LEONARDO LINCOLN BIANCHETTI

SISTEMA DISTRIBUÍDO COM O PADRÃO DE ARQUITETURA CORBA

Trabalho de conclusão de curso apresentado ao Curso de Ciência da Computação.

UNIVERSIDADE PRESIDENTE ANTÔNIO CARLOS

Orientador: Elio Lovisi Filho

BARBACENA

2003

LEONARDO LINCOLN BIANCHETTI

SISTEMA DISTRIBUÍDO COM O PADRÃO DE ARQUITETURA CORBA

Este trabalho de conclusão de curso foi julgado adequado à obtenção do grau de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação da Universidade Presidente Antônio Carlos.

Barbacena – MG, _____

_____

Prof. Elio Lovisi Filho - Orientador do Trabalho

_____

Prof. Lorena Sophia C. de Oliveira - Membro da Banca Examinadora

_____

Prof. Eduardo Macedo Bhering- Membro da Banca Examinadora

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me permitido chegar até aqui. Aos meus pais que me deram muita força e coragem nessa longa caminhada, aos meus irmãos, cunhados e sobrinhas.

RESUMO

O constante aumento na capacidade de processamento dos equipamentos atuais, conjugado ao crescimento das redes de computadores, têm contribuído cada vez mais para o desenvolvimento de aplicações distribuídas. Tais aplicações pressupõem a comunicação constante entre os nodos formadores deste sistema, tendo como uma solução bastante satisfatória a este modelo a utilização de objetos distribuídos. Baseado neste paradigma, a OMG (*Object Management Group*) estabeleceu a arquitetura CORBA (*Common Objects Request Broker Architecture*), como uma forma de especificar as interfaces e procedimentos necessários a comunicação entre esses objetos. Este trabalho tem por objetivo mostrar a aplicação e utilização do padrão CORBA associado a objetos distribuídos.

Palavras Chave: Objetos Distribuídos, CORBA, UML, Implementação, Casos de Uso.

SUMÁRIO

LISTAS

1 INTRODUÇÃO

Com avanço da tecnologia, e com os novos conceitos de programação surge uma nova arquitetura de comunicação entre cliente e servidor: o CORBA (Common Object Request Borker Arquiteture), que tem como finalidade permitir aos objetos de sistemas distribuídos comunicar-se entre si de forma transparente, independente de quais plataformas de hardware, sistemas operacionais e linguagem de programação foram desenvolvidos e se baseia no conceito de objetos distribuídos.

Existem vários tipos de arquiteturas, entre elas a DCOM, EJB e DCE que permitem utilizar-se de aplicações fundamentadas em internet, como a arquitetura fundamentada em serviços http, ou Protocolos Pessoais.

Uma delas é a arquitetura de objeto distribuído, que une orientação a objetos à tecnologia de sistemas distribuídos. Com isto, é possível a criação de aplicativos multicamadas, onde a primeira camada é a interface do aplicativo com o cliente final, sendo que a requisição criada por ele é processada pela segunda camada, que pode estar em outra máquina, que acessa as formações contidas em um banco de dados em uma terceira camada.

Hoje em dia, sem dúvida alguma, esta arquitetura é a melhor opção para desenvolver aplicações de interação de cliente/servidor. [RICCIONI, 2000]

O objetivo deste projeto é estudar objetos distribuídos, com a aplicação do padrão CORBA a fim de desenvolver modelos de objetos distribuídos e analisar as características do padrão. Para tanto, deve-se identificar suas possíveis aplicações e modelar um sistema, utilizando este padrão dentro de um ambiente de programação.

Neste trabalho será desenvolvido um aplicativo de um sistema distribuído com o padrão CORBA, a fim de mostrar sua aplicabilidade e vantagens.

Esta monografia consiste dos seguintes capítulos, visando facilitar a compreensão do leitor.

Neste primeiro momento apresento os objetivos deste estudo.

No capítulo 2 discutiremos os Sistemas Distribuídos e o padrão CORBA, abrangendo uma introdução sobre este padrão, serviços e facilidades CORBA, interface de domínio, interface de aplicação, clientes, repositório de interface, stubs cliente, IDL, ORB, Skeleton, invocações, Adaptador de Objetos Básico (BOA), implementação de objetos, Repositório de implementações e interoperabilidade.

No capítulo 3, abordaremos a UML (*Linguagem Unificada de Modelagem*), apresentando uma definição dos seus principais modelos de elementos que serão utilizados no projeto. Define-se ainda visões e diagramas que serão utilizados para modelar o sistema do estudo de caso do próximo capítulo

No Capítulo 4, apresentaremos o estudo de caso, contendo a modelagem, e a implementação de uma aplicação de uma loja de eletroeletrônicos, no qual pudemos aplicar os conceitos estudados nos capítulos anteriores.

No capítulo 5, são apresentadas as conclusões obtidas a partir da realização deste trabalho.

2 OBJETOS DISTRIBUÍDOS E CORBA

2.1 OBJETOS DISTRIBUÍDOS

Na programação distribuída usando a arquitetura cliente-servidor, clientes e servidores podem ser implementados usando qualquer paradigma de programação. Unidas ao paradigma de orientação a objetos, as aplicações podem referenciar-se a objetos do sistema em qualquer lugar da rede, proporcionando assim, maior flexibilidade e reutilização de códigos, tornando mais eficiente o desenvolvimento das aplicações. Assim, é possível que um serviço específico seja executado por um método de algum objeto. No entanto, mesmo que o cliente também tenha sido desenvolvido orientação a objetos, na comunicação entre o cliente e o servidor esse paradigma deve ser esquecido, devendo ser utilizado algum protocolo pré-estabelecido de troca de mensagens para a solicitação e resposta ao serviço[RICCIONI,2000].

Um sistema de objetos distribuídos é aquele que permite a operação com **objetos remotos**. Dessa forma é possível, a partir de uma aplicação cliente orientada a objetos, obter uma referência para um objeto que oferece o serviço desejado e, através dessa referência, invocar métodos desse objeto -- mesmo que a instância desse objeto esteja em uma máquina diferente daquela do objeto cliente.

O conceito básico que suporta plataformas de objetos distribuídos é o conceito de arquiteturas de objetos. Essencialmente, uma arquitetura orientada a objetos estabelece as regras, diretrizes e convenções definindo como as aplicações podem se comunicar e inter-operar. Dessa forma, o foco da arquitetura não é em como a implementação é realizada, mas sim na infra-estrutura e na interface entre os componentes da arquitetura[RICCIONI,2000].

De modo geral, uma aplicação distribuída é composta de 3 camadas lógicas:

Camadas de Apresentação: Em geral, é centrada nas estações, principalmente em redes Cliente/Servidor.

Camadas de Aplicação: Responsável pelas regras do negócio e pelo gerenciamento do fluxo de dados. Fica localizado nos servidores de aplicação, mas em alguns aspectos são manipulados também pelo cliente e pelo servidor.

Camada de Dados: Assume o controle sobre os dados, propriamente ditos.

O desafio de implementação de uma aplicação distribuída é definir as camadas lógicas e concretizá-las em componentes físicos.Em uma outra etapa, deve-se preocupar como os componentes se comunicarão [RICCIONI,2000]

2.2 CORBA

O padrão **CORBA** (**Common Object Request Broker Architeture),** é um modelo proposto pela OMG (Grupo de Gerenciamento de Objetos), que permite aos objetos de sistemas distribuídos comunicar-se entre si de forma transparente. Estas aplicações podem estar sendo executadas em diferentes plataformas de hardware e sistemas operacionais e podem ser construídas em diferentes linguagens de programação [RICCIONI, 2000].

Utilizando o padrão CORBA é possível ter aplicações completamente distribuídas. Potencialmente, com cada parte de software sendo executado em qualquer local da rede e em qualquer plataforma, sem que o usuário perceba que isto está acontecendo, e sem que o desenvolvedor precise se preocupar em criar soluções que resolvam os problemas de interoperabilidade entre os diferentes pedaços da aplicação [GU-CORBA, 2002].

A primeira versão do CORBA, a 1.1, surgiu em 1991, momento este, em que se definiu a IDL (Interface Definition Language) e a API (Application Programming Interfaces) que é um conjunto normalizador de rotinas e chamadas de software que podem ser referenciadas por um programa aplicativo para acessar serviços essenciais de uma rede [RICCIONI, 2000].

Porém, a interoperabilidade entre os objetos desenvolvidos em linguagens de diferentes fabricantes, só veio em 1994, com a segunda versão do CORBA, a 2.0, quando implementou-se no mesmo o IIOP (Internet Inter-ORB Protocol) [GTA-UFRJ, 2000].

Entretanto, o CORBA é apenas um componente de um grande quadro chamado **Object Management Architecture (OMA),** composta de quatro elementos:

- **Núcleos CORBA e ORB (OBJECT REQUEST BROKER)** – manipulam requisições entre objetos;
- **Serviços CORBA** – definem serviços ao nível de sistema que ajudam a gerenciar e manter objetos;
- **Facilidades Comuns** – definem facilidades e interfaces a nível de aplicação
- **Objetos de Aplicação** – são os objetos propriamente ditos no nível visível de aplicação [RICCIONI, 2000]

2.2.1-SERVIÇOS E FACILIDADES CORBA

Serviços de Objetos são coleções de aplicações (interfaces e objetos) que suportam funções básicas para usar e implementar objetos. Neste estudo apenas serão descrito na integra alguns desses serviços, que serão mais interessantes no desenvolvimento da aplicação, embora existam 16 tipos [GTA-UFRJ, 2000]:

Ciclo de Vida: define serviços e convenções para as 4 operações básicas: criar, copiar, mover, apagar.

Persistência: define uma única interface para armazenamento de objetos nos servidores de armazenamento.

Nome: Sua principal característica é permitir que os clientes possam localizar o objeto pelo nome.

Eventos: Permite que componentes possam ser registrados ou não de acordo com a especificação do evento.

Controle de concorrência: Controla para que o banco de dados ou sistema de arquivos seja acessados por um cliente de cada vez, permitindo-lhe acesso exclusivo a um registro ou arquivo. Funciona como um gerenciador, que é capaz de controlar algumas vantagens para uma determinada execução em depreciação da outra.

Transação: Onde é definido a coordenação entre objetos recuperáveis. Fornece ainda a confiabilidade, a robustez, a atomicidade, consistência, isolamento e durabilidade.

Atomicidade: Assegura que uma transação tenha sido realizada. Quando um trabalho é completamente feito, é dito em acordo (commit). Quando não é feito em acordo é chamado de (rollback).

Externalização: Define um padrão para externalização e internalização de dados de um componente através do mecanismo de stream-like.

Consulta: Define operações de consulta para objetos.

Negociação: Permite que se possa localizar qualquer serviço, de qualquer provedor podendo, ainda, oferecer os seus serviços a outros objetos [RICCIONI,2000]

2.2.2- FACILIDADES COMUNS DO CORBA

Assim como os objetos de serviços, também possui coleções de serviço. Uma diferença das facilidades comuns para objetos de serviços é que as interfaces não são

consideradas fundamentais. Elas são divididas em 2 aspectos: horizontal e vertical[GTA-UFRJ, 2000].

Horizontal: Utilizada por várias aplicações independentes de sua área. As principais categorias são: interface do usuário, administração da informação, administração do sistema, administração de tarefas.

Vertical: "Está relacionada à criação e definição de interfaces IDL vinculadas aos segmentos do mercado vertical" [GTA-UFRJ,2000].

2.2.2.1- INTERFACE DE DOMÍNIO

As premissas são muito parecidas com as das facilidades comuns, mas são orientadas para o domínio de uma aplicação específica [GTA-UFRJ].

2.2.2.2- INTERFACE DE APLICAÇÃO

Interface desenvolvida para uma determinada aplicação. Pelo fato da OMG não produzir aplicações, não há uma padronização para as interfaces [GTA-UFRJ].

2.2.2.3- OBJETOS DE APLICAÇÃO

Dentro dos objetos de aplicação, temos os objetos de negócio que formam o núcleo de uma aplicação compatível em CORBA. Com objeto de negócio é possível construir uma aplicação. Outro fator importante é que os objetos de negócio manipulam processos do mundo real

Para o gerenciamento da complexidade de uma aplicação, os objetos concentram mais interfaces entre eles e as implementações [GTA-UFRJ,2000].

2.2.3 CLIENTES

"Cliente é toda entidade responsável por invocar uma determinada operação sobre uma implementação de objeto" [GTA-UFRJ]. E a própria implementação de objetos pode ser um cliente fazendo requisições a este mesmo.

O cliente conhece apenas a estrutura lógica do objeto de acordo com sua interface e experiências do comportamento do objeto através das invocações.

Como uma das características do CORBA, os detalhes utilizados para que seja feito esse acesso aos serviços de um determinado objeto é transparente e dispensável do conhecimento do cliente, precisando o mesmo apenas saber interagir com a interface do tal objeto [MONTEZ,1998]. Essa invocação feita pelo cliente, que para o mesmo aparenta ser local, poderá ser feita utilizando-se o stub, ou poderá ser feita utilizando-se um conjunto de rotinas de invocações feitas dinamicamente através da interface **DII** (Interface de Invocação Dinâmica) [GTA-UFRJ,2000].

2.2.4- REPOSITÓRIO DE INTERFACES

O Repositório ou depósito de Interfaces é utilizado para armazenar informações de todas as definições dos seus objetos [GTA-UFRJ,2000].

