UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE SISTEMA DE CONSULTA DE PREÇOS DE SUPERMERCADOS UTILIZANDO OBJETOS DISTRIBUÍDOS VIA INTERNET

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

ANDERSON LUIZ FERRARI

BLUMENAU, JUNHO/2000

2000/1-05

PROTÓTIPO DE SISTEMA DE CONSULTA DE PREÇOS DE SUPERMERCADOS UTILIZANDO OBJETOS DISTRIBUÍDOS VIA INTERNET

ANDERSON LUIZ FERRARI

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Everaldo Artur Grahl — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Everaldo Artur Grahl

_____

Prof. Maurício Capobianco Lopes

_____

Prof. Marcel Hugo

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

RESUMO

Esse trabalho demonstra o desenvolvimento de um protótipo de sistema de consulta de preços em supermercados que poderá ser acessado através da internet. Primeiramente são demonstradas as principais tecnologias envolvidas como Objetos Distribuídos, *Active Server Pages* (ASP), *Applications Server* e as técnicas de modelagem em multicamadas. Na segunda parte é demonstrada a implementação do protótipo, os modelos de dados e objetos, e a distribuição física dos componentes do sistema. Para o desenvolvimento do modelo de dados foi utilizada a ferramenta *CASE* ER-win, para a modelagem dos objetos em UML foram utilizadas as ferramentas Rational Rose e Visual Modeler e para a implementação do protótipo, o Visual Basic e Visual Interdev.

ABSTRACT

This paper shows development of a price query system prototype at supermarkets who may will be used through the Internet. Firstly are showed the main technologies involved, like Distributed Objects, Active Server Pages (ASP), Application Server and the multitier modelling techniques. On the second part, prototype's development is showed, as well as data and object models, and the physical distribution of system's components. ER-win was used as data modelling tool, Rational Rose and Visual Modeler as UML object modeler, also Visual Basic and Visual Interdev were used to prototype's implementation.

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Com o crescimento da Internet nos últimos anos, começou a surgir uma demanda muito grande por aplicativos que pudessem suportar Comércio Eletrônico na *World Wide Web* (WWW).

Segundo [MIC1999a] "*as empresas se comunicam com os clientes e parceiros através de canais. A Internet é um dos mais novos canais de comunicação comerciais e, sob muitos aspectos, o melhor deles. Ela é rápida, razoavelmente confiável, barata e de acesso universal – atinge praticamente todas as empresas e mais de 100 milhões de consumidores*", e afirma também que o comércio eletrônico se divide em quatro áreas principais conforme [MIC1999a] e [MIC1999c]:

a) marketing direto, vendas e serviços: atualmente essa é a área onde o maior número de *Web Sites* se encontra. Essa modalidade de comércio eletrônico entre empresas e consumidores incrementa o lucro, atingindo principalmente os clientes certos;

b) serviços financeiros e de informações: os *sites* nessa área ajudam as empresas e instituições financeiras a distribuírem suas informações, como transações bancárias ou faturamento, *on-line* através da internet;

c) compras corporativas: é a utilização da internet para efetivar transações entre empresas parceiras;

d) integração da cadeia de valores: são os documentos trocados entre empresas através do protocolo *Eletronic Data Interchange* (EDI). Devido aos altos custos de implantação esse método de comércio eletrônico é utilizado com frequência apenas em grandes empresas.

Existem, sobretudo, vários problemas que devem ser observados na hora de desenvolver e distribuir os sistemas de comércio eletrônico, tais como: custo, retorno, aproveitamento dos sistemas existentes e interoperabilidade.

Juntamente com essa necessidade de desenvolvimento de aplicações para Comércio Eletrônico e Internet que utilizam aplicações distribuídas, surgiram vários novos paradigmas e problemas a serem solucionados.

Para desenvolver o protótipo desse trabalho será utilizada a arquitetura de multicamadas que normalmente faz um uso mais racional dos recursos computacionais disponíveis para aplicações em internet, essa é uma arquitetura baseada em componente, então, para simplificar o processo de desenvolvimento, faz-se necessário o uso de alguma tecnologia de componentes já existe.

