControlThread** e **ClientUserThread.** Esta aplicação ainda possui quatro métodos: *action, startCounting, stopCounting e hardleEvent.* O *frame* chama o método *action* quando o usuário "clica" sobre o botão *Send* ou *Cancel*. Quando isso acontece, o método chama as funções auxiliares *startCounting e stopCounting.* Com o método

startCounting inicializa-se um contador; calcula-se o tempo inicial o qual serve para o *Benchmark* da operação de gerência e obteve-se a referência do método remoto *reserveUserObject* que foi definido no objeto **Dispenser;** o método *stopCounting* suspende a execução do objeto remoto, isto ocorre pela chamada ao método remoto *releaseUserObject* sobre o objeto **Dispenser**; o método remoto *releaseUserObject*, também pode ser suspenso pelo atendimento do serviço solicitado pelo cliente, com a chamada ao mesmo método. O método *stopCounting*, também calcula o tempo final de execução do serviço, mostrando dados estatísticos ao gerente. A figura 61, mostra a interface gráfica para um cliente utilizar o serviço despertador.

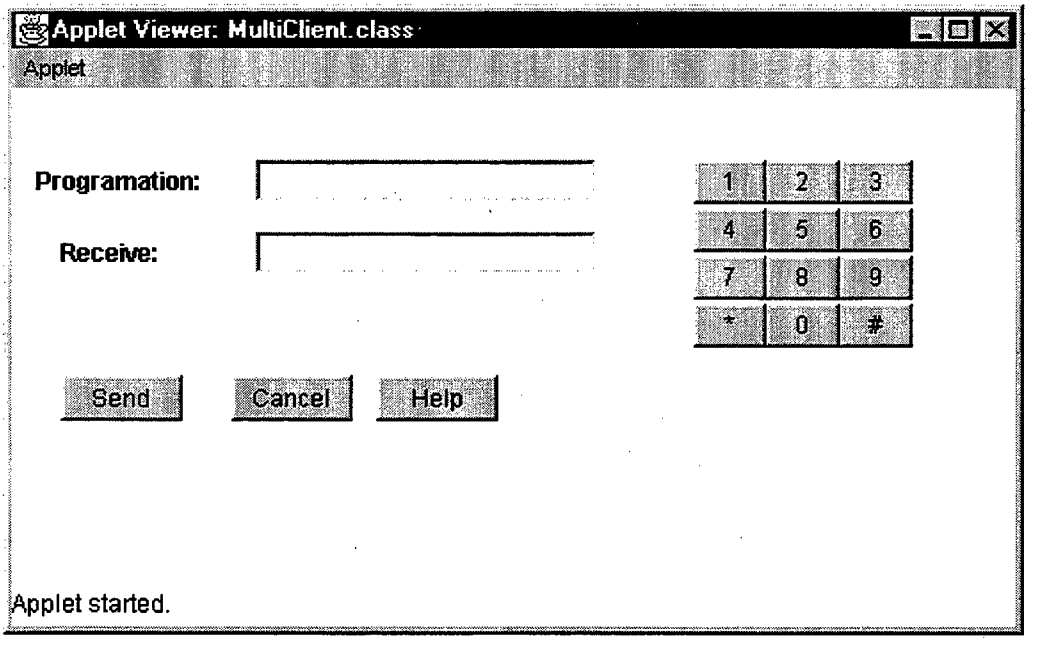


Figura 61. Interface Cliente para utilizar o Serviço Despertador

- ✓ **ClientControlThread** - Nesta aplicação é implementada a interface *callback* definida no CORBA IDL. Ela age exatamente como qualquer aplicação-servidora. Exceto que, ela faz parte da aplicação-cliente. Ver figura 60.

  - ✓ **ClientControlImpl** - Nesta aplicação está implementado o método *callservice,* que simplesmente faz a chamada ao método *startCounting* visto anteriormente; também temos o método *receiveservice* que faz a chamada ao método

stopCounting. Isto permite ao **CoordinatorServer** controlar remotamente os clientes.

- ✓ **ClientUserThread -** É uma extensão da classe *thread* da linguagem Java. Nesta aplicação foi implementado o método *run*, que é chamado quando é dado um *start ou resume* sobre uma *thread.* O método *run* implementa um *loop* continuo que chama o método *callservice*, e em seguida o método *receiveservice,* e a partir deste instante fica controlando o retorno do serviço solicitado pelo cliente; o retorno é controlado por comando de *thread*: *suspend e resume.* Os métodos *callservice e receiveservice* são implementados no objeto **ManagerMib.**

As classes servidoras são:

- ✓ **Coordinator -** Que é constituído por duas classes: **CoordinatorServer** e **CoordinatorImpl.**
- ✓ No método principal do **CoordinatorServer,** foram implementados os seguintes procedimentos: Inicialização do ORB; Inicialização do BOA; criado um novo objeto chamado **CoordinatorImpl;** exportação ao ORB do novo objeto criado que fica aguardando por requisições.
- ✓ No método **CoordinatorImpl,** foi implementado a interface **Coordinator**, que coordenará cada novo cliente do sistema. Ver a especificação IDL desta interface no quadro abaixo:

```
module MultiCoordinator
{ interface Coordinator
  { // registra os objetos e controla as chamadas de solicitação
de serviço
    boolean register(in string clientNumber,
                in Client::ClientControl clientObjRef);
    boolean start();
    string  stop();
  };
```

- ✓ No objeto-cliente foi usado o *register* para enviar um identificador de usuário e a referência do objeto, a que o identificador pertence para ser ativado o *callback* sobre o mesmo quando necessário. O **CoordinatorImpl** armazena esta informação em um *array* , para fazer a referência ao *callback*, isto é, um *array* de referência ao objeto **ClientControl**. Uma das aplicações de gerência, a **MultiConsole**, usa estes métodos para inicializar e parar a execução do cliente remoto e obter algumas estatísticas.

A outra classe servidora é a **User,** que foi implementada no módulo **service** conforme é mostrado na figura 56.

A aplicação-cliente obtém um objeto **User** pela chamada ao método *reserveUserObject* que está descrito no objeto **Dispenser**. Neste caso, o cliente poderá chamar os métodos do objeto **User**, quantas vezes for necessário. Esta implementação serve para medida do *benchmark* para a gerência.

- ✓ **Dispenser** é um *broker* do servidor de objetos, ele pre-inicializa e gerencia um conjunto de servidores de objetos, no qual atende as requisições que chegam. Cada um desses objetos tem sua própria *thread* e sua própria conexão com a base de dados através do JDBC.

- ✓ **UserServer** - No método principal desta classe foi implementado o seguinte: Inicialização do ORB; obtenção da referência para o BOA; criação de um novo objeto chamado **DispenserImpl** e é passado o número de servidores de objetos que estão sendo atendido, isto é, quantidade de chamadas que a central de comutação suporta (este dado é obtido na tabela **CentralComut** da base de dados **Mib_Servico)**; é chamado *obj_is_ready* para registrar o novo objeto **UserImpl** criado através deste ORB.

- ✓ Objeto **DispenserImpl** criado, executa dois métodos: o *reserveUserObject,* que é usado para encontrar um objeto **User** disponível, ou seja uma vaga na tabela **CentralComut** e retornar a referência deste à aplicação coordenadora. Esta então, marca o objeto como ocupado; o método *releaseUserObject* reverte, o que o método *reserveUserObject* executou.

- ✓ **UserImpl** - Nesta classe foi implementada a interface **User** definida na especificação IDL figura 56. A classe **Java *constructor*** cria um novo objeto chamado de **ManagerMib,** e conecta-se a ele. Nesta classe estão implementados os métodos:
  - ✓ *receiveservice* - Neste método está a lógica da aplicação referente ao serviço solicitado pelo cliente que faz o controle da hora exata que o despertador executará a chamada ao cliente através de um *callback;*
  - ✓ *Consultausr* - Este método contém a lógica da aplicação referente à verificação e autenticação do cliente;
  - ✓ *AddCentralComut* - Este método implementa a lógica da aplicação referente ao controle de adicionar um serviço na tabela referente ao controle da Central de Comutação;
  - ✓ *consContChamada* - Neste método foi implementada a lógica da aplicação referente à verificação do provedor de endereço dos equipamentos;
  - ✓ *decCentralComut* - Neste método está a lógica da aplicação referente ao atendimento de um serviço na tabela referente ao controle da Central de Comutação;
  - ✓ *addServico* - Neste método está implementada a lógica da aplicação referente a adição e controle de um serviço;
- ✓ *UpServico* - Este método implementa a lógica da aplicação referente a atualização de serviços executados, para futuras contabilizações.

- ✓ **UserStatus** - Esta classe implementa uma estrutura com dois campos: *reference* e o *inUse*; o primeiro, armazena a referência ao objeto-servidor; o segundo, contém uma situação que identifica os objetos(em execução, ou inativos). Esse método é utilizado para medidas de *benchmark* e para controles do gerente.