Fornece objetos que representam informações da IDL (Interface Definition Language), disponíveis a tempo de execução. A informação é utilizada pelo ORB (Object Request Broker) e para realizar uma requisição, verificar os tipos de parâmetros, as relações de heranças entre interfaces e auxiliar no processo de interação entre os vários ORB existentes. Estas informações permitem que o programa encontre objetos onde a interface era desconhecida quando ocorreu a compilação, sendo assim possível determinar que operações são válidas para este objeto e invocá-lo [RICCIONI,2000].

Na figura 1, mostramos como as interfaces e as informações referentes as implementações são disponibilizadas ao cliente e às implementações de objeto.

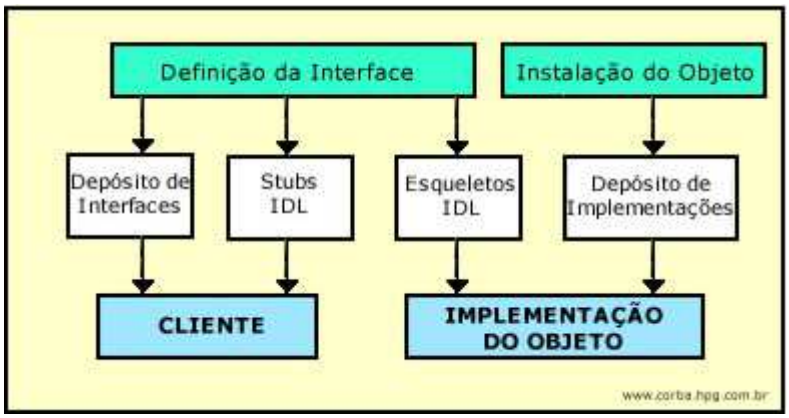

Figura 1 Repositório de Interfaces e de Implementações **[RICCIONI, 2000]**

2.2.5 STUB CLIENTE

Está localizado no lado do cliente, e quando este cliente deseja chamar um método de um objeto, basta ele indicar qual objeto deseja. O stub tem como função receber a chamada, localizar o objeto desejado, transformar a chamada para poder ser enviada pela rede e transmiti-la ao ORB (Object Request Broker) para dar continuidade ao processo [GTA-UFRJ,2000].

Tem como característica promover interfaces estáticas para criar e enviar requisições correspondentes dos serviços desejados do cliente para o servidor. O Stub é criado utilizando-se um compilador IDL (Interface Dinamic Language).

Os stubs fazem chamadas no resto da ORB (Object Request Broker) usando interfaces privadas e otimizadas para o núcleo ORB particular. Se mais de uma ORB está disponível, podem existir diferentes stubs correspondentes a diferentes ORBs. Neste caso, é necessário que a ORB e o mapeamento de linguagem cooperem para associar os stubs corretos com a referência de objeto correta [RICCIONI,2000].

2.2.6 LINGUAGEM DE DEFINIÇÃO DE INTERFACE

CORBA utiliza a *IDL* (*Interface Definition Language*) como uma forma de descrever interfaces, isto é, de especificar um contrato entre os objetos. IDL é uma linguagem puramente declarativa baseada em C++. Isso garante que os componentes em CORBA sejam

auto-documentáveis, permitindo que diferentes objetos, escritos em diferentes linguagens, possam interoperar através das redes e de sistemas operacionais [MONTEZ, 1998].

A figura 2 mostra como a IDL provê independência de linguagem de programação entre os objetos.

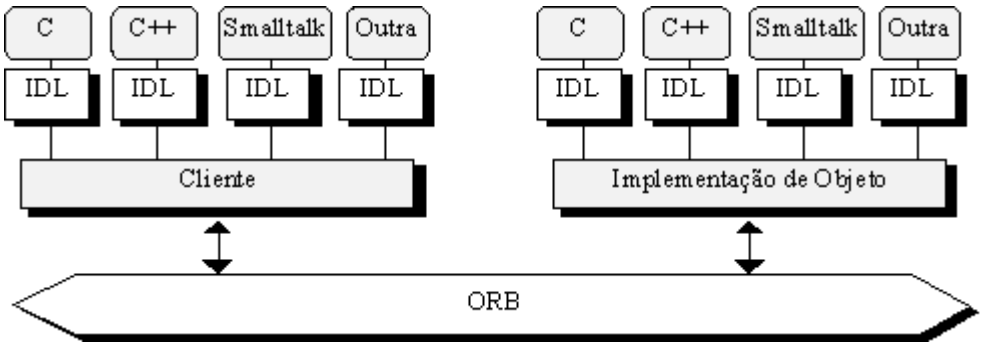

Figura 2 Independência de linguagem de programação [MONTEZ, 1998]

Uma definição de interface escrita em IDL define completamente a interface e especifica cada parâmetro da operação. É importante ressaltar que os objetos não são escritos em IDL, que é uma linguagem puramente descritiva. Eles são escritos em linguagens que possuem mapeamentos definidos dos conceitos existentes em IDL [MONTEZ, 1998].

A seguir daremos um exemplo de especificações usando IDL:

```
/*************************/
/* Arquivo impressao.idl */
/*************************/
exception SemPapel {};
exception PapelEnroscou {
   short posicao;
};

typedef sequence<octet> Arquivo;

interface Impressora {
   void print (in Arquivo arq) raises (SemPapel, PapelEnroscou);
   int  estado (out FilaDeImpressao fdi);
}

interface ImpressoraColorida : Impressora {
   enum ModoDeCores { BrancoEPreto, TonsDeCinza, Colorido };
   void modo (inout ModoDeCores mdc);
}
```

2.2.6.1 ESPECIFICAÇÕES IDL

"Uma especificação IDL é independente do ORB e da linguagem utilizada na implementação do cliente e da implementação do objeto, assim como da máquina e do sistema operacional no qual estes se executam" [RICCIONI,2000].

"As especificações IDL são formadas por definições de tipos, constantes, exceções, módulos e interfaces. Este conjunto de definições compõem um arquivo de extensão .idl, que contém a informação necessária para o interfaceamento entre o cliente e a implementação de objeto, tendo como elemento intermediário o ORB" [RICCIONI,2000].

2.2.7 ORB (OBJECT REQUEST BROKER)

O ORB (Object Request Broker) é a estrutura mais importante da arquitetura CORBA. Ele é responsável por todos os mecanismos requeridos para encontrar o objeto, preparar a implementação de objeto para receber a requisição, e executar a requisição.

O cliente enxerga a requisição de forma independente de onde o objeto está localizado, qual linguagem de programação ele foi implementado, ou qualquer outro aspecto que não está refletido na interface do objeto. Também, é função do ORB, o retorno de parâmetros de saída da requisição para o cliente, se assim houver [RICCIONI, 2000].

O ORB permite que seja feito de forma transparente e heterogênea, requisições a objetos que podem ser localizados localmente ou remotamente. Com isso o cliente não necessita ter conhecimento de quais os mecanismos utilizados para se comunicar com o objeto desejado.

O ORB se estende desde o Stub IDL do cliente, onde é invocado o serviço, até o Skeleton no servidor, sendo responsável pela comunicação entre esses componentes.

A figura 3, mostra a estrutura e o funcionamento de um ORB que se segue da seguinte maneira: O cliente invoca uma requisição ao ORB através do stub ou através da interface de invocação dinâmica ou diretamente ao ORB através da interface do ORB. Ao

receber a requisição o ORB localiza o código da implementação, transmite parâmetros e transfere o controle para a implementação de objetos. O ORB faz uma invocação a implementação de objetos através de um Esqueleto Dinâmico ou Estático que após a requisição ser recebida pelo objeto e executada, ele receberá a objeto de volta e o transmitirá de volta ao ORB, passa pelo Stub e chegará ao cliente. Após a implementação de objetos ter executado o serviço, a implementação de objetos terá acesso aos serviços oferecidos pelo ORB através de um adaptador de objetos [GTA-UFRJ, 2000].

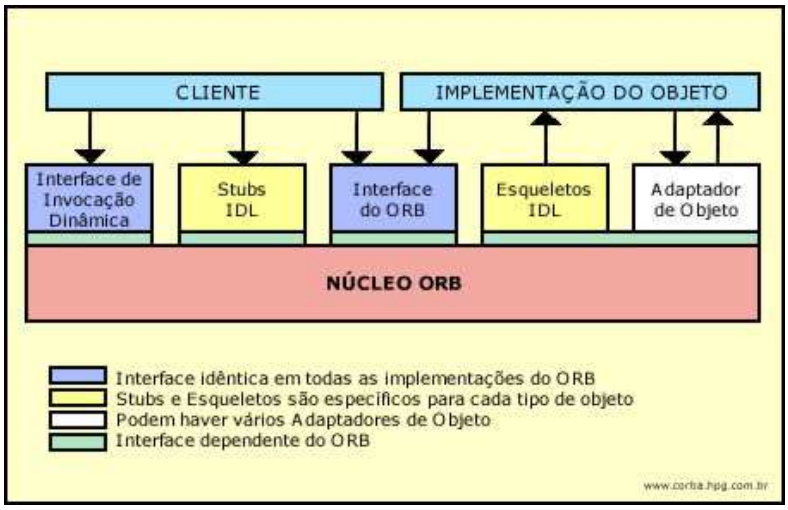

Figura 3 Estrutura de um ORB [RICCIONI,2000]

Para fazer uma requisição, um cliente pode usar a *Interface de Invocação Dinâmica* (DII - *Dynamic Invocation Interface*) ou um *Stub de IDL*. Para algumas poucas e determinadas funções, um cliente pode interagir diretamente com a interface do ORB. A interface o ORB oferece funções que são independentes do adaptador de objetos utilizado, como, por exemplo, funções que executam algumas operações sobre referências de objetos [GTA-UFRJ,2000].

A requisição só poderá ser feita por um cliente, se ele conhecer a referência do objeto e da operação que deseja executar. Em seguida, o cliente pode chamar a rotina stub, ou formular a requisição dinamicamente através da DII. O método utilizado para a implementação de objetos deve ser bem transparente, já que os fatores que os diferenciam são tratados internamente pelo ORB [RICCIONI,2000].

Feita a requisição, o ORB encontra o código da implementação de objetos, transmite os parâmetros e transfere o controle para essa implementação de objetos através de um esqueleto IDL específico à interface e ao adaptador de objeto.

Com o serviço solicitado já executado, a implementação de objetos terá acesso aos serviços oferecidos pelo ORB através de um adaptador de objetos. Com a existência de vários adaptadores de objetos, a escolha para utilizar um destes, depende do serviço que a implementação de objetos deseja obter do ORB[RICCIONI, 2000].

O ORB utiliza a mesma interface para todas as implementações, independente do adaptador de objeto utilizado. Por meio dessa interface podem ser requeridas operações executadas pelo ORB necessárias tanto para os clientes quanto para as implementações de objeto.

O núcleo ORB oferece os mecanismos de representação de objetos e de comunicação para que se torne possível o processamento e a execução das requisições [RICCIONI,2000].

2.2.7.1 INTERFACE ORB

A interface ORB é a interface que vai direto a ORB e é a mesma para todas ORBs, não dependendo da interface do objeto nem do adaptador de objeto. A maioria das funcionalidades da ORB são fornecidas através dos adaptadores de objeto, stubs, esqueletos ou invocações dinâmicas e por isso existem apenas algumas operações que são comuns a todos os objetos. Essas operações são úteis para clientes e implementações de objetos [GTA-UFRJ].

2.2.8 SKELETON

Tem uma função parecida com o Stub do cliente, porém está situado do lado do servidor. O Skeleton tem a função de disponibilizar os métodos dos objetos servidores para o resto do sistema e encontrar os serviços que são solicitados pelo Stub.

“O Skeleton funciona da seguinte maneira: recebe um pacote do ORB, envia para a implementação de objetos, a requisição será recebida pelo objeto que irá executa-la, quando receber a resposta ele enviará o objeto de volta ao skeleton, passa pelo ORB, pelo stub e chegará ao cliente” [GTA-UFRJ,2000].

2.2.9 INVOCAÇÃO

Invocação nada mais é que um processo utilizado para fazer uma requisição a um objeto qualquer para que possa ser feito um acesso a um serviço de um determinado objeto. Existe dois tipos de processos de invocações: Invocação Estática e Invocação Dinâmica.

O fato de permitir tanto a invocação dinâmica quanto a estática, torna CORBA bastante flexível. A invocação estática, que utiliza *Stubs* de IDL que podem ser gerados automaticamente através de pré-compiladores, possui uma série de vantagens sobre a invocação dinâmica: é mais fácil de programar, faz verificação de tipos mais robusta, executa mais rápido e é auto-documentável. Já a invocação dinâmica permite a adição de novos serviços às implementações de objetos sem alterações nos clientes. Dessa forma, os clientes podem descobrir em tempo de execução quais são os serviços oferecido [MONTEZ, 1998].

Antes de se entrar em detalhes das invocações estáticas e dinâmicas, é importante falar um pouco sobre três formas de invocações que existem no CORBA:

Invocação Síncrona: O cliente faz a chamada e tem que ficar travado esperando o resultado.

Invocação Síncrona Deferida: O cliente ao realizar uma chamada não fica travado, esperando pelo resultado, podendo continuar a executar normalmente as chamadas.