Segundo [SES1997], *Component Object Model* (COM) é o modelo de componentes da Microsoft, que compete com o modelo *Commom Object Request Broker Architecture* (CORBA) da *Object Manegement Group* (OMG) para definir objetos em uma rede de computadores, e DCOM (*Distributed* COM) é a tecnologia para distribuir os componentes COM em máquinas remotas. O DCOM é visto como um componente COM. Para todo o efeito, pode-se considerar COM/DCOM como uma simples tecnologia.

Quando se desenvolve aplicações em multicamadas, esses objetos são distribuídos entre as várias camadas que compõem o sistema. [SES1997] afirma que os Sistemas de Comércio baseados em componentes rodam em arquiteturas de 3 camadas. A primeira, a camada do cliente, é responsável por manter os componentes responsáveis pela interface com o usuário. A segunda, a camada de negócios, é responsável por manter os componentes responsáveis pela regra de negócio da aplicação, que por sua vez recebem as instruções da camada de interface do cliente. E a terceira camada é responsável por manter os componentes de acesso a dados, que recebem a requisição dos componentes da camada de regra de negócio.

O ramo de supermercado está aos poucos se adequando à essa nova realidade de comércio e vendas através da internet. Boa parte dos grandes supermercados já possuem o seu site na internet e em muitos deles inclusive , como o do Pão de Açúcar ([PAO2000]), das Lojas Americanas ([LOJ2000]) , da Sé Supermercados ([SES2000]) bem como no supermercado Angeloni ([ANG2000]), é possível fazer pedidos *on-line* através da internet.

Porém não existe até o momento, pelo menos amplamente divulgado e reconhecido, nenhum *site* na internet onde seja possível fazer uma comparação entre os produtos de vários supermercados ao mesmo tempo. Essa proposta de TCC, então, visa desenvolver uma aplicação *Web* que permitirá aos usuários do *site* pesquisar e descobrir quais os estabelecimentos que possuem os produtos consultados pelo menor preço.

A aplicação deverá funcionar da seguinte forma:

a) todos os supermercados que quiserem participar do *site* deverão se comprometer, mantendo os preços dos produtos sempre atualizados;
b) o usuário irá consultar os vários produtos que ele tem interesse e a aplicação irá informar os supermercados que o possuem e seu respectivo preço de venda;
c) o usuário poderá ainda ir selecionando os produtos que ele escolher, dentre os pesquisados, para no final obter algumas estatísticas, como qual supermercado tem o melhor preço para todos os itens selecionados;
d) outros benefícios visando uma melhor interação e navegabilidade, como consultar os produtos por categoria e personalização do *site*, também serão adicionados.

Inicialmente foi escolhido para a implementação, o ramo de supermercados, mas pela característica da implementação, a mesma poderá ser facilmente adaptada para outros ramos ou até mesmo atuar como uma consulta para qualquer tipo de produto de qualquer ramo de atividade.

A modelagem de dados foi desenvolvida com a ferramenta *CASE ER-win*. As ferramentas *CASE Rational Rose* e o *Visual Modeler* foram utilizadas para o desenvolvimento dos casos de uso e da modelagem dos objetos e componentes em multicamadas. O *Microsoft Transaction Server* (MTS) foi utilizado para o gerenciamento de objetos (*Application Server).* Foram utilizadas as ferramentas *Visual Basic* e *InterDev*, para o desenvolvimento dos programas, objetos e componentes, bem como para fazer a interface com o usuário. O SQL Server e o Access foram utilizados como banco de dados para o desenvolvimento, porém a aplicação deverá ser desenvolvida independente de banco e deverá funcionar normalmente com outros bancos relacionais existentes no mercado como o Oracle ou o Sybase.

1.2 ÁREA

O trabalho desenvolvido abrange várias áreas do conhecimento, porém as mais relevantes são as de:

a) engenharia de software: o protótipo será modelado em multicamadas;

b) sistemas e objetos distribuídos: o sistema será orientado à objetos e esses objetos poderão ser executados em outras máquinas, distribuindo assim o processamento, principalmente das regras de negócio.