- ✓ **DispenserImpl** - Nesta classe é mantido um vetor referente aos campos da estrutura **UserStatus**, com base nesse vetor, controla os servidores de objetos. Esses servidores de objetos mantém conexões própria com a base de dados.

- ✓ **ManagerMib** - Esta classe implementa vários métodos de acesso e manutenção da base de dados **Mib_servico**, através de conexão feita por JDBC.

Além dessas aplicações acima citadas, ainda temos outras: a **CreateMib_Servico , Gerencia** e a **Multiconsole**.

- ✓ A **CreateMib_Servico** - É uma aplicação que roda *stand-alone* no servidor e que tem como funções principais: criar; manter e remover tabelas da base de dados **Mib_Servico**.

- ✓ A **Gerencia e o MultiConsole** - São aplicações de Gerência de Serviços, são executada de qualquer ponto da WEB, permitem ao gerente analisar e manter o bom funcionamento do servidor e dos serviços solicitados pelos clientes.

O cenário das classes acima descrita pode ser melhor compreendido observando a figura 62.

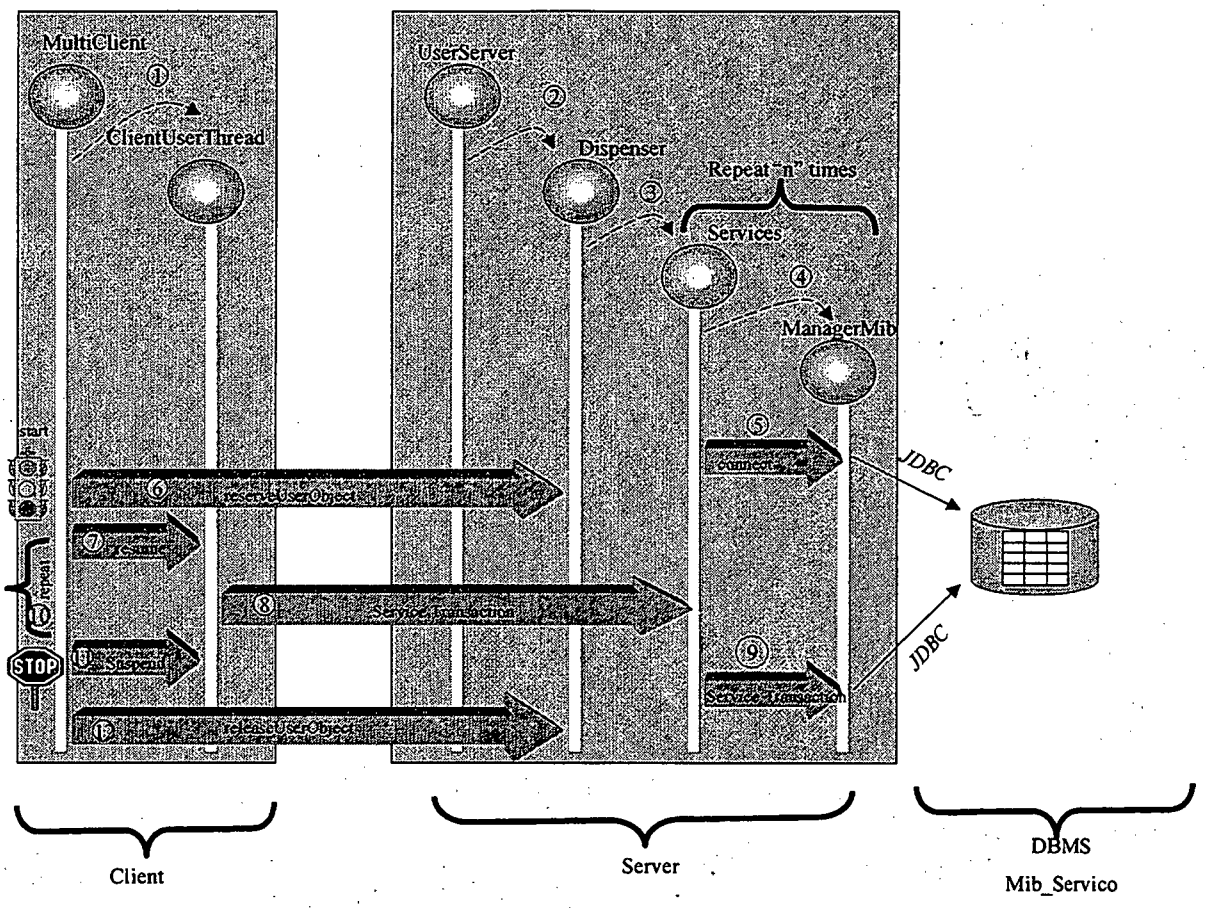


Figura 62. Cenário de execução interna do serviço.

Acompanhamento passo-a-passo a figura 62.:

1. Com a aplicação *MultiClient* (vários Client), é criada uma nova *thread* para cada novo serviço, os quais são chamados de *ClientUserThread*, e é executada como *thread;*
2. Do lado servidor é executado um objeto chamado *UserServer* , que cria um outro objeto chamado *Dispenser*, esse objeto acessa a tabela *CentralComut* na base de dados, Mib-Servico e obtém o dado referente a quantidade máxima de thread, ou seja serviços, que poderá processar, observe na figura 56 a descrição em IDL da Interface *Dispenser;*
3. A implementação do *Dispenser*, é chamada de *DispenserImpl*, o qual cria um conjunto antecipado de usuários, que chamei de *UserImpl (Services),* e esse armazena uma referência, para cada serviço solicitado no *array* de um outro objeto que é chamado de *UserStatus*. O *DispenserImpl* também invoca o *obj_is_ready* , que notifica ao *Basic Object Adpter* (BOA), que o objeto está pronto, podendo ser ativado, e registrado. Com isso, cada nova solicitação de serviço é criada no objeto *UserImpl*, ocorrendo esta criação através do ORB;
4. Para cada *UserImpl* criado, também é criado um canal de comunicação com o objeto *ManagerMib*, via JDBC;
5. O *UserImpl (Service)*, faz a chamada através de um *connect* para o objeto *ManagerMib*, em seguida efetiva a transação via a linguagem SQL (*Standard Query Language*). Isto é feito desta forma, pois a conexão a base de dados remota é lenta, por este motivo quando o objeto servidor de serviço é ligado, ele já deixa preparado várias *threads* com suas conexões a base de dados, para que cada serviço solicitado pelo cliente via WEB, não fique a espera de seu serviço por longo tempo;
6. No momento em que um serviço é solicitado por um cliente, o sistema, chama um método *reserveUserObject* do objeto *Dispenser,* para obter um dos objetos criados antecipadamente;
7. No mesmo instante que foi solicitado um serviço é ativado o *ClientUserThread* que chama o método *resume*, o qual ativa a thread, O método *resume, start e suspend* são métodos herdados da classe *Thread*;
8. O objeto *ClientUserThread*, após ser ativado, fica a executar o serviço (*Service*