Invocação ONEWAY: é feita pelo cliente quando ele não necessita receber o resultado de uma chamada, deixando o cliente livre para continuar o seu processo [GTA-UFRJ].

2.2.9.1 INVOCAÇÃO ESTÁTICA

- A invocação estática é realizada por uma chamada ao stub que tem como objetivo acessar uma operação sobre um objeto [GTA-UFRJ,2000].

Passos para Invocação Estática:

- Definição da classe de objetos usando IDL;
- Execução do arquivo IDL através de um compilador;
- Adicionamento do código de implementação para o skeleton;
- Compilação do código;
- Ativação das definições da classe através do Repositório de Interfaces;
- Instanciação do objeto no servidor;
- Registro em tempo de execução do objeto no Repositório de Interfaces [GTA-UFRJ,2000].

2.2.9.2 INVOCAÇÃO DINÂMICA

A Invocação Dinâmica surgiu para corrigir um problema da Invocação Estática. Esse problema é que quando se necessitava de inserir um novo objeto, era necessário parar o sistema, já com a Invocação Dinâmica, é possível inserir um novo objeto em tempo de execução sem precisar parar o sistema. Temos dois tipos de interface para esse tipo de invocação:

2.2.9.2.1-INTERFACE DE INVOCAÇÃO DINÂMICA (DII)

“Trata-se de uma interface que permite, através de um acesso direto aos mecanismos de requisição providos por um ORB, descobrir ou construir dinamicamente uma invocação a um objeto, permitindo a inclusão do mesmo sem que seja necessário um conhecimento prévio de quais objetos ou métodos estão disponibilizados. Nesse processo, o

cliente não precisa utilizar as rotinas do Stub, apesar de usar a mesma semântica que é utilizada nele, diferenciando apenas no ponto de que, em uma invocação dinâmica, os parâmetros repassados são todos checados em tempo de execução e não de compilação" [GTA-UFRJ,2000].

Para se executar esse tipo de invocação, é necessário passar pelos seguintes caminhos:

-Identificar o Objeto Alvo;

-Pegar a interface do Objeto ;

-Construir uma invocação ou chamada;

-Invocar a requisição e receber o resultado (quando houver).

A DII permite que o cliente realize operações de envio e recebimento de forma separada e também operações unidirecionais. Um ponto de desvantagem, em comparação com as interfaces de invocações estática, com relação às invocações dinâmicas, é justamente que, esta última, acarreta um maior tráfego na rede, já que necessita realizar acessos adicionais ao Repositório de Interfaces, além do fato de que verificar os parâmetros em tempo de execução prejudica também um pouco o sistema [GTA-UFRJ,2000].

2.2.9.2.2 INTERFACE SKELETON DINÂMICA (DSI)

Igual ao DII, mas os recursos são oferecidos para os objetos servidores. A Interface de Esqueleto Dinâmica (DSI - *Dynamic Skeleton Interface*) é uma forma de se enviar requisições de um ORB para uma implementação de objetos que não possui informações sobre a implementação do objeto em tempo de compilação. O cliente que invoca um objeto, não pode determinar se a implementação está usando um esqueleto de um tipo específico ou DSI para conectar à implementação ao ORB. DSI é útil em soluções de interoperabilidade implementando pontes entre diferentes ORBs, e para suportar linguagens de tipos dinâmicos [GTA-UFRJ,2000].

2.2.10-ADAPTADOR BÁSICO DE OBJETOS

O adaptador básico de objetos é a interface através do qual uma implementação de objetos acessas funções do ORB, bem como também ajuda na ativação de objetos [RICCIONI,2000].

O adaptador de objetos possui alguns tipos de serviços:

- **Geração de referência a objetos:** gera referência a objetos

- **Registro de implementação de objetos:** localiza a implementação do objetos

- **Ativar e desativar uma implementação de objetos:** ativa e desativa um determinado objeto quando necessário

- **Capacidade de invocações:** tem a capacidade de invocar objetos e é auxiliado pelo skeleton [GTA-UFRJ,2000].

Com o surgimento de vários adaptadores de objetos, a OMG propôs a criação do BOA (Basic Object Adapter) que está disponível em todas as implementações de ORB´s o que o torna dependente deste.

O BOA é um adaptador “genérico”, sendo possível criar adaptadores especializados para determinadas especificações como, por exemplo, para objetos C++ e OODB [GTA-UFRJ,2000].

Na figura 4, é mostrado como o adaptador de objetos participa da invocação de métodos utilizando como intermediário o esqueleto IDL. Neste caso, a função do adaptador é ativar a implementação e autenticar a requisição recebida, verificando se cliente possui permissão para acessar o serviço requisitado.

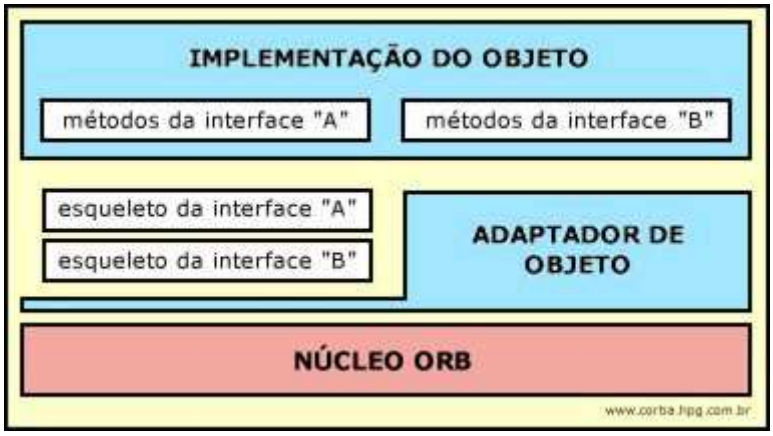

Figura 4 Estrutura de um Adaptador de Objetos [RICCIONI, 2000]

2.2.10.1 ESTRUTURA DO ADAPTADOR BÁSICO DE OBJETOS

O BOA interage com o núcleo ORB para promover uma série de serviços às implementações de objetos [RICCIONI,2000].

São de responsabilidade do BOA operações como: a ativação das implementações e o registro destas junto ao ORB, a ativação de objetos, a utilização dos esqueletos IDL para chamar os métodos do objeto, e o interfaceamento de serviços providos pelo ORB e pelo núcleo. E é exatamente o que a figura 5 mostra.

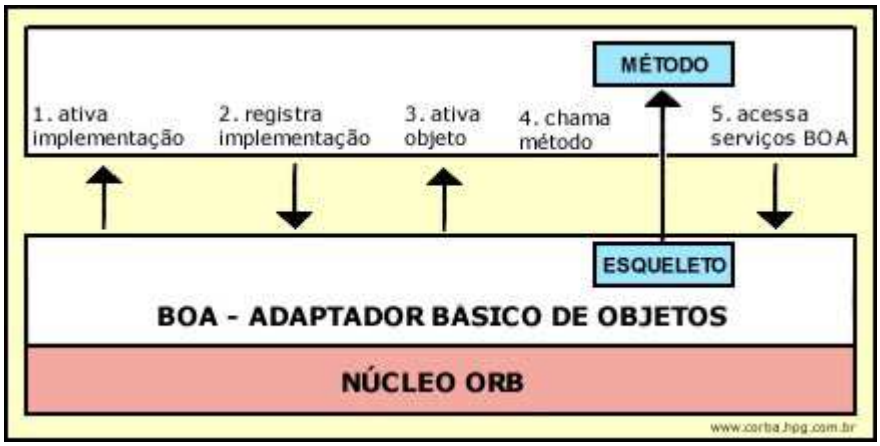

Figura 5 Estrutura Operacional do BOA [RICCIONI,2000]

As implementações podem ser ativadas pelo BOA segundo 4 políticas de ativação:

- **Política compartilhada de servidor:** múltiplos objetos podem ser acessados ao mesmo tempo;

- **Política Não-compartilhada de servidor:** o objetos só podem ser acessados um por vez;

- **Política de Servidor por método:** cada invocação de um método inicia um servidor, que é deslocado quando o método termina, independentemente do fato de uma implementação de objeto estar ativa; não e necessário notificar o BOA quando o objeto inicia ou termina;

- **Política de Servidor Persistente:** o servidor fica sempre ativo [RICCIONI,2000].

2.2.11 IMPLEMENTAÇÃO DO OBJETO

Provê a semântica do objeto, definindo dados para a instância do objeto e código para os métodos. As diversas implementações de objetos que podem ser suportadas são: servidores separados, bibliotecas, um programa por método, aplicação encapsulada, SGBDOO, etc. Pode-se suportar diversos estilos de implementação com a definição de adaptadores de objetos adicionais [RICCIONI,2000].

A portabilidade também é importante: implementações de objetos são portáteis entre quaisquer ORBs que suportem o mapeamento de linguagem adequado. A não dependência das implementações dos objetos em relação aos ORBs se dá através da existência dos Adaptadores de Objetos [RICCIONI,2000].

Na figura 6, mostramos como uma implementação de objeto encapsula o estado e o comportamento do objeto, permitindo o contato com o meio externo através de suas interfaces. Os métodos implementados pelo objeto são ativados pelo ORB quando ocorre um requisição, através dos esqueletos IDL. A interação do objeto com o ORB permite o estabelecimento de identidade, a criação de novos objetos, e a obtenção de serviços do ORB através de um adaptador de objetos.

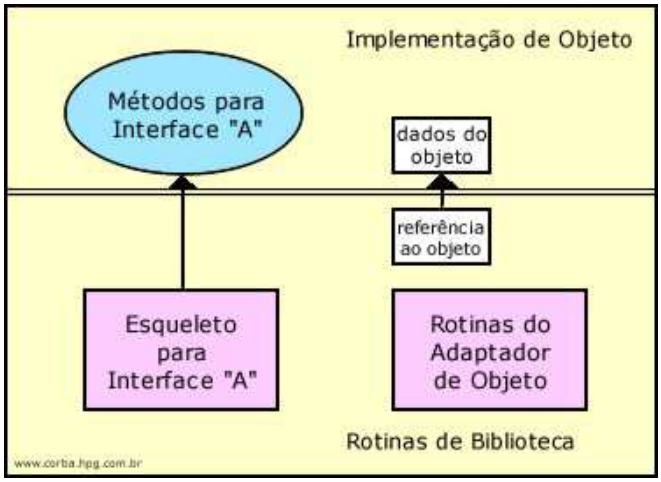

Figura 6 Estrutura de uma Implementação de Objeto CORBA [RICCIONI,2000]

2.2.12 REPOSITÓRIO DE IMPLEMENTAÇÕES

Contém informações que permitem que a ORB localize e ative as implementações de objetos. Embora a maioria das informações do repositório de interface seja específica de uma ORB ou de um ambiente de operação, o repositório de implementação é um lugar conveniente para se gravar certas informações. Geralmente, a instalação de implementações e controle de políticas relacionadas com a ativação e execução da implementação de objeto são através de operações no repositório de implementação.

O repositório de implementação é um bom lugar para guardar informações adicionais, associadas com a implementação de objetos ORB [RICCIONI,2000].

Na figura 7, mostramos como as interfaces e as informações referentes as implementações são disponibilizadas ao cliente e às implementações de objeto.

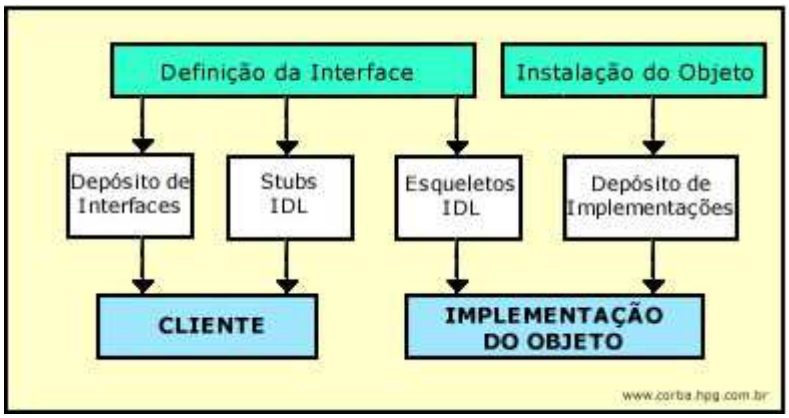

Figura 7 Repositório de Interfaces e de Implementações [RICCIONI,2000]

2.2.13 INTEROPERABILIDADE

Com o avanço da tecnologia CORBA, foram surgindo no mercado ORB's de diferentes fabricantes. Entretanto, a primeira versão do CORBA só trata de interoperabilidade entre ORB's de um mesmo fabricante, com isso surge a necessidade de novos padrões, conceitos que permitam a relação entre esses novos ORB's [GTA-UFRJ,2000].

A característica de interoperabilidade entre objetos só foi coberta na versão 2.0 do CORBA, introduzida em dezembro de 1994. Isso se deu através da especificação de uma arquitetura de interoperabilidade, um suporte para pontes, que serve para comunicação entre ORBs, um protocolo para comunicação entre ORBs genérico e um protocolo para comunicação entre ORBs para *Internet*.

A *arquitetura de interoperabilidade* do CORBA identifica claramente a regra de diferentes tipos de domínios para informação específica de ORBs. Tais domínios podem incluir domínios de referências de objeto, domínios de tipos, domínios de segurança, dentre outros. Quando dois ORBs estão no mesmo domínio, eles podem se comunicar diretamente. Entretanto, quando a informação em uma invocação deve deixar seu domínio, a invocação deve atravessar uma ponte [MONTEZ, 1998].