1.3 PROBLEMA

Quando se necessita fazer qualquer tipo de compra e em especial em supermercados, dois dos principais fatores determinantes para se escolher em qual estabelecimento comprar é, logicamente, saber se eles possuem os itens que se deseja e em qual deles pode-se conseguir comprá-los por um preço menor.

Hoje ainda é muito difícil determinar de uma maneira eficaz, seja através de panfletos, anúncios de TV e jornal ou outros meios de comunicação, qual supermercado possui os itens de que se necessita e qual possui o menor preço.

Como essa aplicação irá ser executada na internet não existe uma estimativa muito certa de quantos usuários irão fazer acesso simultaneamente ao sistema, então, para garantir que ele seja escalável, o protótipo faz uso de objetos distribuídos, que tem a característica de executarem sem problemas em uma ou mais máquinas em qualquer lugar da rede.

Como a aplicação possui muito conteúdo dinâmico, retirado do banco de dados e proveniente de interações do usuário, parte da aplicação utiliza *Active Server Pages* (ASP) para gerar esse conteúdo dinâmico diretamente em um servidor de internet.

1.4 JUSTIFICATIVA

O protótipo faz uso de inúmeras tecnologias que recentemente estão se tornando populares no mercado, como a utilização de objetos distribuídos para sistemas na internet, *Application Servers* e modelagem de objetos em multicamadas, mas que ainda precisam ser amplamente estudados para que se possa usá-las da melhor forma possível.

1.5 OBJETIVO

O objetivo do trabalho é especificar e desenvolver um protótipo em multicamadas de um sistema de consultas de preços para supermercados que utilize objetos distribuídos via internet.

1.6 ORGANIZAÇÃO DO TEXTO

No primeiro capítulo é feita a introdução ao trabalho, demonstrando brevemente as deficiências atuais bem como as necessidades às quais a aplicação se destina a resolver.

No segundo capítulo serão abordados os tópicos relacionados a parte técnica da aplicação, demonstrando as tecnologias e produtos utilizados na confecção do protótipo.

No terceiro capítulo será apresentado o protótipo em si, suas especificações, seu desenvolvimento, seu funcionamento e sua distribuição. Essa capitulo é dividido em três seções distintas, a primeira que demonstra a especificação do sistema, a segunda que demonstra como o mesmo foi implementado e a terceira que faz referência a como o sistema está e como ele pode ser distribuído.

No quarto capítulo serão apresentadas as conclusões finais sobre o trabalho, alguns problemas enfrentados durante o desenvolvimento e algumas sugestões para trabalhos futuros.

2 TECNOLOGIAS E PRODUTOS ENVOLVIDOS

Este TCC envolve muitos termos técnicos e várias tecnologias diferentes. Portanto nesse capítulo, para uma maior compreensão do protótipo implementado, serão demonstradas as tecnologias e produtos relevantes utilizados durante o desenvolvimento do protótipo.

2.1 OBJETOS DISTRIBUÍDOS

A tecnologia de Objetos Distribuídos (OD) permite que uma aplicação seja distribuída em diversos componentes, que podem rodar em diferentes computadores. Esses objetos não parecem estar em computadores diferentes, todos os componentes parecem estar em um grande computador com enorme poder de processamento e capacidade ([MIC1999b]).

Ela vem sendo cada vez mais utilizada e estudada e vem se caracterizando como uma das melhores soluções para desenvolvimento de sistemas para a Internet, conforme afirma [PEN2000]:

"*Acreditamos que a tecnologia de Objetos Distribuídos causará um impacto tão grande no desenvolvimento de software quanto a Internet tem causado nos negócios em geral. Mesmo porque é considerada a melhor forma de desenvolvimento de sistemas para a Internet.*"

Ele coloca, também, que segundo uma estudo do Standish Group avaliando o código de negócios em relação ao código de infra-estrutura, descobriu-se que 30% de todo o código é relativo às regras de negócio e o restante refere-se à infra-estrutura. E ainda que esses 30% de código são normalmente muito mais complexos do que o necessário para os negócios da empresa. E ainda complementa:

"*O Mercado, portanto, precisa de melhores tecnologias, padrões e ferramentas para o desenvolvimento de aplicações cada dia mais complexas. Dentre as soluções que surgiram nas últimas duas décadas para melhorar a situação atual dos projetos de desenvolvimento de software, a Tecnologia de Objetos é a mais recomendada, em especial, a tecnologia de OD.*"

2.2 MULTICAMADAS (*N-TIER*)

Um problema comum encontrado no desenvolvimento de sistemas é o de não haver uma separação lógica e física entre os códigos referentes às regras de negócio de uma aplicação e à infra-estrutura propriamente dita.

A arquitetura em multicamadas pode ser implementada lógica ou fisicamente, ou seja, as duas não precisam necessariamente aparecer juntas, o sistema pode ser implementado logicamente e não distribuído fisicamente [MIC1999b]. Lógica quando o sistema for implementado separando em camadas, que serão vistas ainda nessa seção, durante a especificação. Física, se essas camadas forem, de alguma maneira, separadas em peças de código diferentes e que podem ser distribuídas separadamente, mesmo que rodando em uma mesma máquina.

Essa arquitetura lógica é uma evolução de outros modelos existentes. Primeiramente surgiram as aplicações monolíticas, onde todos os componentes de uma aplicação (interface, regras de negócio e dados) estavam num mesmo lugar, ou seja, em uma única camada, como por exemplo os sistemas desenvolvidos em Clipper, Cobol ou FoxPro sem utilização de banco de dados.

Com o aparecimento das *Local Area Network* (LAN) os *Personal Computers* (PCs) saíram do isolamento e logo começaram a se conectar não só mutuamente, mas também com servidores. Nasce assim a computação Cliente/Servidor [D-T1998].

"*Nos sistemas cliente/servidor de primeira geração, tudo, com exceção do controle de dados, rodava no PC cliente. Existiam dois problemas: desempenho e confiabilidade. Desempenho porque todos os dados, mesmo os pertencentes a uma mesma transação, caminhavam de cá para lá, entre a estação e o servidor. Confiabilidade porque, como as regras de negócio estavam embutidas em cada sistema, nada impedia que dois sistemas tratassem o mesmo dado de duas maneiras diferentes...*" [MEN2000]

A grande maioria dos serviços disponíveis hoje [D-T1998] são utilizados para rodar servidores de arquivo e de banco de dados, sendo que os servidores de aplicação são a exceção. Os servidores de banco de dados oferecem apenas dados, consequentemente a inteligência da aplicação precisa estar implementada no cliente. Sendo assim, a arquitetura

desse modelo está claramente dividida em duas camadas, um servidor de dados e um cliente. Esse modelo é chamado de arquitetura 2-*tier* ou cliente/servidor e atualmente ainda é o predominante.

Surge então uma nova abordagem que propõe que todas as regras de negócio, que possuem a maior parte da inteligência de um sistema, sejam separadas do sistema e isoladas em uma única camada específica para isso. Surge então a arquitetura em 3-camadas, que separa logicamente todo o código da aplicação em três camadas (que serão vistas detalhadamente mais adiante):