Implementação em Linguagem Java dos objetos da figura 62.

```
import java.awt.*;
import java.applet.*;
import java.net.*;
import java.util.*;
import org.omg.CORBA.*;
public class MultiClient extends Applet
.....
.....
// Initialize the ORB.
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
    // Bind to the Dispenser Object, reserve User object
    myDispenser   = service.DispenserHelper.bind(orb,"My Dispenser");
    myUser = myDispenser.reserveUserObject();
.....
.....
 clientNumber = IP;
 controlThread = new ClientControlThread(this);
 controlThread.start();
 controlThread.setPriority(Thread.NORM_PRIORITY + 1);
 userThread = new ClientUserThread(this);
 userThread.start();
.....
.....
 myUser = myDispenser.reserveUserObject();
.....
.....
 userThread.resume();
.....
.....
// chamada do Service Transaction
 class ClientUserThread extends Thread
{ MultiClient myClient;
  int sit = 0;
  int pare = 0;
  ClientUserThread(MultiClient client)
  { myClient = client;
  }

  public void run()
  { // run call/receive transactions
   myClient.statusField.setText("Waiting for running Service");
   while (true)
   { try
     ....
     ....
     myClient.retorno = myClient.myUser.receiveservice(myClient.usr,
                                        myClient.prog);
     myClient.statusField.setText(myClient.retorno );
.....
.....
```

```
public class DispenserImpl extends service._DispenserImplBase
{
private UserStatus[] user =
                new UserStatus[maxObjects];
public DispenserImpl(java.lang.String name, int num)
 {
  super(name);
  try
  {
   // get reference to boa, needed for obj_is_ready
   org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
   org.omg.CORBA.BOA boa = orb.BOA_init();

   // prestart n User Objects
   numObjects = num;
   for (int i=0; i < numObjects; i++)
   {
    user[i] = new UserStatus();
    user[i].ref = new UserImpl("User" + (i+1));
    boa.obj_is_ready(user[i].ref);
   }
  } catch(org.omg.CORBA.SystemException e)
  { System.err.println(e);
  }
 }
 public service.User reserveUserObject( )
                throws                          service.BankException,
org.omg.CORBA.SystemException
                        {
  for (int i=0; i < numObjects; i++)
  {
   if (!user[i].inUse)
     { user[i].inUse = true;
      System.out.println("User" + (i+1) + " reserved.");
      return user[i].ref;
     }
  }
  return null;
 }
 public void releaseUserObject(
                service.User userObject)
                throws                          service.BankException,
org.omg.CORBA.SystemException
 {
  for (int i=0; i < numObjects; i++)
  {
   if (user[i].ref == userObject)
     { user[i].inUse = false;
      System.out.println("User" + (i+1) + " released.");
      return;
     }
  }
  System.out.println("Reserved Object not found");
  return;
 }
```

```
class UserStatus
{
 UserImpl ref;
 boolean       inUse;
 UserStatus()
 { ref = null;
  inUse = false;
 }
```

```
class UserServer
{
 static public void main(String[] args)
 {
  int    numberInstances = arqs;
  // Numero de servidores, ou seja
capacidade
  // da central de comutaçao

  try
  {
   if (args.length == 1)
     numberInstances                    =
Integer.parseInt(args[0]);

   // Initialize the ORB.
   org.omg.CORBA.ORB          orb        =
org.omg.CORBA.ORB.init();

   // Initialize the BOA.
   org.omg.CORBA.BOA          boa        =
orb.BOA_init();

   // Create the BankDispenser object
   DispenserImpl dispenser =
        new              DispenserImpl("My
Dispenser", numberInstances);

   // Export dispenser object
   boa.obj_is_ready(dispenser);

   // Ready to service requests.
   boa.impl_is_ready();

  } catch(org.omg.CORBA.SystemException
e)
  { System.err.println(e);
  }
 }
}
```

```
UserImplBase
{
 ....
 ....
 public UserImpl(java.lang.String name)
 {
  super(name);
  try
  {
   myBankDB = new Mibservico.ManagerMib();
   myBankDB.connect("c:", "Mib_Servico", "", "");
   System.out.println("ManagerMib Object " + name + "
Created");
   instanceName = name;
  } catch (Exception e)
  { System.out.println("System Exception ");
  }
 }
 public UserImpl()
 {
  super();
 }

 public String receiveservice(String usr, String prog)
     throws org.omg.CORBA.SystemException
 {
  try
  {
   ....
   ....
  retornoImpl = myBankDB.receiveservice(usr,prog);
  }
 }
public String Consultausr(String usr, String pwd)
      throws org.omg.CORBA.SystemException
 {
  try
  {
    ....
    retorno = myBankDB.Consultausr(usr,pwd);
    ....
  }
 }
 ....
 ....
}
```

```
package Mibservico;
import java.lang.*;
import java.net.URL;
import java.sql.*;
import java.util.Date;
import java.awt.*;
 public class ManagerMib
 {
 ....
 ....
  public void connect(String datasource, String db,
              String id, String pw)
              throws Exception
  {
   try
   {
    // Load the jdbc-odbc bridge driver
    Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
    String url   = "jdbc:odbc:" + "Mib_Servico";
    System.out.println("Connecting to " + url);
    con = DriverManager.getConnection(url, id, pw);
   } catch(Exception e)
   { System.err.println("System Exception in connect");
    System.err.println(e);
    throw e;
   }
  }
  public void closeConnection() throws Exception
  {
   try
   {
    System.out.println("Closing connection");
    con.close();
   } catch (Exception e)
   {    System.err.println("System    Exception    in
closeConnection");
    System.err.println(e);
    throw e;
   }
  }
  public String callservice(String usr, String prog) throws
Exception
  ....
  ....
  public String receiveservice(String usr, String prog) throws
Exception
  ....
  ....

  public void addEvento(String IP, String deevento) throws
Exception
  ....
  ....
```

6. CONCLUSÕES E PERSPECTIVAS

O modelo construído neste trabalho, para **Gerenciamento de Serviços de Telecomunicações**, consiste em focalizar alguns pontos relevantes às aplicações distribuídas, em combinação com ambientes heterogêneos e algumas vezes proprietários e que com isso promovam subsídios suficientes e necessários para a avaliação supra citada.

Não foi objetivo deste trabalho simular fielmente as características, o comportamento e a gerência dos equipamentos reais do ambiente de telecomunicações que prestem serviços aos usuários e tampouco prover tais serviços. Como a proposta da aplicação é apenas uma simulação simples e abstraída sem compromisso com o mundo real, e também objetivando uma clareza acadêmica, a aplicação constitui-se de dois subsistemas: um gerenciado, o qual simula o serviço citado; e outro gerenciador, o qual exerce algumas atividades de gerência sobre o primeiro.

Como considerado no escopo deste trabalho, a aplicação-gerente procurou tanger os aspectos clássicos da distribuição, que possibilitaram o estudo e a experiência com plataformas de distribuição, no caso o CORBA. Na comunicação entre aplicação gerente e a aplicação gerenciada foram usados recursos do ambiente Java. Tal experiência permitiu a verificação das possibilidades, das facilidades e das dificuldades da interação entre CORBA e o Java. Para se alcançar as metas foi simulado o serviço despertador automático e uma aplicação de gerência para este serviço. Esta aplicação de gerência exerce atividades simples de gerência sobre a simulação do serviço de despertador, utilizando características reflexivas. Ao se separar os aspectos de gerência, da própria construção das aplicações de serviços, usou-se a técnica de **Reflexão Computacional,** além de oferecer condições para o uso e o teste das principais características da arquitetura CORBA. Esta arquitetura usa o

mecanismo do ORB, para prover o meio de transporte das informações. Desta forma conseguimos unir as vantagens oferecidas por este novo paradigma aos benefícios dos serviços definidos como objetos CORBA. Isto possibilita que outros objetos CORBA, localizem um ao outro, independente de localização e da linguagem de programação usada, além de usufruir da interoperabilidade entre meios heterogêneos.

O uso do Visigenic 3.x, ofereceu a facilidade do mecanismo de *callback* e a linguagem de programação Java, a disponibilidade de várias estruturas prontas e que atendiam necessidades específicas como as classes *vector, threads* e o método *constructor*.

Os serviços de telecomunicações não se constituem em um paradigma, relacionando-se apenas às aplicações na Internet, como uma nova proposta de oferecê-los aos clientes.

Em sistemas distribuídos de modo geral, a possibilidade de transferir a computação para outro ponto da rede pode propiciar o desenvolvimento de soluções mais eficiente para certos problemas como por exemplo, gerência de rede, gerência de sistemas, automação de *WorkFlow,* gerência de documentação eletrônica, *Benchmark* de aplicações, gerência de Banco de Dados e outros.

O atual enfoque deste trabalho, buscou satisfazer as diferentes tecnologias existentes (analógica e digital) e torná-las interoperavéis, fugindo do escopo onde se buscava definir padrões, o que causou o aparecimento de tecnologias proprietárias no setor de telecomunicações, bem como em outros setores que utilizam a tecnologia de rede como ferramenta imprescindível ao desenvolvimento de aplicações distribuídas.

Ao elevarmos o nível de abstração, a proposta deixou em aberto o meio de se implementar os serviços, permitindo assim o uso de qualquer mecanismo independente de CORBA-Java, o que na prática poderá não ser eficiente. A interoperabilidade entre sistemas na WEB, já está comprometida pela dependência da linguagem de programação.

A aplicação de *Benchmark*, usada neste modelo de Gerência de Serviços de Telecomunicações, poderá ser utilizada em qualquer sistema que se preocupe com gerência de recursos computacionais. Da maneira que esta aplicação foi feita, o responsável pela gerência, é capaz de avaliar: se a configuração do ambiente computacional está em conformidade com o seu planejamento ou se precisa ser alterada; se o custo/performace de todo o sistema de gerenciamento de serviço está adequado aos gastos previstos.

Esta aplicação de *Benchmark* foi desenvolvida para WEB, podendo ser ativada a qualquer momento pelo gerente, e de qualquer lugar. Esta aplicação foi escrita em linguagem Java, e com referências a objetos, controlados pelo controlador de objetos da aplicação, permitindo com isso, as avaliações mencionadas acima.

Os acessos à Base de Dados Remotas, poderiam ser mais rápidos, caso fizéssemos, os acessos diretamente ao SGBD via JDBC, porém, por questão de segurança, os acessos foram feitos através do ORB a um servidor remoto, que têm implementado todos os acessos à Base de Dados do sistema de gerenciamento de serviços. Outra vantagem, além da segurança, é a reutilização e a padronização dos códigos de acesso à Base de Dados, tornando as manutenções futuras, facilmente realizadas, bem como, novas implementações de aplicações de um modo geral.