A regra de uma ponte deve assegurar que o conteúdo e a semântica sejam mapeados adequadamente de um ORB para outro. O suporte para ponte entre ORBs pode também ser usado para prover interoperabilidade com outros sistemas não CORBA.

"O protocolo entre ORBs genérico (GIOP - Generic Inter-ORB Protocol) especifica uma sintaxe de transferência padrão e um conjunto de formatos de mensagens para comunicação entre ORBs. O protocolo entre ORBs para Internet (IIOP - Internet Inter-ORB Protocol) especifica como mensagens GIOP são trocadas usando conexões TCP/IP. Ele especifica um protocolo de interoperabilidade padrão para Internet" [MONTEZ, 1998].

O relacionamento entre GIOP e IIOP é similar ao mapeamento existente entre o OMG IDL e uma linguagem específica: o GIOP pode ser mapeado em diferentes protocolos de transportes, e especifica os elementos de protocolo que são comuns a todos os mapeamentos. Entretanto, o GIOP não fornece uma completa interoperabilidade, da mesma forma que IDL não pode ser usado para se construir programas completos. O IIOP, e outros mapeamentos similares para diferentes protocolos de transportes, são realizações concretas das definições abstratas de GIOP [MONTEZ, 1998].

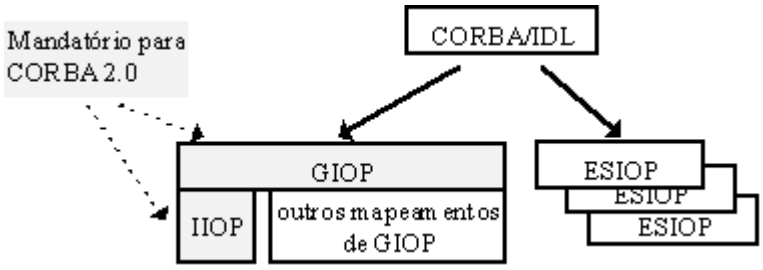

Figura 8 Relacionamentos de protocolos entre ORBs [RICCIONI,2000]

Os protocolos entre ORBs para ambientes específicos (ESIOP - Environment-Specific Inter-ORB Protocol) devem ser usados para interoperar em locais onde uma rede ou infra-estrutura de computação distribuída já esteja em uso. Apesar de cada ESIOP poder ser otimizado para um ambiente em particular, toda especificação ESIOP deve estar conforme as convenções da arquitetura de interoperabilidade para facilitar o uso de pontes, se necessário. O suporte de pontes entre ORBs habilita às pontes serem construídas entre domínios de ORBs que usam IIOP e domínios que usam um ESIOP particular [MONTEZ, 1998].

2.3 OUTROS TIPOS DE ARQUITETURAS

Além do CORBA existem outros tipos de arquitetura para serem utilizadas com o sistema distribuído. A seguir falaremos um pouco dessas arquiteturas.

2.3.1 ARQUITETURA DCOM

DCOM (Distributed Component Object Model) tem suas raízes nas tecnologias de objetos da Microsoft: o COM (Component Object Model), OLE (Object Linking and Embedding) e ActiveX (COM habilitado para a Internet).

O DCOM é a versão distribuída da tecnologia COM, possibilita a comunicação de objetos em processos diferentes e a execução em computadores distintos, interconectados numa rede. Ou seja, o DCOM é uma evolução do COM, dispõe de suas mesmas características somadas a possibilidade de comunicação em máquinas distintas.

2.3.2 ARQUITETURA EJB

EJB (Enterprise Java Beans) é um modelo de componentes no lado do servidor que trata de problemas que envolvem o gerenciamento de objetos comerciais distribuídos em uma arquitetura de várias camadas.

Os Componentes EJB são componentes Java que são executados em um servidor (servidor de aplicativos ou de banco de dados).

EJB pode funcionar e ser executado em qualquer ambiente que possua um interpretador Java (uma máquina virtual Java - JVM) e um contêiner EJB.

Com EJB os problemas complexos, como a comunicação de componentes, gerenciamento de transações e gerenciamento de thread, não são mais responsabilidade do desenvolvedor e sim do contêiner.

3 DESENVOLVIMENTO ORIENTADO A OBJETO

3.1- INTRODUÇÃO

A UML (*Unified modeling language)* é a linguagem padrão para especificar, visualizar, documentar e construir artefatos de um sistema e pode ser utilizada com todos os processos ao longo do ciclo de desenvolvimento, e através de diferentes tecnologias de implementação. A modelagem é a parte central de todas as atividades que levam à implantação de um bom software [FURLAN, 1998].

A UML disponibiliza uma forma padrão de modelagem de projetos de Sistemas, incluindo seus aspectos conceituais tais como processos de negócios e funções do sistema, além de itens concretos como as classes escritas em determinada linguagem de programação, processos de banco de dados e componentes de software reutilizáveis.

3.2 DIAGRAMAS

Quando uma modelagem é desenvolvida simplifica-se a realidade para um melhor entendimento do sistema em desenvolvimento. Em UML você desenvolve seus modelos a partir de blocos distintos tais como: classes, interfaces, colaborações, componentes, dependências, generalizações, associações, etc. Os diagramas são meios utilizados para visualização de blocos de construção [BOOCH, 2000].

Depois de definido qual aplicação será desenvolvida, deverá ser definido os diagramas que serão criados para captar detalhes essenciais de cada visão.

Os diagramas utilizados pela UML são compostos de nove tipos: diagramas de casos de uso, classes, objetos, estados, seqüência, colaboração, atividade, componentes e execução. A seguir apresentaremos cada um destes tipos de diagramas.

3.2.1 DIAGRAMA DE CASO DE USO

Os diagrama de caso de uso exibem a visão externa do sistema e suas interações com o mundo exterior descrevendo seus requerimentos e suas responsabilidades. Eles possuem três elementos principais:

Ator - agente que interage com o sistema

Caso de Uso - o comportamento da classe

Interação - envio e recebimento de mensagens da comunicação ator - sistema.

Os casos de uso são usados para modelar como um sistema ou empresa funciona, ou como os usuários desejam que ele funcione. Os casos de uso geralmente são o ponto de partida da análise orientada a objetos utilizando a UML [BOOCH 2000].

O modelo de caso de uso consiste de atores e casos de uso. Os atores representam usuários e outros sistemas que interagem com sistema modelado. Os casos de uso mostram o comportamento do sistema, cenários que o sistema percorre em resposta ao estímulo de um ator. Eles são desenhados como elipses.

No modelo de caso de uso o relacionamento entre um ator e um caso de uso representa a participação deste ator no caso de uso. Além deste relacionamento, existe dois outros tipos de relacionamentos entre casos de uso:

- o relacionamento extends: é representado graficamente por uma seta como estereótipo <<extends>>, mostrando que o caso de uso destino pode incluir o comportamento especificado pelo caso de uso origem;

- o relacionamento uses: é representado por uma seta com o estereótipo <<uses>>, mostrando que o caso de uso origem inclui o comportamento especificado pelo caso de uso destino

A figura 9 mostra um exemplo de diagrama de casos de uso. Os casos de uso modelados são de um sistema de validação de cartão de crédito. O sistema possui os atores Cliente, Instituição de venda a varejo e Instituição financeira patrocinadora. Esses atores são responsáveis pelos seguintes casos de uso: realiza a transação com o cartão, processa a conta do cliente, reconcilia as transações e gerencia a conta do cliente.

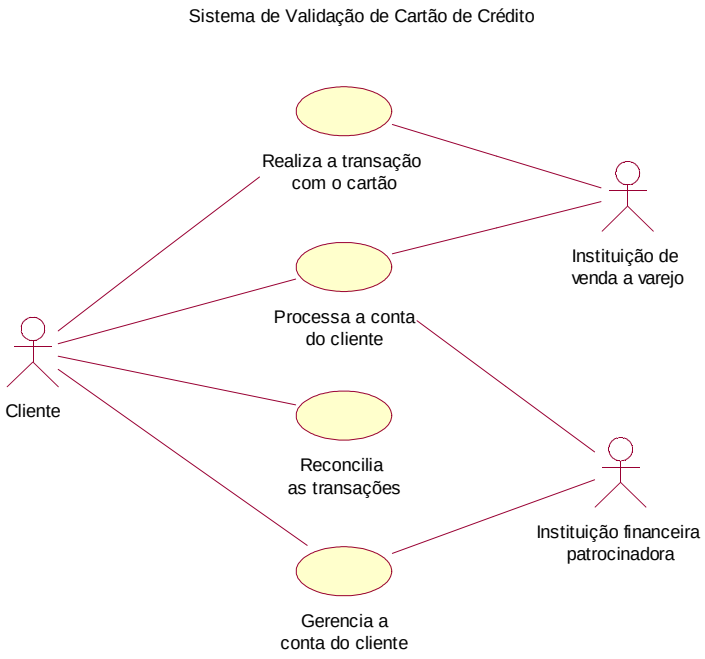

Figura 9 Diagrama de Casos de Uso [BOOCH, 2000]

3.2.2 DIAGRAMA DE CLASSE

Um diagrama de classe mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos.

O diagrama de classe é apenas um tipo especial de diagrama e compartilha as mesmas propriedades de todos os outros diagramas. O que diferencia os diagramas de classes dos outros tipos é o seu conteúdo particular [BOOCH, 2000].

Os diagramas de classes costumam conter os seguintes itens:

- Classes;
- Interfaces;
- Colaborações e
- Relacionamento de dependência, generalização e associação

Uma classe em UML é representada por uma caixa retangular com três compartimentos: um com o nome da classe, o outro com a lista de atributos da classe e o último com a lista de operações da classe.

As associações representam relacionamentos estruturados entre objetos de diferentes classes, e são representados graficamente através de uma linha conectando as classes. Uma associação pode ter um nome. E as extremidades da linha que representa uma associação pode ter nome de papéis mostrando como a classe é vista pelas outras classes na associação.

A multiplicidade de uma associação especifica quantas instâncias de uma classe relacionam-se a uma única instância de uma classe associada. Uma multiplicidade é representada por um intervalo de valores possíveis, no seguinte formato: limite_inferior..limite_superior, onde esses limites são valores inteiros (o caracter * pode ser usado como limite_superior para indicar falta de limite).

A agregação é uma forma especial de associação que representa o relacionamento todo-parte entre objetos. Ela é representada incluindo-se um losango na extremidade do objeto todo do relacionamento todo-parte.

A generalização é uma ferramenta poderosa para a abstração. A generalização é um relacionamento existente entre uma classe mais geral(superclasse) e uma classe mais específica(subclasse), onde a classe mais específica é consistente com a mais geral e adiciona informações a ela. Uma generalização é representada por uma linha com um triângulo, que liga a classe mais específica a mais genérica [FURLAN, 1998].

A figura 10 mostra um exemplo de diagrama de classes. Esse diagrama de classes mostra a modelagem de uma venda. O cliente realiza a compra de um produto através de um pedido. A classe Cliente se relaciona com a classe pedido através da associação denominada realiza. A classe Pedido se relaciona com a classe Produto através de uma agregação onde à classe Pedido representa “o todo” e a classe Produto “as partes”.

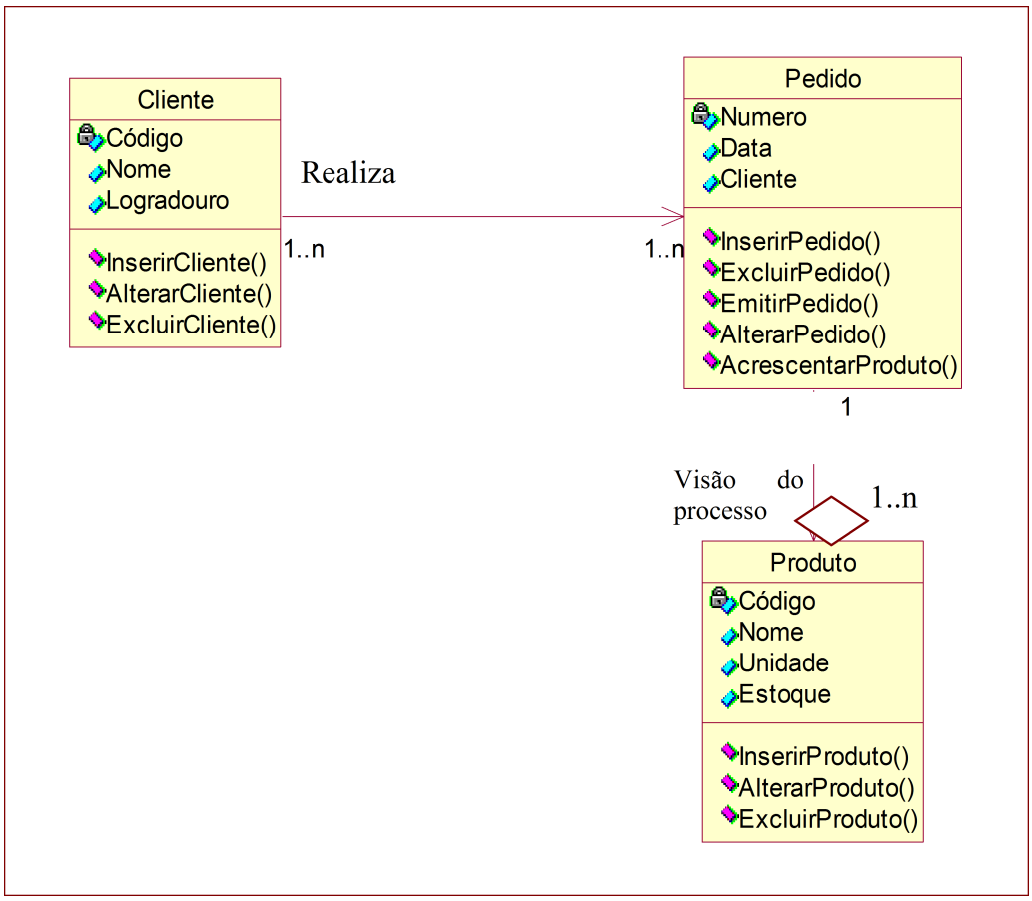

Figura 10 Diagrama de Classes

3.2.3 DIAGRAMA DE OBJETO

Os diagramas de objetos fazem a modelagem de instâncias de itens contidos em diagramas de classes. Um diagrama de objetos mostra um conjunto de objetos e seus relacionamentos em determinado ponto no tempo.