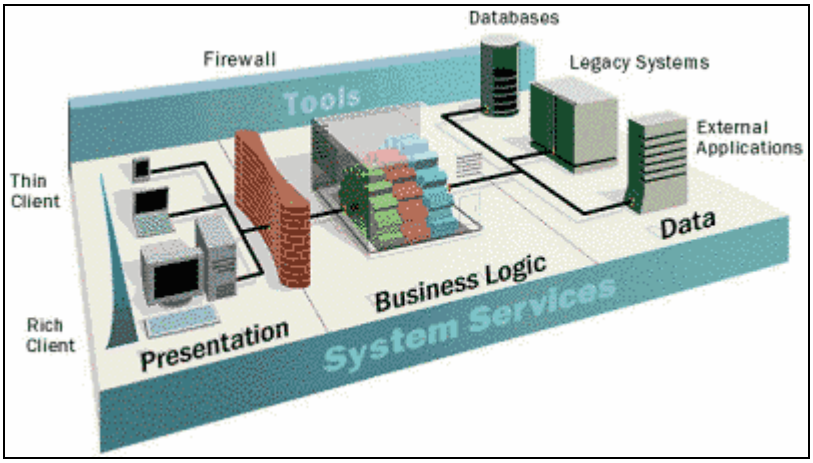
a) **interface;**

b) **regras de negócio;**

c) **acesso a dados.**

Comumente aparecem algumas variações na nomenclatura de cada uma das 3 camadas, mas que não interferem no compreendimento da arquitetura bem como no do protótipo aqui apresentado.

Figura 1 – Modelo da arquitetura em 3 camadas



Fonte: [MIC1999d]

A figura 1 demonstra a divisão lógica da arquitetura em 3 camadas. Na camada de interface ou apresentação pode-se ter desde clientes “magros”, com menor capacidade de processamento, até clientes “gordos”, com maior processamento. Na camada de regras de

negócio tem-se o conhecimento do sistema, na figura representado por uma engrenagem. E na camada de dados a figura apresenta algumas das possibilidades de persistência dos dados, como por exemplo os bancos de dados.

Algumas das principais vantagens de se implementar o modelo físico em três camadas em relação aos outros modelos, segundo [MIC1999b] são:

a) independência de banco de dados;

b) algumas variações de implementação oferecem independência de linguagem. Por exemplo, utilizando-se um servidor de aplicação pode-se executar objetos, que sejam suportados por ele, escritos em qualquer linguagem;

c) em alguns casos ela é mais escalável do que outras arquiteturas, pois permite que várias partes do código sejam executadas distribuídas em locais diferentes.

Algumas outras vantagens adicionais da arquitetura em três camadas que podem ser ressaltadas, segundo [MEN2000] são:

a) como boa parte do processamento é deslocada do cliente para o servidor, o *upgrade* de uma única máquina, o servidor de aplicação, terá um impacto significativo no desempenho do sistema como um todo;

b) as regras de negócio podem ser armazenadas em um único lugar, o próprio servidor de aplicação, independente do tipo de banco de dados envolvido, facilitando a manutenção e aumentando a garantia de que nenhuma regra de negócio será desobedecida.

Em contrapartida como desvantagens, ainda segundo [MIC1999b], a implementação do modelo físico de três camadas em relação a outros modelos:

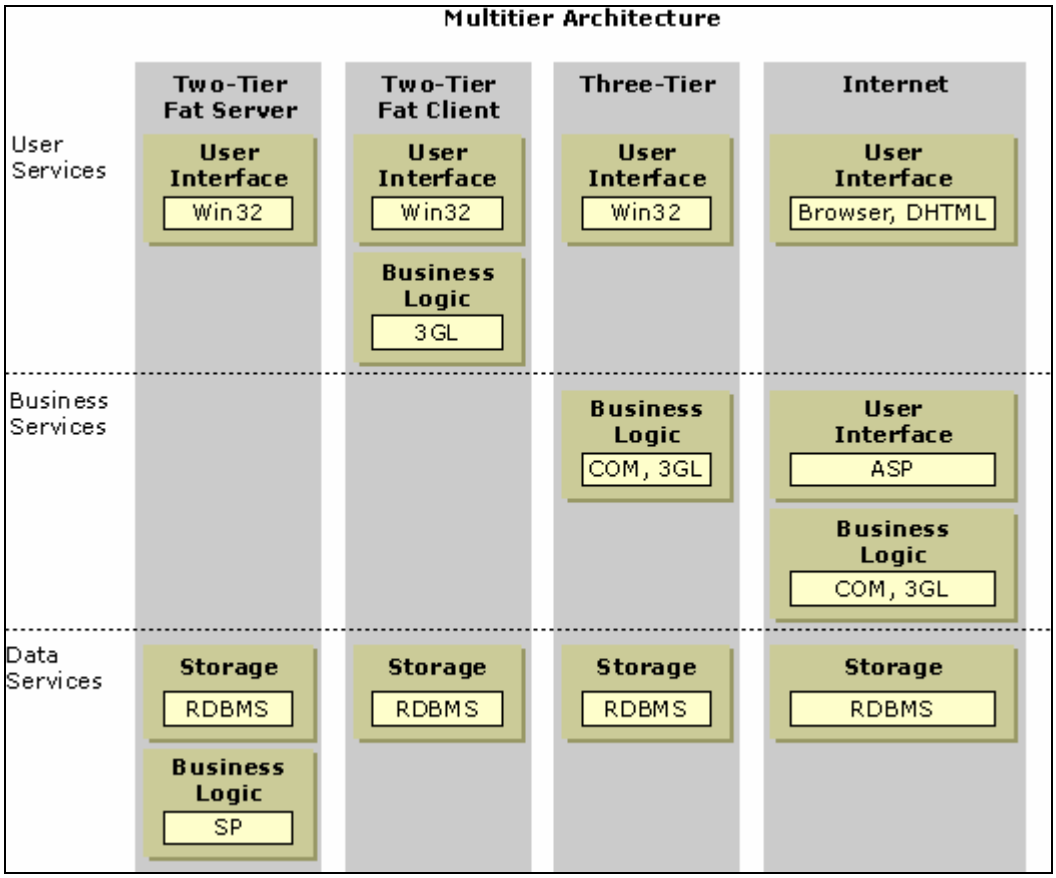
a) tende a requerer mais administração;