A utilização de *object reference*, foi um poderoso recurso oferecido pelo CORBA, na negociação de serviços distribuídos, bem como o *callback* que é muito eficiente para o controle de clientes em aplicações Cliente/Servidor.

Esse trabalho detectou, que aplicações na WEB, que manipulam hora, devem ter uma atenção especial quanto a sincronização de relógio, tanto por parte do Cliente, quanto por parte do Servidor. Além desta sincronização, temos que dar a devida atenção a horários de verão regionais, pois, prejudicam sensivelmente estas aplicações.

Outro aspecto relevante neste trabalho é o fato de que a maioria das aplicações WEB baseadas em dados, sofrem com a falta de velocidade ou largura de banda dos meios de comunicação. Como a natureza da WEB é *stateless,* ou seja, sem estado e também, sem conexão, cada vez que um Cliente requisita dados ao Servidor, esta requisição precisa conter todas as informações de estado que são necessárias para satisfazer a solicitação. Portanto, o Servidor, tipicamente, executa uma nova consulta à base de dados, ao invés de reutilizar os conjuntos de resultados previamente estabelecidos. O Servidor gera também um novo documento HTML a cada vez, contendo os dados selecionados e pré-formatados para exibição pelo *browser* WEB. Com o ambiente CORBA, melhorou-se o desempenho e o tempo de resposta ao Cliente, e ao mesmo tempo, reduzindo o número de requisições que o Cliente faz ao Servidor e a quantidade de trabalho que o Servidor precisa executar. Para se conseguir este melhor desempenho, documentos HTML comuns ou com *applets,* descrevem como os dados devem ser exibidos, porém os dados em si são fornecidos ao Cliente separadamente, através do *stub* CORBA da aplicação. Essa divisão de responsabilidade permite que o Cliente visualize, pagine, busque e ordene os dados, localmente, atualizando a tela conforme necessário sem fazer novas conexões ao Sevidor.

Concluindo, podemos afirmar que este modelo de gerência é flexível à medida que pode ser implementado por um outro ORB, o qual possua mapeamento IDL-Java. Esta exigência não chega a ser comprometedora, visto que, entre as linguagens mais adequadas para a implementação de aplicações na WEB, a linguagem Java é a mais eficiente.

PERSPECTIVAS

Com a realização deste trabalho, novas perspectivas poderão ser desenvolvidas para dar segmento a este:

⇒ Avaliação do desempenho do paradigma da Reflexão Computacional sobre a arquitetura CORBA;

⇒ implementação da proposta em domínios diferentes ao de telecomunicações, utilizando as técnicas apresentadas;

⇒ comparação da proposta apresentada neste trabalho, com a utilização de outras arquiteturas distribuídas, como por exemplo: OSF/DCE, Java/RMI e DCOM.

⇒ desenvolvimento de aplicações de gerenciamento de redes de computadores ou de redes de telecomunicações, utilizando CORBA e Reflexão Computacional;

⇒ desenvolvimento de aplicações voltadas a força-tarefa, manutenção em equipamentos ligados a serviços, planilhas estatísticas para tomada de decisão sobre ampliação e melhoria de serviços oferecidos aos clientes.

7. REFERÊNCIAS BIBLIOGRÁFICAS

[ANC95] Ancona, M.; Doderao, G.; Gianuzzi, V.

Reflective architeture for reusable faut-tolerant software

Latin American Conference in Informatics, 21., 1995, Canela, RS, BR.

Anais - SBC-1995.

[BAR96] Barros, Claudius D´Artagnan C.

Excelência em Serviços, Questão de Sobrevivência no Mercado

QualityMark Editora Ltda. - 1996.

[BOO91] Booch, Grandy

Object Oriented Design With Applications

The Benjamin/Cummings Publishing Company, Inc. - 1991.

[BOO92a] Booch, Grandy

The Booch Method : Notation Part I

Computer Language, September - 1992.

[BOO92b] Booch, Grandy

The Booch Method : Notation Part II

Computer Language, September - 1992.

[BOO94] Booch, Grandy

Object-Oriented Analysis and Design

The Benjamin/Cummings Publishing Company, Inc. - 1994, pag 589.

[BUS96] Buschmann, F.

System of Patterns : pattern-oriented software architecture.

John Wiley & Sons, England, 1996.

[BRI93] BRISA

Gerenciamento de Redes Uma Abordagem de Sistemas Abertos

Makrons Books do Brasil Editora Ltda - 1993.

[COA92] Coad, Peter & Yourdon, Edward

Análise Baseada em Objetos

Editora Campus/Yourdon Press - Rio de Janeiro - 1992.

[CUR97] D.Curtis

Java, RMI and CORBA

OMG White paper - 1997

[DIL93] Dilley, J.

Object-Oriented Distributed Computing With C++ and OSF DCE

OSF DCE SIG, Request for Comments (RFC) 49.0, outubro 1993.

[FER89] Ferber J.

Computational Reflection in Class Based Object-Oriented Language.

SIGPLAN Notices New York, v.24, n.10, pag.317-326, october 1989.

[GOS95] Gosling, James & McGilton, Henry

The Java Language Environment - A White Paper

Sun Microsystems Computer Company, outubro de 1995.

[GRA89] Graube, N.

Metaclass compatibility. SIGPLAN Notices New York, v.24, n.10, pag.305-315, october 1989.

[GRI97] Grimes, Richard

Professional DCOM Programming

Wrox Press Ldt, 1997

[HER94] Herbert A .

Distributed Object,

ANSA, tech report APM.1009.01, maio 1994.

[JUN96] Junior, José Helvécio Teixeira & Sauvé, Jácques Phillipe & Moura, José Antão Beltrão

Do Mainframe para a Computação Distribuída

Livraria e Editora Infobook S.A. - 1996.

[KHO86] Khoshafian, S. & Copeland, G.

Object Identify

SIGPLAN Notices, Vol. 21 (11) - 1986.

[KIC96] Kiczales, G.

Beyond the Black Box: Open Implementation. IEEE software, v.13, n.1, pag 8-11, jan. 1996.

[KON96] Kong Qinzheng & Graham Chen

Integrating CORBA and TMN Environment

CiTR Pty Ltd. Queensland, Austrália 1996.

[KON97] Kong Qinzheng & Graham Chen

Integrated TMN service provisioning and management environment

CiTR Pty Ltd. Queensland, Austrália 1997.

[KRA96] Kraig Brockschmidt

What OLE is Really About

OLE Team, Microsoft Corpoartion, July 1996.

[LIS97] Lisboa Black, Maria L.

Reflexão Computacional no Modelo de Objetos

Texto apresentado na Universidade Federal do Rio Grande do Sul

CPGCC, UFRGS, 1997. Tese de Doutorado

[LUN97] Lung C.,Lau

Tecnologia de Desenvolvimento de Aplicações Distribuídas em Ambiente Heterogêneo

LCMI/DAS/CTC/UFSC, agosto de 1997.

[MAE87] Maes P.

Concepts and experiments in computational reflection. SIGPLAN Notices, New York, v.22, n.12, p. 147-169. Trabalho apresentado no OOPSLA, 1987, Orlando, Flórida.

[MAE87] Maes P.

Issues in computationl reflection. Meta-Level Architecture and Reflection. Amsterdan:Elsevier Science, 1988.

[MAR93B] Roger Martin.

Changing the Mind of The Corporation

Havard Buniness Review. 71(6):81-96, novembro-dezembro 1993.

[MAR94] James Martin

Princípios de Análise e Projetos Baseados em Objetos

Editora Campus - RJ - 1994.

[MON94] Montenegro, Fernando & Pacheco, Roberto

Orientação a Objeto em C++

Editora Ciência Moderna Ltda - Rio de Janeiro - 1994.

[M3200] Recommendation M.3200 - **Maintenance Telecommunication Management Network, TMN Management Services: Overview, ITU-T, 1992.**

[OSK94] Oskiewicz, E. & Edwarrs J.

A Model for Interface Groups

ANSA Project, APM 1002.01, maio 1994.

[OMG 91] Object Management Group

The Common Object Request Broker : Architecture and Specification

Revision 1.1, OMG Document , dezembro 1991.

[OMG 94] Object Management Group

IDL C++ Language Mapping Specification

OMG Document , setembro 1994.

[OMG 96] Object Management Group

The Common Object Request Broker 2.0/IIOP specification

Revision 2.0, OMG Document , agosto 1996.

[OMG 97] Object Management Group

Java Language Mapping RFP

OMG Document , março 1997.

[ORL96] R.Orlafi, D. Harkey, J. Edwards

The Essencial Distributed Objects - Survival Guide

John Wiley & Sons, Inc, 1996

[ORL97] R.Orlafi, D. Harkey

Client/Server Programming with JAVA and CORBA

John Wiley & Sons, Inc, 1997

[PEN96] Manoel Camillo Penna, Andrey Patiticci

Relatório Técnico à respeito de Sistema de Gerenciamento Distribuído, com exemplo sobre despertador automático.