Estes diagramas não são importantes apenas para a visualização, especificação e documentação de modelos estruturais, mas também para a construção de aspectos estáticos de sistemas por meio de engenharia de produção e engenharia reversa [Booch, 2000].

A UML permite a utilização de diagramas de classes para uma visualização dos aspectos estáticos dos blocos de construção do sistema. Você usa os diagramas de interação

para visualizar os aspectos dinâmicos de seu sistema, formados por instâncias desses blocos de construção e mensagens enviadas entre eles. Os diagramas de objetos cobrem um conjunto de instâncias dos itens encontrados nos diagramas de classes. O diagramas de objetos, portanto, expressa a parte estática de uma interação, composta pelos objetos que colaboram entre si, mas sem qualquer uma das mensagens passadas entre eles.

Os diagramas de objetos são usados para fazer a modelagem da visão de projeto estática ou da visão de processo estática de um sistema, da mesma forma como faz com os diagramas de classes, mas a partir da perspectiva de instâncias reais ou prototípicas. Isso envolverá a modelagem de um retrato do sistema em determinado momento e a representação de um conjunto de objetos, seus estados e relacionamentos. Essa visão atende principalmente aos requisitos funcionais do sistema – ou seja, os serviços que o sistema deverá proporcionar aos seus usuários finais. Os diagramas de objetos permitem que você faça a modelagem de estruturas de dados estáticos.

A figura 11 mostra um exemplo de diagramas de objetos. Esse diagrama mostra o relacionamento da universidade com seus departamentos. A universidade se relaciona tanto com o departamento DCC quanto com o departamento DE.

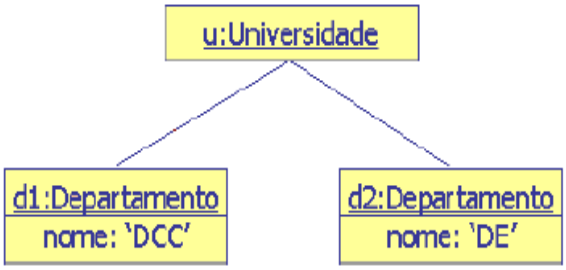

Figura 11 Diagrama de Objeto

3.2.4 DIAGRAMA DE INTERAÇÃO

Um diagrama de interação mostra uma interação formada por um conjunto de objetos e seus relacionamentos, incluindo as mensagens que poderão ser trocadas entre eles [BOOCH, 2000].

O diagrama de interação enfatiza a interação de objetos, uma especificação comportamental que inclui uma seqüência de trocas de mensagens entre objetos dentro de um contexto a fim de realizar um determinado propósito.

Devemos utilizar diagramas de interação quando necessitamos ver o comportamento de vários objetos dentro de um único caso de uso, a partir de mensagens que são trocadas entre eles. Estes diagramas modelam os aspectos dinâmicos do sistema.

Segundo [BOOCH, 2000] os diagramas de interação são utilizados para modelar os aspectos dinâmicos do sistema, isso envolve a modelagem prototípica de classes e interfaces.

Os diagramas de interação podem ser apresentados de duas formas: Diagrama de Seqüência e Diagrama de Colaboração [BOOCH, 2000].

3.2.4.1 DIAGRAMA DE SEQUÊNCIA

Um diagrama de seqüência é um diagrama de interação que dá ênfase à ordenação temporal de mensagens. Um diagrama de seqüência mostra conjunto de objetos e as mensagens enviadas e recebidas por esses objetos. Tipicamente os objetos são instâncias nomeadas ou anônimas de classes, mas também podem representar instâncias de outros itens,

como colaborações, componentes e nós. Os diagramas são utilizados para ilustrar a visão dinâmica de um sistema. [BOOCH 2000].

Na figura 11 é mostrado um exemplo de um diagrama de seqüência. O ator denominado professor realiza o login no sistema. O sistema valida o nome do usuário e a senha.

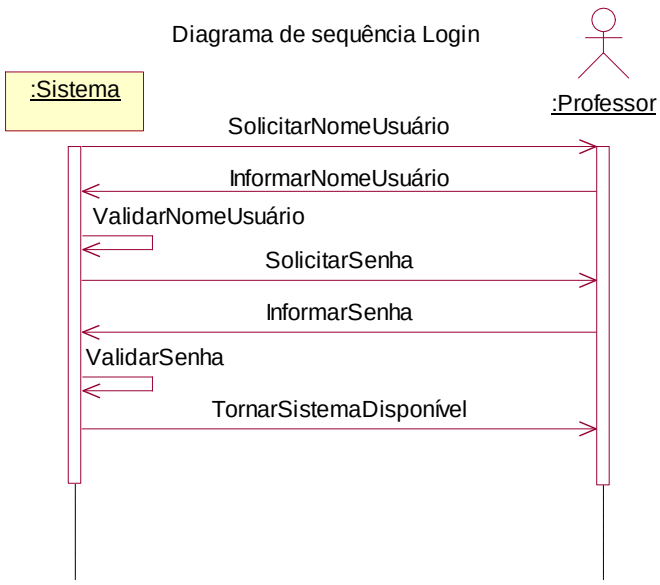

Figura 12 Diagrama de Sequência

3.2.5 DIAGRAMA DE ESTADOS

Os diagramas de estados são empregados para a modelagem dos aspectos dinâmicos de um sistema. Na maior parte, isso envolve a modelagem do comportamento de objetos reativos. Um objeto reativo é aquele cujo comportamento é mais bem caracterizado por sua resposta a eventos ativados externamente ao seu contexto. Um objeto reativo tem um claro tempo de vida cujo comportamento atual é afetado pelo seu passado. Os diagramas de estados podem ser anexados a classes, a casos de uso ou a sistemas inteiros para visualizar, especificar, construir e documentar a dinâmica de um objeto individual.

Um diagrama de estados mostra uma máquina de estados, dando ênfase ao fluxo de controle de um estado para outro. Uma máquina de estados é um comportamento que

especifica as seqüências de estados pelos quais um objeto passa durante seu tempo de vida em resposta a eventos, juntamente com suas respostas a esses eventos. Um estado é uma condição ou situação na vida de um objeto durante a qual ele satisfaz a alguma condição, realiza alguma atividade ou aguarda algum evento. Um evento é uma especificação de uma ocorrência significativa que tem uma localização no tempo e no espaço. No contexto de uma máquina de estados, um evento é uma ocorrência de um estímulo capaz de ativar uma transição de estados. Uma transição é um relacionamento entre dois estados, indicando que um objeto no primeiro estado realizará certas ações e entrará no segundo estado quando um evento específico ocorrer e as condições específicas estão satisfeitas. Uma atividade é uma execução não-atômica em andamento em uma máquina de estados. Uma ação é uma computação atômica executável que resulta em uma alteração do estado do modelo ou no retorno de um valor.

A figura 12 mostra um exemplo de um diagrama de estados. Este diagrama mostra a modelagem de um elevador.

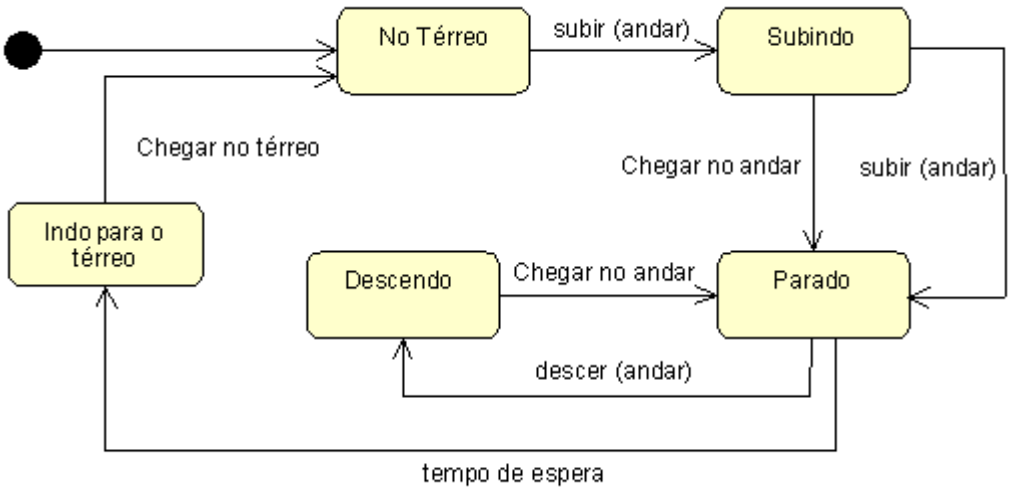

Figura 13 Diagrama de Estados

3.2.6 DIAGRAMAS DE COMPONENTES

Um diagrama de componentes de software mostra um conjunto de componentes e seus relacionamentos.

Os diagramas de componentes são empregados para a modelagem da visão estática de implementação de um sistema. Isso envolve a modelagem de itens físicos que residem em um nó, como executáveis, bibliotecas, tabelas, arquivos e documentos. Os diagramas de componentes são essencialmente diagramas de classes que focalizam os componentes de um sistema [BOOCH 2000].

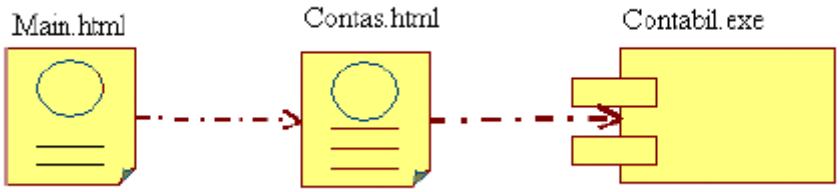

Figura 14 Diagrama de Componente

3.2.7 DIAGRAMA DE IMPLANTAÇÃO

Um diagrama de implantação mostra um conjunto de nós e seus relacionamentos. Usa-se esses diagramas para ilustrar a visão estática da implantação de uma arquitetura. Os diagramas de funcionamento estão relacionado aos diagramas de componentes, pois tipicamente um nó contém um ou mais componentes [BOOCH 2000].

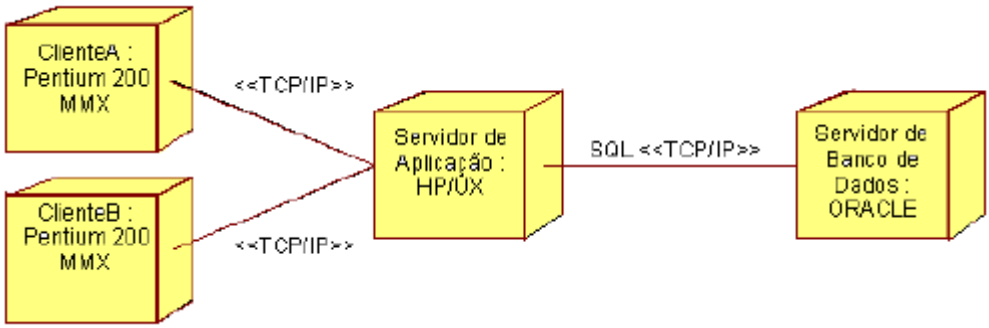

Figura 15 Diagrama de Implantação

4 ESTUDO DE CASO

O estudo de caso é um dos vários modos de realizar uma pesquisa sólida. Outros modos incluem experiências vividas, histórias, e a análise de informação de arquivo. Cada estratégia tem vantagens e desvantagens que dependem de três condições: 1) o tipo de foco da pesquisa; 2) o controle que o investigador tem sobre eventos comportamentais atuais, e 3) o enfoque no contemporâneo ao invés de fenômenos históricos.

Em geral, estudos de casos se constituem na estratégia preferida quando o "como" e/ou o "por que" são as perguntas centrais, tendo o investigador um pequeno controle sobre os eventos, e quando o enfoque está em um fenômeno contemporâneo dentro de algum contexto de vida real.

Estudos de casos podem ser classificados de várias maneiras explicativos, cognitivos, expositivos. Porém o que iremos tratar neste trabalho é "estudo de caso explicativo" .

4.1 MODELAGEM

4.1.1 APLICATIVO DE UMA LOJA DE ELETROELETRÔNICOS

1- O sistema deve cadastrar prateleiras (Código, Nome, Descrição)

2- O sistema deve cadastrar os produtos (Código do Produto, Prateleira, Nome da Prateleira, Nome do Produto, Descrição do produto, NomeFabricante).