b) sua relação preço/performance geralmente é muito pior do que aplicações que implementam parte da lógica em *Stored Procedures*, que são procedimentos que ficam armazenados e rodam no servidor de banco de dados.

Uma pequena variação da arquitetura em 3-camadas é a arquitetura de internet. A principal diferença é que ela processa também boa parte da interface na camada de negócios, conforme demonstrado na figura 2.

A figura 2 demonstra ainda a divisão da lógica de implementação entre as camadas dos sistema em 2-camadas com servidor *fat*, ou seja, que possuem uma boa capacidade de processamento no servidor, e cliente *fat*, onde a maior capacidade de processamento está no cliente. Demonstra também a arquitetura em 3-camadas e a arquitetura de Internet em multicamadas.

Figura 2 –Diferenças entre as arquiteturas em multicamadas



Fonte: [MIC1999b]

Para o desenvolvimento do protótipo serão utilizadas as arquiteturas em três camadas para as aplicações desenvolvidas com o Visual Basic e a arquitetura de Internet para os módulos do sistema que irão estar disponíveis na *Web*.

2.2.1 SERVIÇOS

Um serviço ou camada ([MIC1996]) é uma unidade lógica da aplicação que implementa uma operação, função ou transformação que é aplicada a um objeto. Serviços podem garantir a integridade das regras do negócio, executar cálculos ou manipulações nos dados e definir características para entrar, recuperar, visualizar ou modificar informação.

Serão apresentadas as três principais camadas lógicas da arquitetura multicamadas utilizadas para desenvolver a aplicação.

2.2.1.1 CAMADA DE INTERFACE (*USER SERVICES* OU *PRESENTATION*)

É a unidade lógica da aplicação que provê a interface da aplicação e permite que sejam feitas iterações pelo usuário. O usuário de uma aplicação pode ser uma pessoa ou outra aplicação, portanto a interface de uma aplicação pode ser uma interface gráfica de usuário ou uma interface programada [MIC1996].

2.2.1.2 CAMADA DE REGRAS DE NEGÓCIO (*BUSINESS RULES*, *BUSINES LOGIC* OU *BUSINESS SERVICES*)

Essa é a unidade lógica da aplicação que controla a sequência, execução das regras de negócio e a integridade transacional das operações que ele executa. Os objetos dessa camada transformam os dados em informação através da aplicação das regras de negócio apropriadas [MIC1996].