[POM95] S. Pompilho

Análise Essencial

Livraria e Editora Infobook S.A . - RJ - 1995.

[ROS92] Rosenberry W., Kenney D., Fisher G.,

Understanding DCE

O'Reilly Associates Inc., 1992.

[STE94] Steel, L.

Beyond Objects. EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, 8, 1994, Bologna, Italy.

[SUN 96] Sun Microsytem,

Java Remote Method Invocation Specification

Microsystem Computer Corporation, Revision 0.9, Draft may 1996.

[TAN94] Tanenbaum, Andrew S.

Redes de Computadores

Editora Campus - 1994.

[TAN95] Tanenbaum, Andrew S.

Sistemas Operacionais Modernos

Editora Prentince Hall do Brasil LTDA. - 1995.

[UMA97] Amjad Umar

Object-Oriented Client/Server Internet Environments

Prentice Hall PTR, Inc. 1997.

[VOG97] Andreas Vogel & Keith Duddy

Java Programming with CORBA

John Wiley & Sons, Inc. - 1997.

[WIN90] Winblad, Ann L.; Edward, Samuel D. & King, David R.

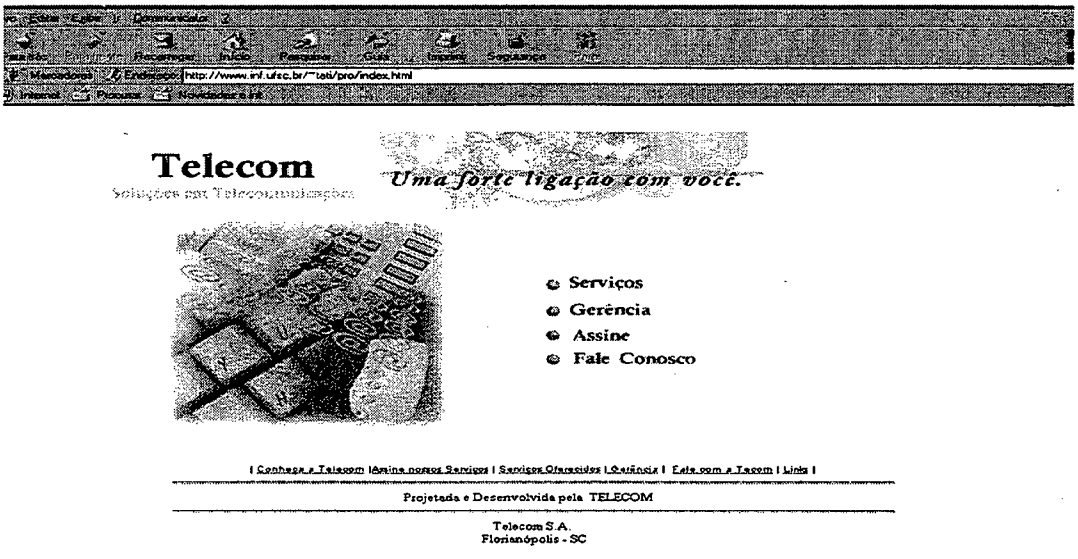
Object-Oriented Software

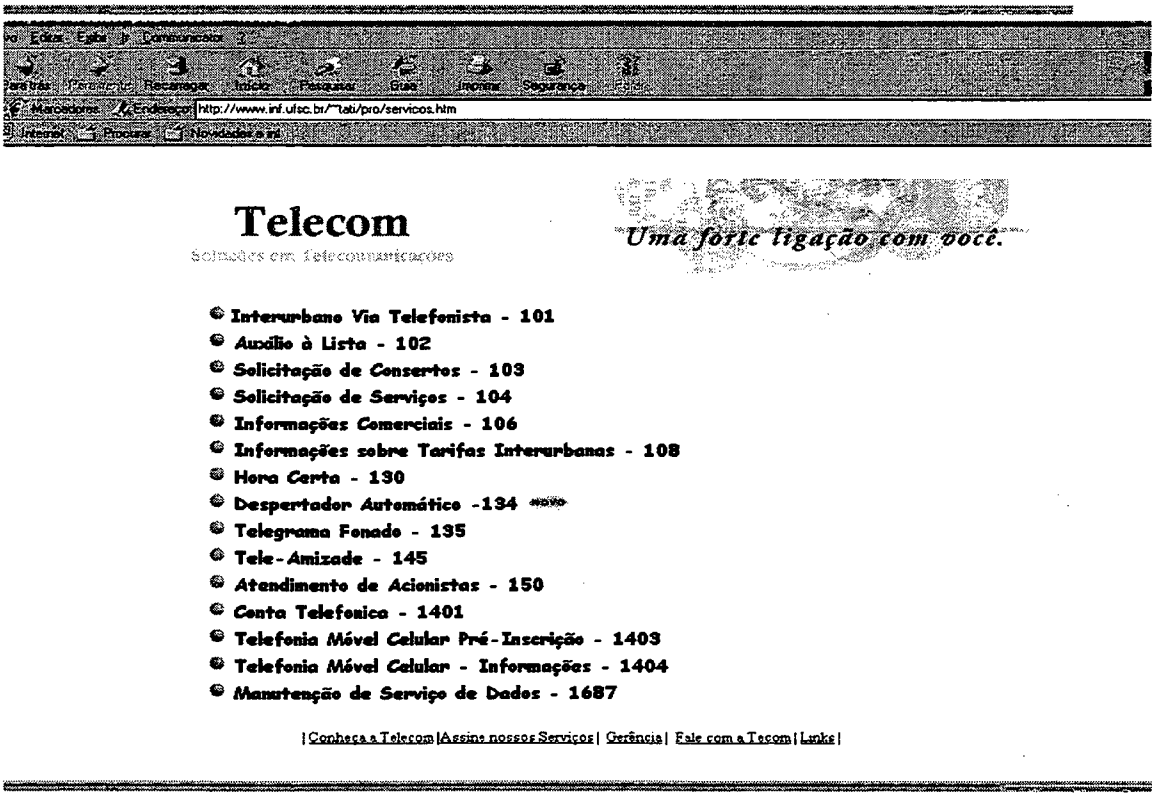
Addison-Wesley Pub. Comp. Inc. - 1990.

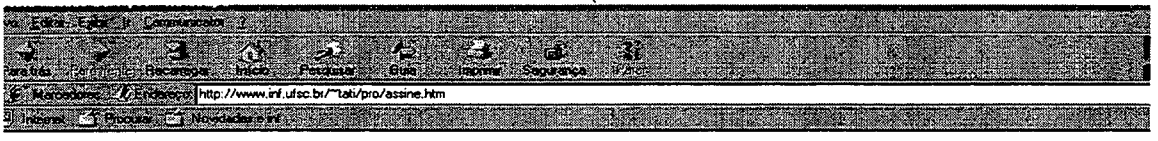
[X700] Recommendation X.700 - **Data Communication Networks, ITU-T, 1992.**

ANEXOS

Interfaces de Acesso ao Sistema Proposto







Telecom

Soluções em Telecomunicações

Uma forte ligação com você.

Para a sua segurança, apenas a solicitação da assinatura do serviço é feita através desta página. Você receberá em sua próxima fatura mensal seu user Name e senha para o acesso os serviços da Telecom. Quando você for usar o serviço pela primeira vez, modifique imediatamente a sua senha.