3- O sistema deve permitir procurar produto por nome (Nome do Produto)

4- O sistema deve permitir procurar o produto por prateleira (Código, Nome, Descrição).

5- O sistema deve permitir cadastrar os fornecedores (Código, Nome, Endereço, Cidade, Estado, Cep, Telefone, Fax, Email, CGC).

6- O sistema deve permitir a procura dos dados do fornecedor (Código, Nome, Endereço, Cidade, Estado, Cep, Telefone, Fax, Email, CGC).

4.1.2 DIAGRAMA DE CASO DE USO

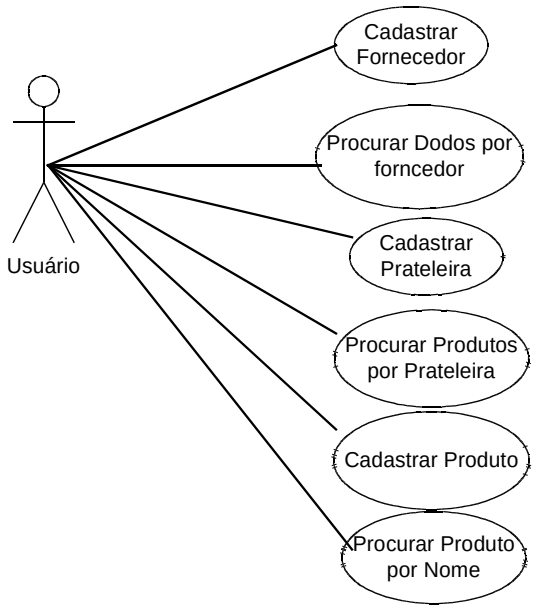

Figura 16 Diagrama de Caso de Uso do Sistema

4.1.3 DIAGRAMA DE CLASSES

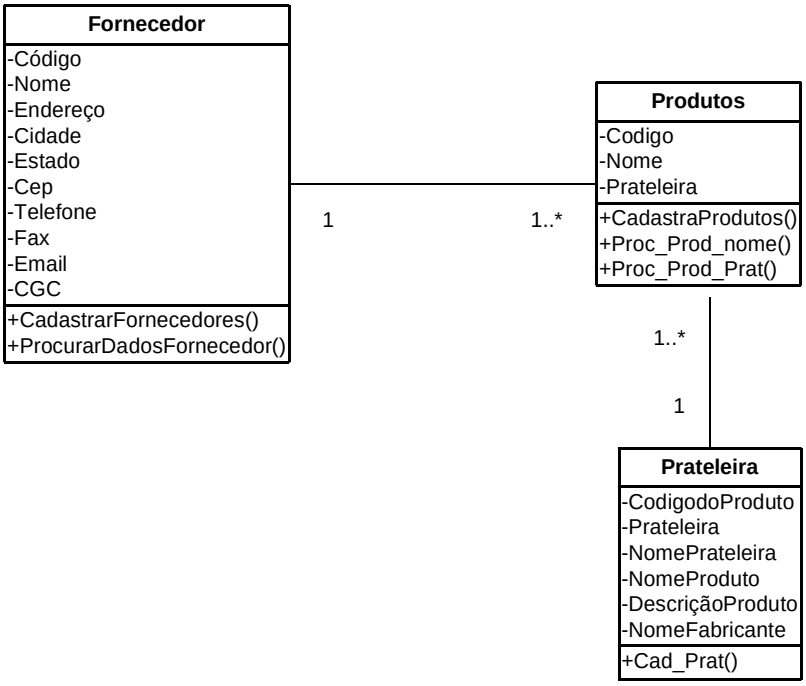

Figura 17 Diagrama de Classe do Sistema

4.1.4 DIAGRAMA DE DISTRIBUIÇÃO

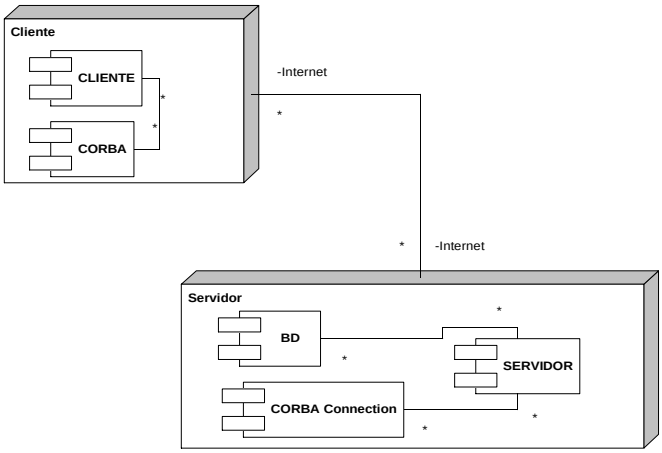

Figura 18 Diagrama de Distribuição do Sistema

4.2 IMPLEMENTAÇÃO

O objetivo dessa aplicação é fazer um aplicativo simples só para testar a utilização do padrão CORBA em uma linguagem de programação.

A aplicação que será desenvolvida terá como objetivo controlar os produtos de uma loja de eletroeletrônicos e possuirá os seguintes módulos:

- Cadastro de Prateleira;
- Cadastro de Produtos;
- Procura de Produtos por Nome;
- Procura de Produtos por Prateleira;
- Cadastro de Fornecedor;
- Procura por Dados do Fornecedor

Dentro do objeto, no servidor, são adicionados os provedores de dados que são ligados a tabelas do banco de dados. Posteriormente, o cliente estabelece uma conexão CORBA com o objetivo no aplicativo servidor e liga o componente ClientDataset, no lado do cliente, com o componente provedor de dados (DatasetProvider) no lado do servidor.

Ao fazer a conexão com o provedor de dados no servidor, o ClientDataset recebe todo o conteúdo da tabela que está conectada ao componente DatasetProvider.

4.2.1 O SERVIDOR

O servidor, é um computador equipado com software que disponibiliza serviços a uma rede de computadores. A figura abaixo mostra a criação de um Aplicativo Servidor.

Figura 19 Janela do Aplicativo Servidor

Em seguida foi inserida uma janela para o objeto Servidor, onde será adicionado dois componentes Table e dois componentes DatasetProvider. O Componente Table é responsável pela conexão com a tabela de banco de dados e o DataSetProvider é o provedor de acesso à tabela disposta por Table , ou seja, ele é o intermediador do aplicativo servidor e aplicativo cliente.

Figura 20 Janela do Servidor

Na Janela servidor abaixo já foram adicionados os componentes Table e DatasetProvider.

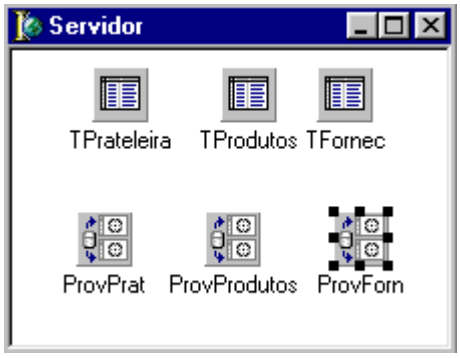

Figura 21 Janela do Servidor com os componentes

4.2.2 O CLIENTE

No cliente foi feito a aplicação de Controle de Produtos de uma loja de produtos eletroeletrônicos. Na figura abaixo estão os botões necessários para que os clientes possam fazer cadastro e procuras necessárias.

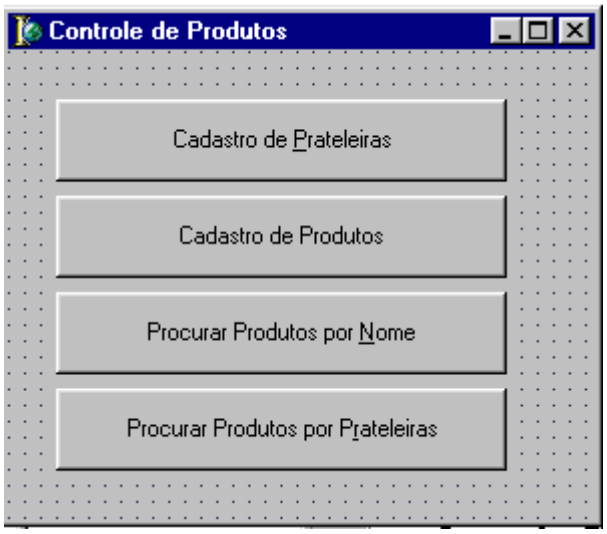

Figura 22 Janela para consultas de produtos

A figura abaixo é para o cadastro de prateleiras que possui código, nome e descrição. No canto superior direito estão os componentes corbaconnection, clientdataset e o datasource que foram usados para fazerem as conecções e ligar as tabelas.

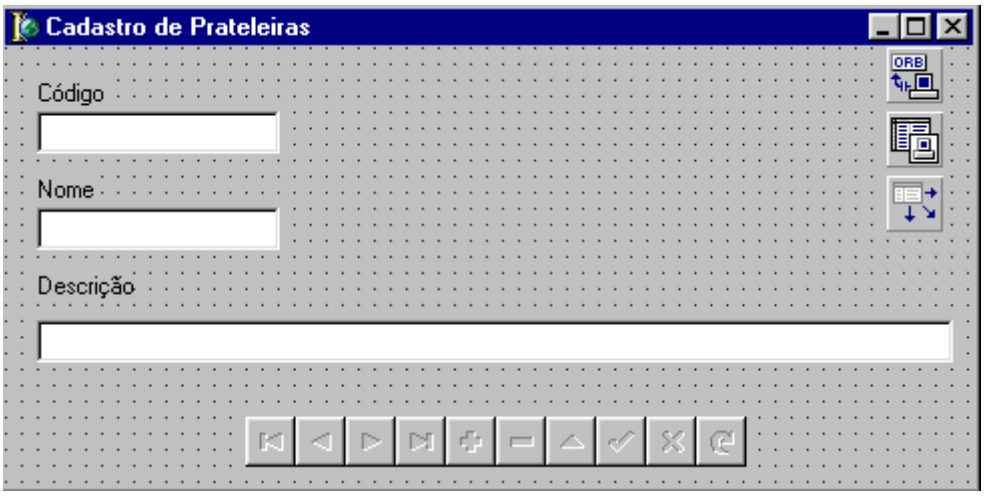

Figura 23 Janela Cadstro de Prateleiras

Na Próxima figura é mostrado o Cadastro de Produtos que possui Código do Produto, Prateleira, Nome da Prateleira, Nome, Nome do Fabricante e descrição dos produtos e os componentes no lado direito ConnectionCorba, DatasetProvider e Datasource que servem para fazer as conexões e ligar as tabelas.

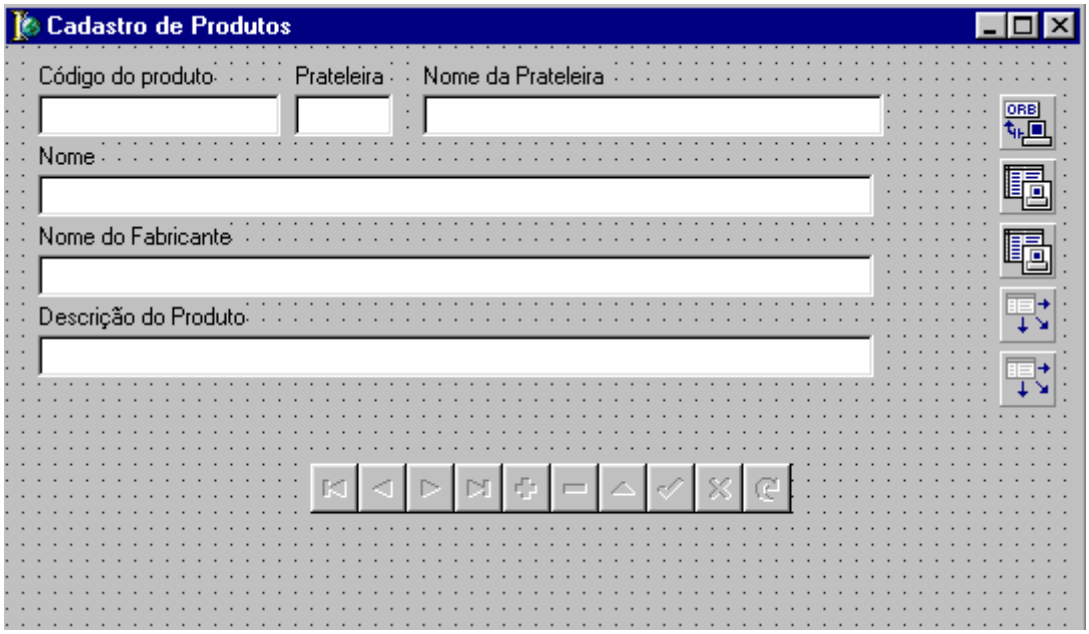

Figura 24 Janela para cadastro de produtos

Na figura Consulta de Produtos por Nome, será possível fazer as consultas dos produtos pelo nome. E os componentes no lado direito ConnectionCorba, DatasetProvider e

Datasource que servem para fazer as conexões e ligar as tabelas.