2.2.1.3 CAMADA DE DADOS (*DATA* OU *DATA SERVICES*)

Essa unidade lógica da aplicação provê o nível mais baixo visível de abstração usado para a manipulação dos dados. Os serviços dessa camada mantém a disponibilidade e integridade dos dados persistentes e não-persistentes como um recurso da empresa. Eles provêm os serviços de Criação, Leitura, Atualização e Exclusão os quais os serviços de negócio (consumidores dos serviços dessa camada) não precisam conhecer onde os dados se localizam, como eles estão implementados, ou como eles são acessados [MIC1996].

2.3 SERVIDOR DE APLICAÇÃO (*APPLICATION SERVER*)

"*Um servidor de aplicação é definido pelo mercado como qualquer coisa que fique entre o cliente e o servidor de dados, por isso há uma série de tecnologias muito diferentes que recebem esse nome.*" [MEN2000].

O *Application Server* que é utilizado nesse trabalho é na verdade, e algumas vezes assim chamado, um servidor de componentes atuando como um *middleware* para a aplicação. Ele mantém e gerencia os componentes que são utilizados pelas aplicações.

Esse tipo de servidor torna mais fácil de distribuir e escalar as aplicações distribuídas que rodam em servidores [MIC1999b]. Os componentes são colocados nele e referenciados diretamente na aplicação.

Normalmente esses servidores implementam outros tipos de tarefa como gerenciar transações entre um ou diversos bancos de dados, segurança no acesso aos componentes e gerenciamento de *threads* entre outros.

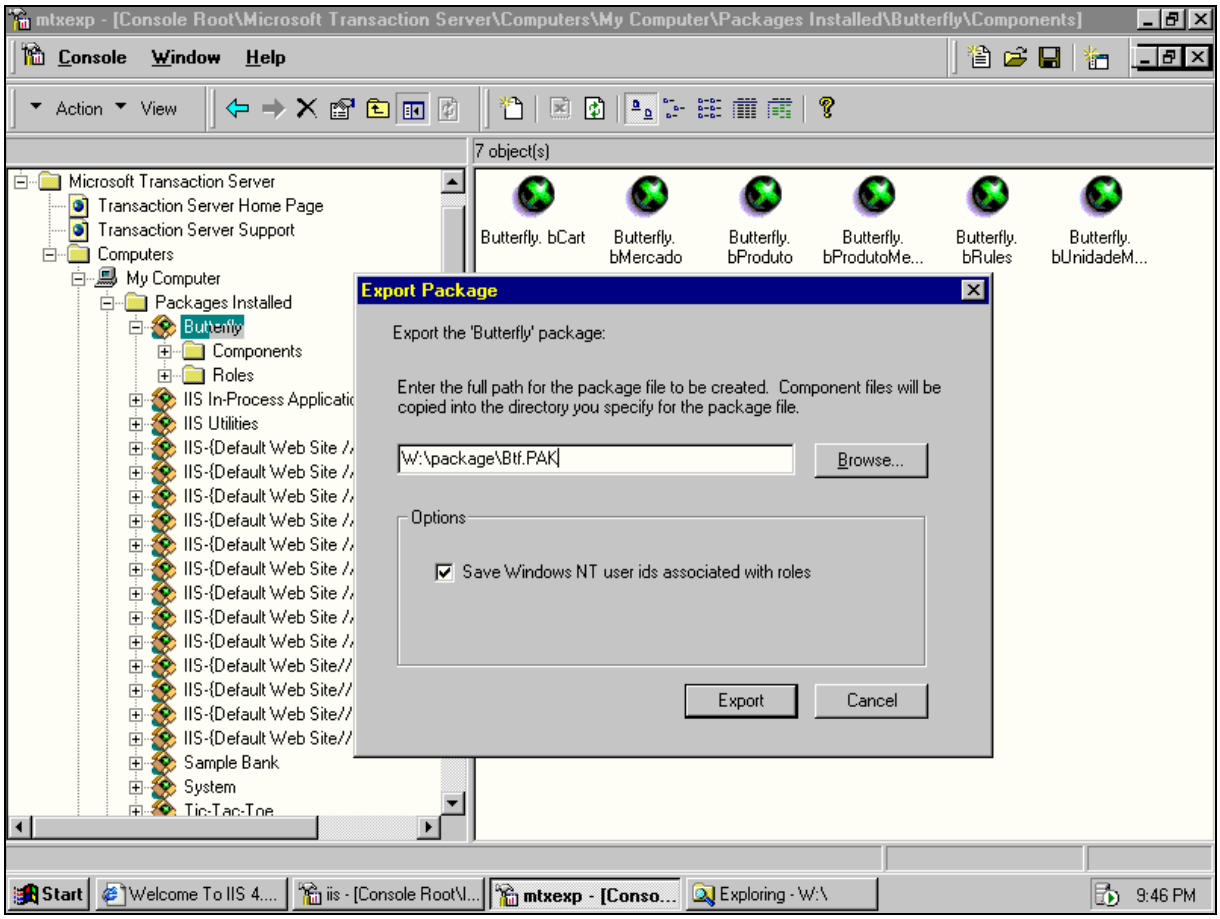
No mercado existem *Application Servers* de vários fabricantes, como o Microsoft Transaction Server (MTS) da Microsoft, o Jaguar da Powersoft/Sybase e o Voyager da ObjectSpace.