Dados do Assinante

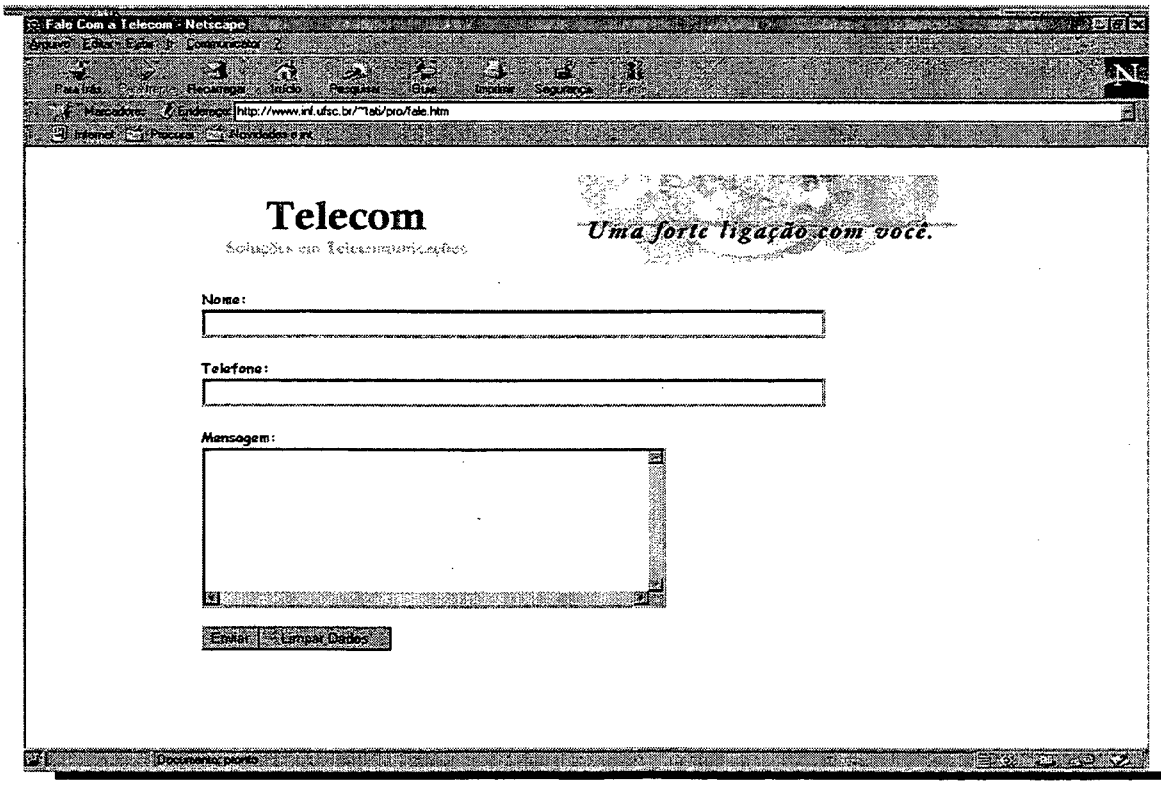
Nome:

CPF:

Telefone:

Enviar | Limpar Dados

| Conheça a Telecom | Gerência | Serviços Oferecidos | Fale com a Tecom | Links |



Telecom

Soluções em Telecomunicações

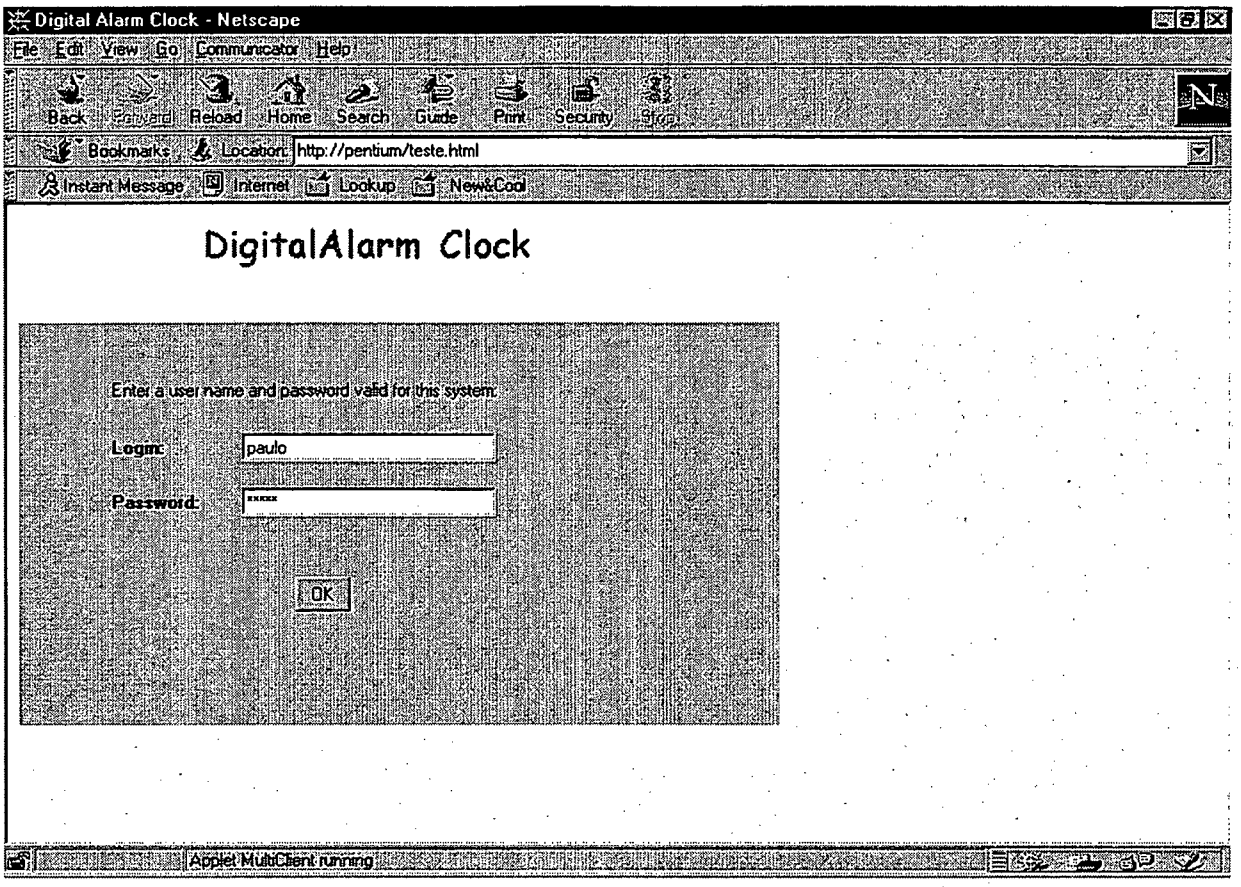
Uma forte ligação com você.

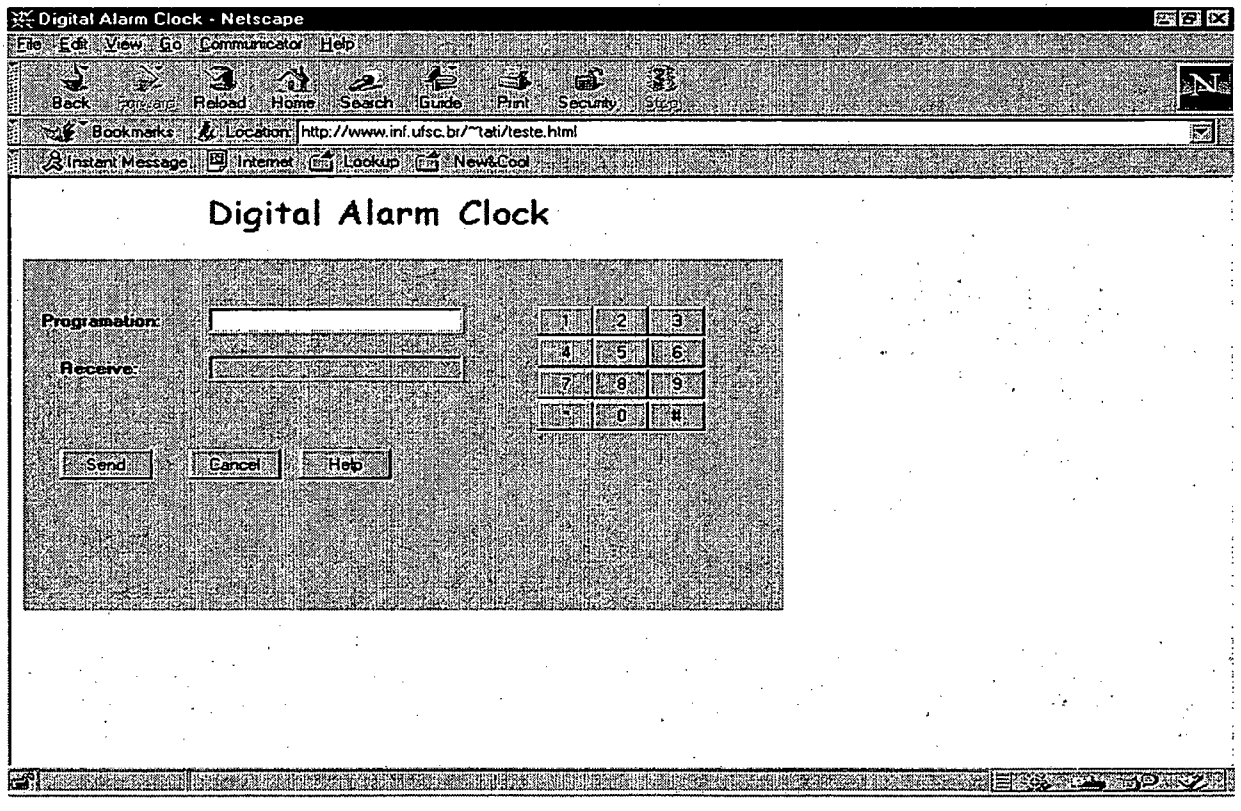
Nome:

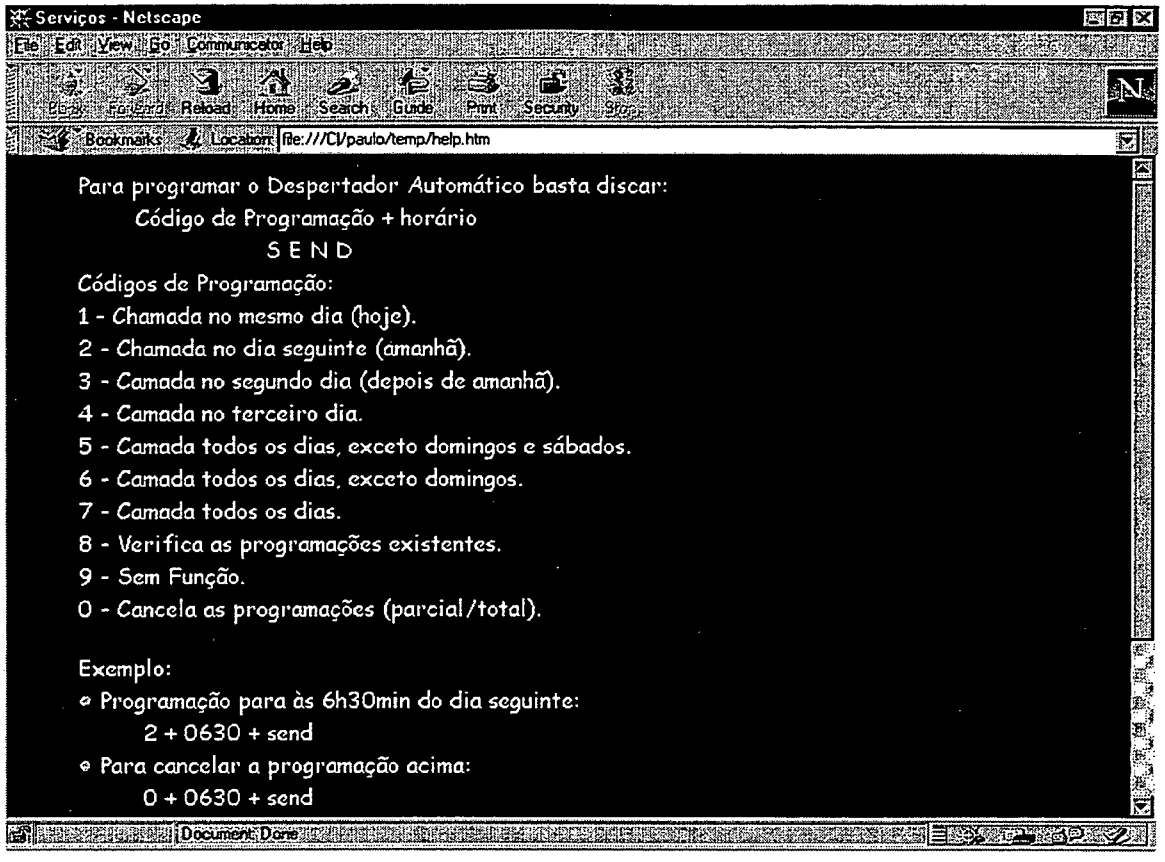
Telefone:

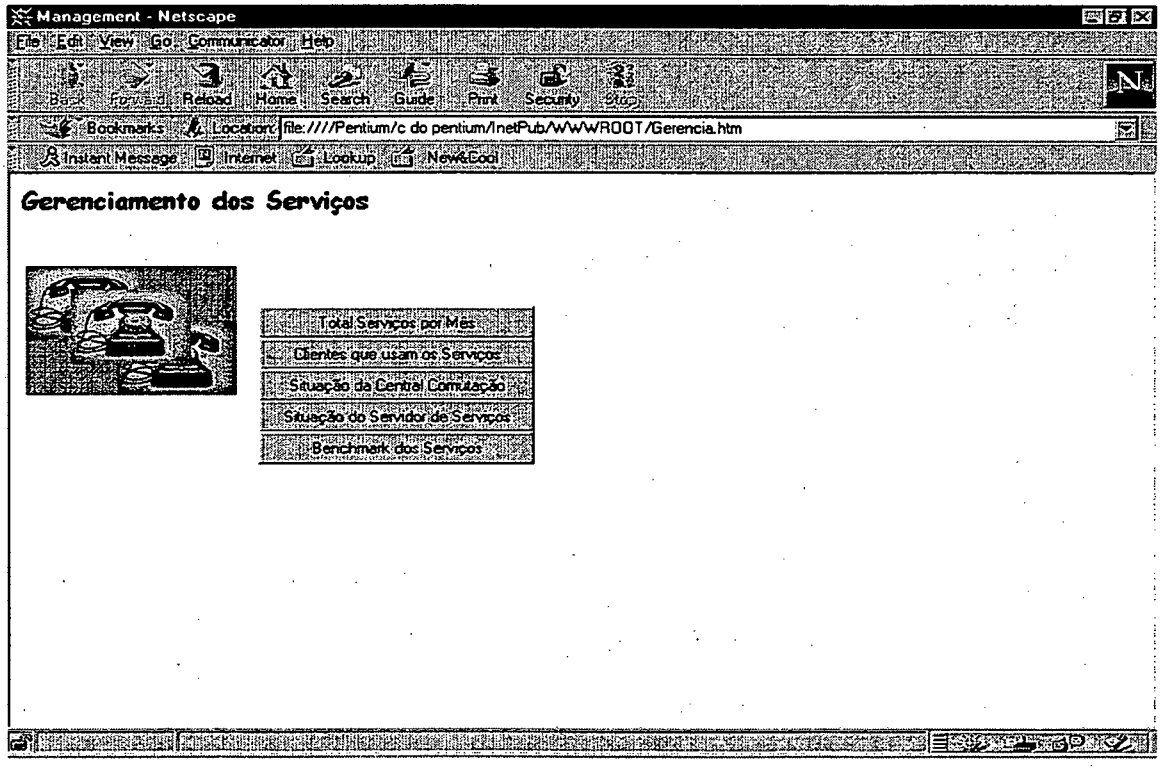
Mensagem:

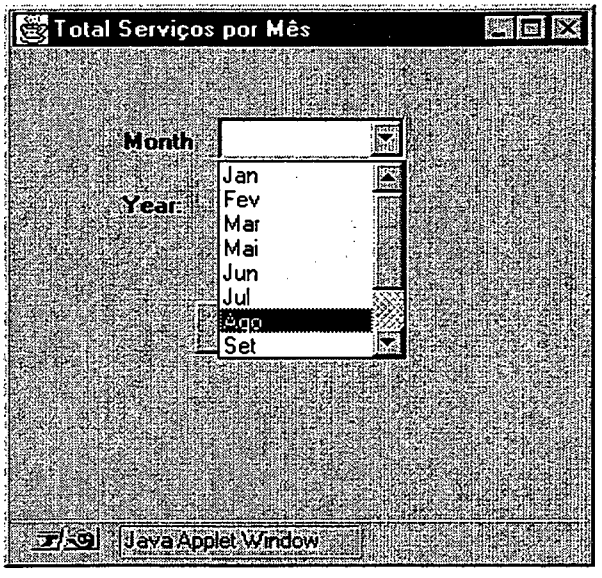
Enviar | Limpar Dados

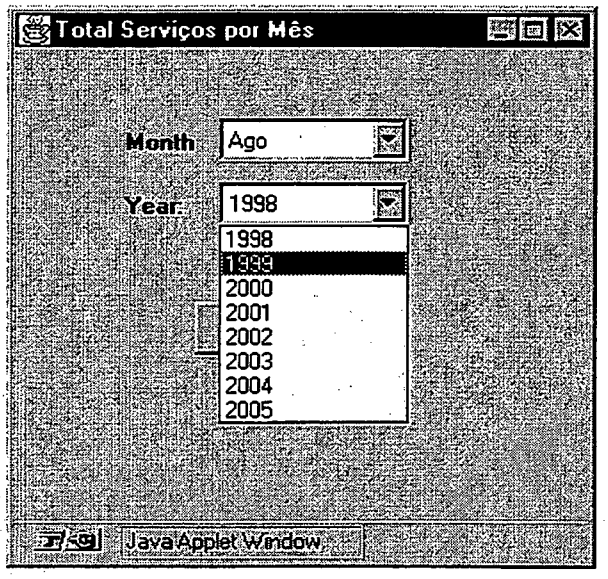

Digital Alarm Clock - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://pentium/teste.html
Instant Message Internet Lookup New&Cool
DigitalAlarm Clock
Enter a user name and password valid for this system:
Login: paulo
Password: *****
OK
Applet MultiClient running