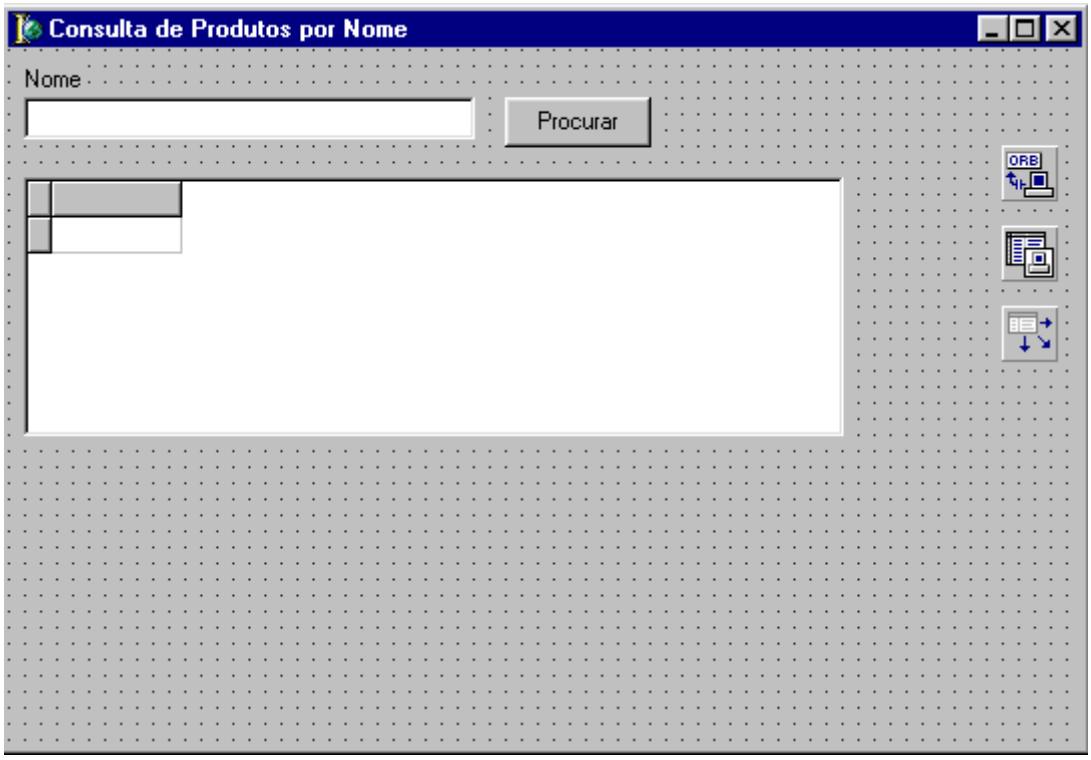

Figura 25 Janela Consultas de Produtos Por Nome

A figura abaixo apresenta a Consulta de Produtos por Prateleiras, nela poderemos fazer as consultas apenas pela prateleira desejada. E os componentes no lado direito ConnectionCorba, DatasetProvider e Datasource que servem para fazer as conexões e ligar as tabelas.

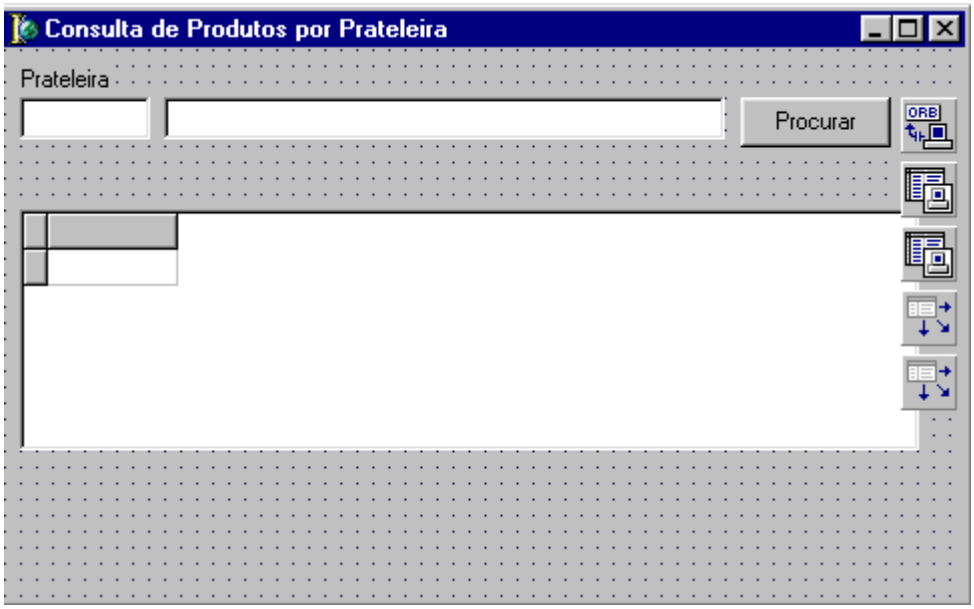

Figura 26 Janela de Consulta de Produtos por Prateleira

Na figura abaixo é mostrado o Cliente, que possui Código, Nome, Endereço, Cidade, Estado, Cep, Telefone, Fax, Email, CGC e os componentes no lado direito ConnectionCorba, DatasetProvider e Datasource que servem para fazer as conexões e ligar as tabelas.

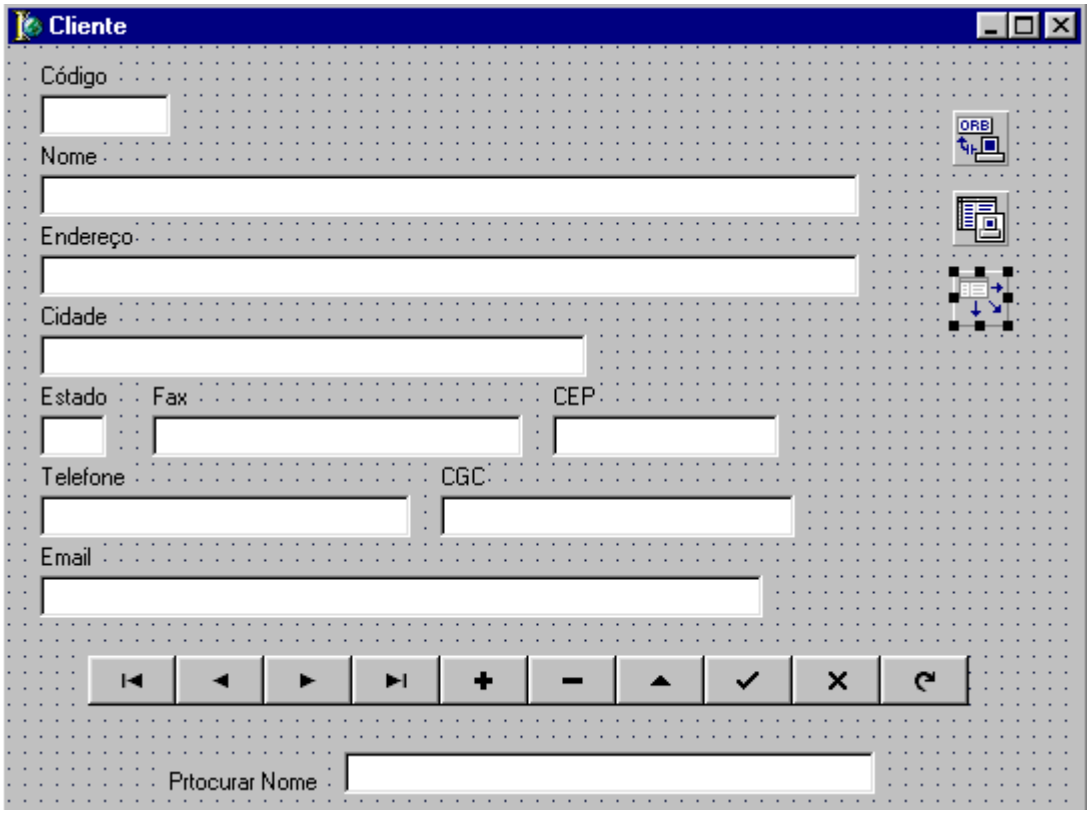

Figura 27 Janela para Cadastro de Fornecedores

4.2 EXEMPLO DE UTILIZAÇÃO

Na Janela baixo iremos mostrar o Servidor rodando

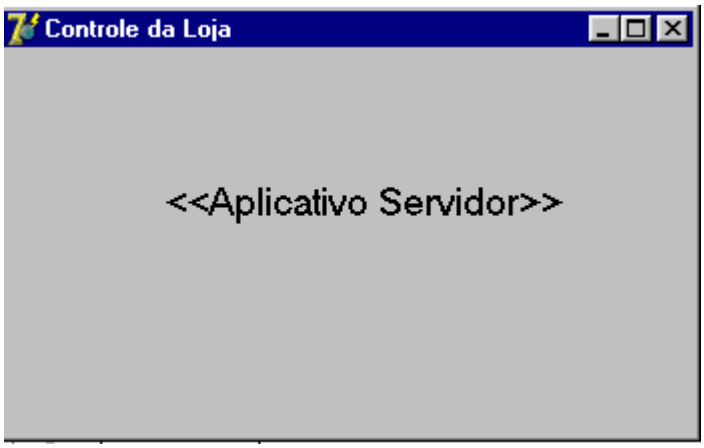

Figura 28 Janela Servidor Rodando

A janela abaixo mostra o controle da loja, onde serão

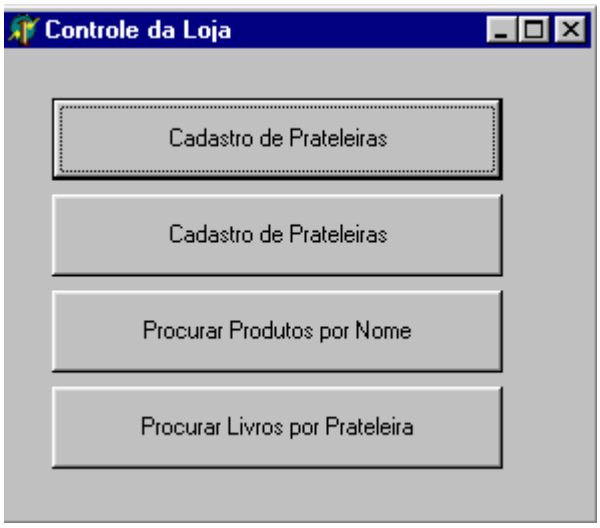

Figura 29 Janela Controle de loja rodando

A janela abaixo mostra o cadastro das prateleiras, que contém o código, o nome e a descrição.

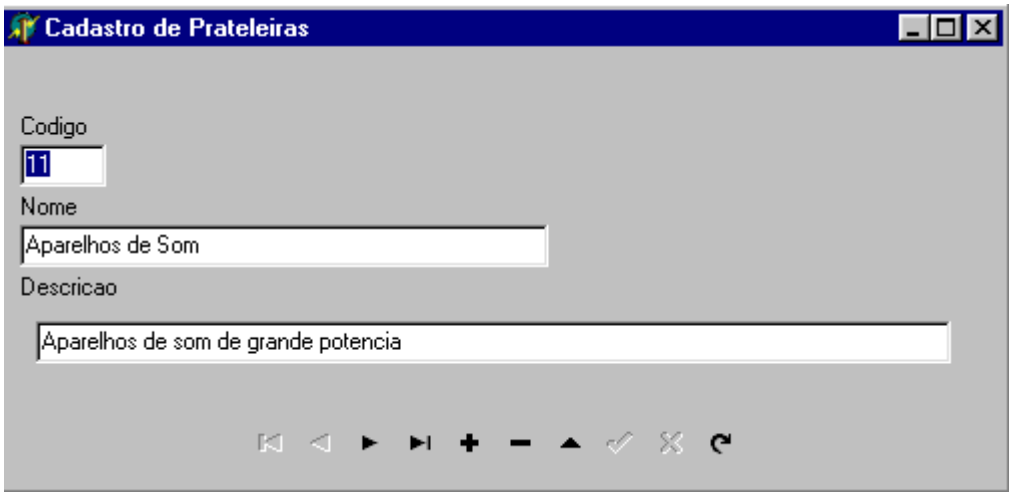

Figura 30 Janela Cadastro de Prateleira Rodando

A janela abaixo mostra a consulta de produtos por nome, que contém os campos prateleira, produto, Título e fabricante.

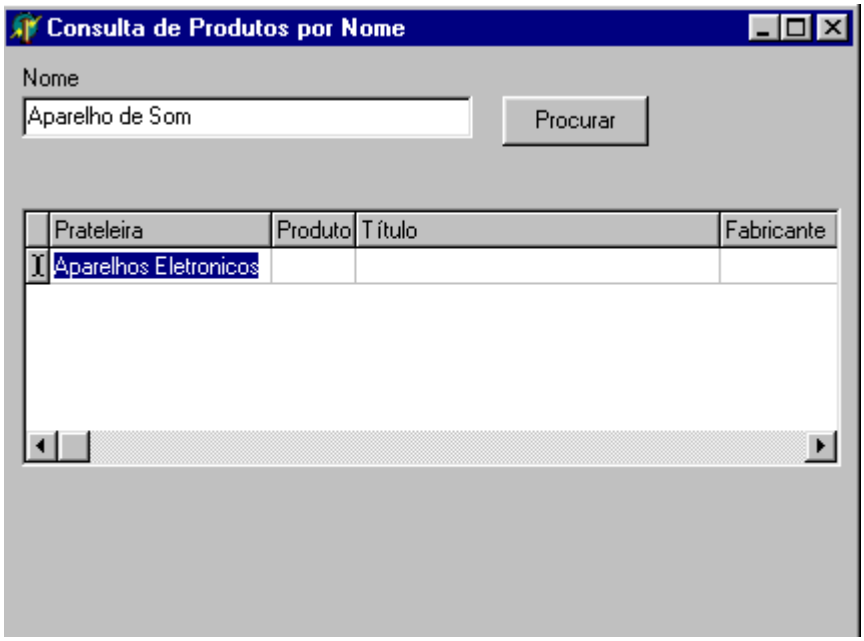

Figura 31 Janela Consulta de produtos por nome rodando

A janela abaixo mostra o cadastro de prateleiras da loja de eletroeletrônicos.