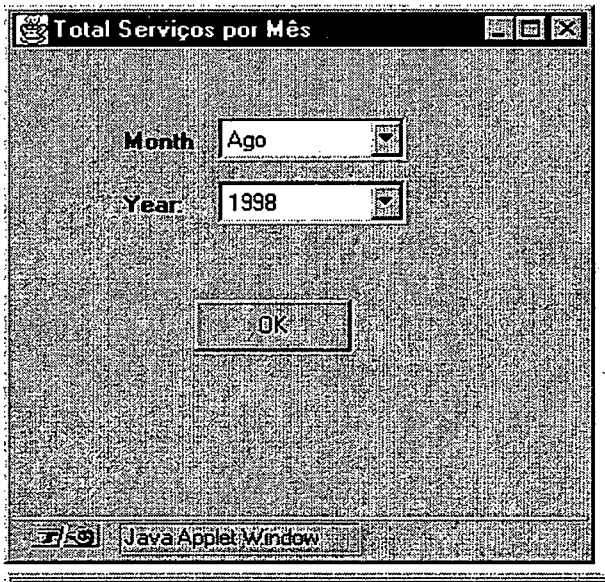

Digital Alarm Clock - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://www.inf.ufsc.br/~tati/teste.html
Instant Message Internet Lookup New&Cool
Digital Alarm Clock
Programation:
Receive:
1 2 3
4 5 6
7 8 9
* 0 #
Send
Cancel
Help

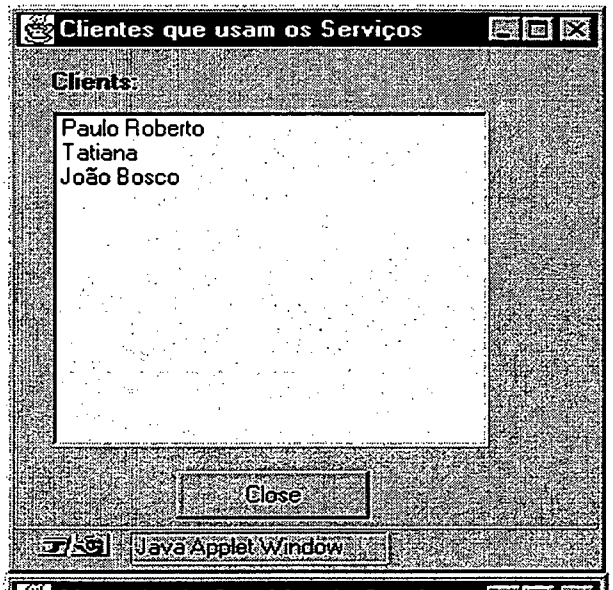

Serviços - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///C|/paulo/temp/help.htm
Para programar o Despertador Automático basta discar:
Código de Programação + horário
S E N D
Códigos de Programação:
1 - Chamada no mesmo dia (hoje).
2 - Chamada no dia seguinte (amanhã).
3 - Camada no segundo dia (depois de amanhã).
4 - Camada no terceiro dia.
5 - Camada todos os dias, exceto domingos e sábados.
6 - Camada todos os dias, exceto domingos.
7 - Camada todos os dias.
8 - Verifica as programações existentes.
9 - Sem Função.
0 - Cancela as programações (parcial/total).
Exemplo:
Programação para às 6h30min do dia seguinte:
2 + 0630 + send
Para cancelar a programação acima:
0 + 0630 + send
Document: Done


Management - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:////Pentium/c do pentium/InetPub/WWWROOT/Gerencia.htm
Instant Message Internet Lookup New&Cool
Gerenciamento dos Serviços
Total Serviços por Mes
Clientes que usam os Serviços
Situação da Central Comutação
Situação do Servidor de Serviços
Benchmark dos Serviços


Total Serviços por Mês
Month
Year:
Jan
Fev
Mar
Mai
Jun
Jul
Ago
Set
Java Applet Window

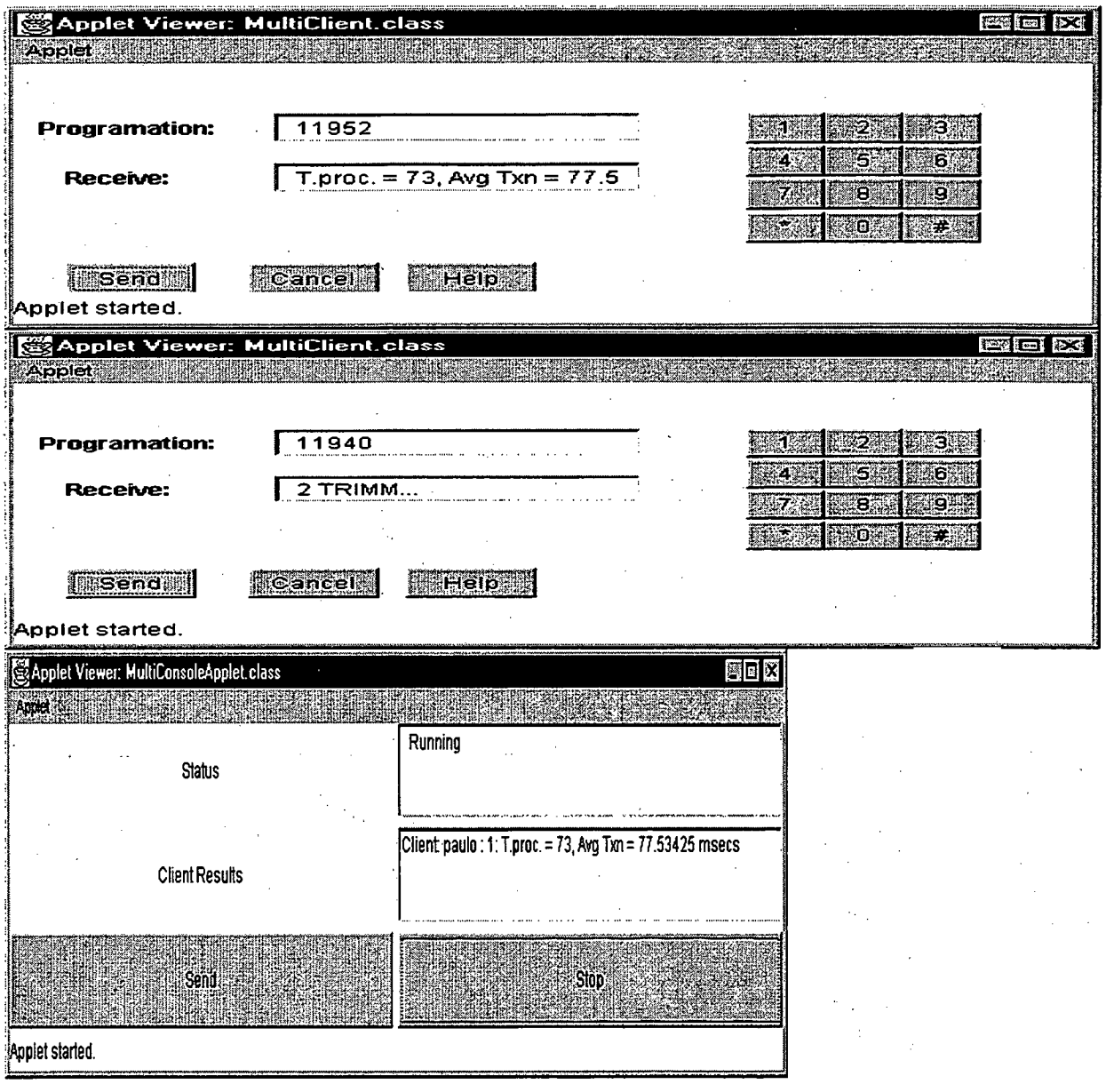

Total Serviços por Mês
Month
Ago
Year:
1998
1998
1999
2000
2001
2002
2003
2004
2005
Java Applet Window


Total Serviços por Mês
Month
Ago
Year:
1998
OK
Java Applet Window


Clientes que usam os Serviços
Clients:
Paulo Roberto
Tatiana
João Bosco
Close
Java Applet Window


Situação da Central de Comutação
A capacidade Utilizada da
Central de Comutação é de 3.33 %
Close
Java Applet Window


Situação do Servidor de Serviços
A capacidade Utilizada
Central de Comutação é de 3.44 %
Close
Java Applet Window


Applet Viewer: MultiClient.class
Applet
Programation:
11952
Receive:
T.proc. = 73, Avg Txn = 77.5
1 2 3
4 5 6
7 8 9
0 #
Send
Cancel
Help
Applet started.
Applet Viewer: MultiClient.class
Applet
Programation:
11940
Receive:
2 TRIMM...
1 2 3
4 5 6
7 8 9
0 #
Send
Cancel
Help
Applet started.
Applet Viewer: MultiConsoleApplet.class
Applet
Status
Running
Client Results
Client: paulo : 1: T.proc. = 73, Avg Txn = 77.53425 msecs
Send
Stop
Applet started.