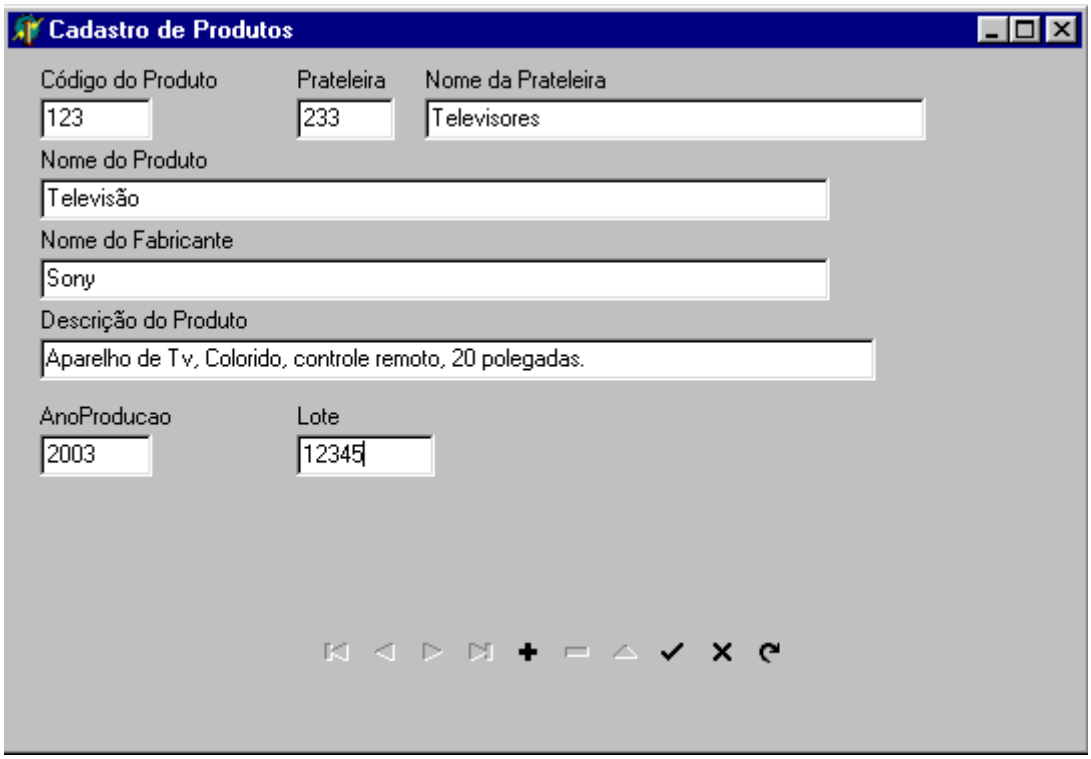

Figura 32 Cadastro de produtos rodando

A janela abaixo mostra a consulta de produtos por prateleiras que possui os campos produto, nome do produto e nome do fabricante.

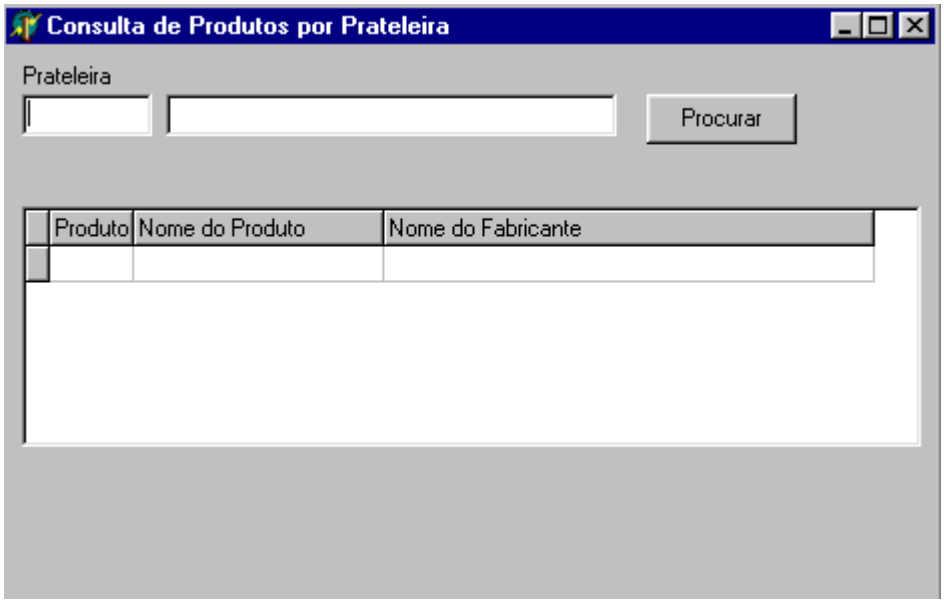

Figura 33 Janela consulta de produtos por prateleira

A janela abaixo serve tanto para cadastrar clientes, quanto para pesquisar

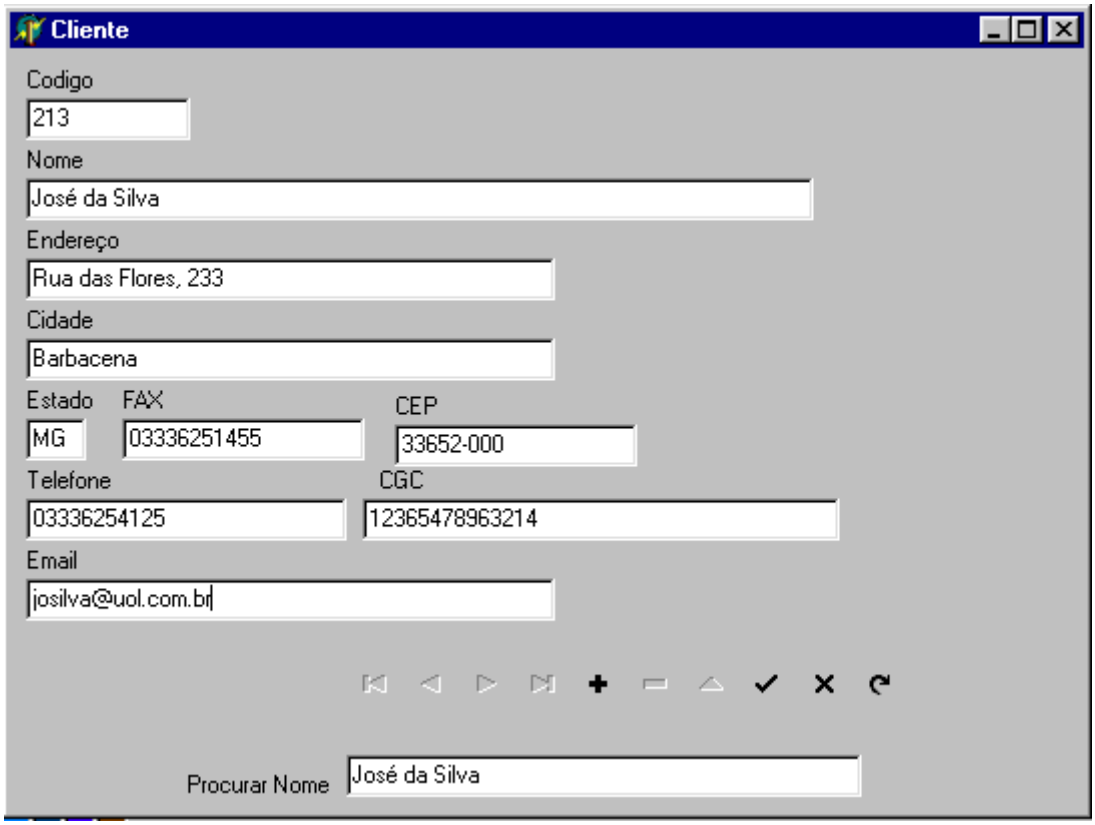

Figura 34 Janela Cliente rodando

5 CONCLUSÃO

Sintetizando todos os aspectos que foram discutidos nesse trabalho de conclusão de curso, no primeiro capítulo foi feito uma introdução com objetivo de se falar sobre o assunto, a proposta que seria apresentada e uma visão geral sobre o que seria falado em cada capítulo.

No segundo capítulo foi apresentado uma explicação e definição do que é Sistemas Distribuídos bem como a definição e apresentação do CORBA (*Common Object Request Broker Architecture*) objeto de estudo desse trabalho.

No terceiro Capítulo trabalhei o desenvolvimento Orientado a Objetos, explicando e exemplificando cada um dos diagramas.

No capítulo 4 foi realizado um estudo de caso, onde foi construído a modelagem do sistema e a Implementação de uma aplicação sobre uma loja de eletroeletrônicos.

Nesse sentido pode-se precisar que sistemas distribuídos tornam-se peças cada vez mais importantes em sistemas de informações, devido, principalmente, às inovações

tecnológicas, como as redes de comunicação de alta velocidade e estações de trabalho com grandes capacidades de processamento.

A heterogeneidade existente de equipamentos, de sistemas operacionais e de autoridades têm levando ao surgimento de especificações abertas, devido à necessidade de padronização desses sistemas. Essa mesma heterogeneidade, também leva à necessidade de fornecer às aplicações uma série de mecanismos de transparências de distribuição.

Sistemas distribuídos costumam ser construídos em modelos cliente/servidor utilizando mecanismos de chamada de procedimento remoto. Recentemente, o paradigma de orientação a objetos, estendeu-se aos sistemas distribuídos, com diversas vantagens sobre o tradicional: melhor reusabilidade, manutenção e depuração, só para citar algumas.

O OMG, um consórcio formado por empresas e instituições de pesquisa, foi criado com o intuito de estabelecer uma padronização de arquitetura para o mundo de objetos distribuídos. Essa padronização, denominada CORBA, é constituída de especificações abertas, que facilitam portabilidade e possuem preocupações com aspectos de interoperabilidade através da padronização de protocolos de comunicação entre seus componentes. A arquitetura CORBA, define um número de serviços indispensáveis às aplicações em geral, variando desde serviços os básicos até os de níveis mais alto.

Este trabalho de conclusão de curso buscou o provimento de informações sobre os Sistemas Distribuídos e da arquitetura CORBA, caracterizando sua arquitetura, definindo seus principais componentes, *interfaces* e principais funções. Portanto esse trabalho pode ser usado no futuro como fonte de pesquisa para o desenvolvimento de novos aplicativos utilizando a arquitetura CORBA.

REFERÊNCIAS BIBLIOGRÁFICAS

[BOOCH 2000] BOOCH, G., RUMBAUGH, J., JACOBSON, I. **UML: guia do usuário**. Rio de Janeiro: Campus, 2000.

[FURLAN 1998] FURLAN, J., D. **MODELAGEM DE OBJETOS ATRAVÉS DA UML : *The Unified Modeling Languagem*.** São Paulo - Makron Books, 1998.

[RICCIONI,2000] RICCIONI, Paulo**. Introducao a objetos distribuidos com CORBA.** Florianopolis: Bookstoore, 2000. 104 p., il. [Artigo de uma revista com autor defini

[GTA-UFRJ,2000] Grupo de Teleinformática e Automação. Acessado em www.gta.ufrj.br/grad/00_2/corba/. Consultado em 02/08/2003.

[GU-CORBA, 2002] Grupo de Usuários CORBA. **O que é CORBA?** São Paulo – 2002. Acessado em www.sucesusp.com.br/html/grupos/corba/corba.htm. Consultado em 20/07/2003.

[MONTEZ, 97] Montez, Carlos. **Um Modelo de Programação para Aplicações de Tempo Real em Sistemas Abertos**, Monografia do Exame de Qualificação de Doutorado, DAS, UFSC, Julho de 1997.

ANEXO A – BANCO DE DADOS

5.1 O BANCO DE DADOS

Primeiramente foi criado um banco de dados a partir da ferramenta Database Desktop do ambiente Delphi para Prateleira, Produtos e Fornecedores:

Prateleira.db

Field Name	Type	Size	Key
Codigo	A	3	*
Nome	A	20	
Descrição	A	40	

Produtos.db

Field Name	Type	Size	Key
CodPrat	A	3	*
CodProd	A	4	*
NomeProduto	A	30	
NomeFabricante	A	30	
DescriçãoProduto	A	40	

Fornecedores.db

Filed Name	Type	Size	Key
Código	A	3	*
Nome	A	20	
Endereço	A	30	
Cidade	A	20	
Estado	A	2	
Cep	A	9	
Telefone	A	15	
Fax	A	15	
E-mail	A	30	
CGC	A	14	