2.3.1 MICROSOFT TRANSACTION SERVER (MTS)

Por questões de facilidade de uso e praticidade e pelo fato de já vir incluído com qualquer produto da família de servidores Windows NT, foi escolhido o MTS para gerenciar os componentes que serão utilizados no sistema.

O MTS gerencia componentes COM que são adicionados nele dentro de pacotes. Esses pacotes podem ser exportados para serem utilizados nas aplicações. Esses pacotes exportados geram aplicativos executáveis que configuram o cliente DCOM para acessar o MTS. A figura 3 mostra a interface de exportação de um desses pacotes. O MTS gera um arquivo que deve ser executado nas máquinas cliente, para que as mesmas consigam encontrar os ciomponentes distribuídos.

Figura 3 – Exportação de pacotes do MTS



Nativamente com o Windows 2000 é distribuído o COM+, que é na verdade uma evolução do MTS. Ele funciona de uma maneira muito parecida e os componentes do MTS são completamente compatíveis com essa nova tecnologia.

2.4 *ACTIVE SERVER PAGES* (ASP)

Active Server Pages (ASP) é um ambiente de programação que possibilita combinar HTML, *scripting* e componentes para criar poderosas aplicações Internet que rodem em seu servidor. Usando ASP pode-se criar uma interface HTML para as aplicações, adicionando comandos nos *scripts* em páginas HTML e pode-se encapsular a sua lógica de negócio em componentes reusáveis. Esses componentes podem ser chamados através de *scripts* ou por outros componentes [MIC1999b].

Na prática os códigos ASP sempre são executados no servidor e enviados para o cliente que o requisitou, normalmente em formato HTML. Eles mesclam *scripts* com HTML.

O quadro 1 mostra um trecho de código que codifica uma aplicação exemplo do tipo “Hello World!!!”.

Quadro 1 – Exemplo “Hello World!” em ASP

```
<P>
<%If Time > = #12:00:00 AM# And Time < #12:00:00 PM# Then%>
     Bom Dia!
<%Else%>
     <FONT COLOR=green> Hello World!!! </FONT>
<%End If%>
</P>
```

O Quadro 2 demonstra um outro grande recurso do ASP que é o de criar objetos no servidor.

Quadro 2 – Criação de objetos em ASP

```
Dim oDGA
Set oDGA = server.CreateObject(“Butterfly.dDGA”)
```

2.4.1 VISUAL INTERDEV

Códigos em ASP são arquivos texto normais e podem ser escritos utilizando-se qualquer editor de textos, porém várias ferramentas integradas atuais trazem suporte e vários recursos para auxiliar no desenvolvimento com essa tecnologia. Uma delas, que vem junto com o pacote *Visual Studio* da Microsoft, e que foi utilizada para auxiliar no processo de desenvolvimento desse trabalho, é o *Visual Interdev*. Ela reúne várias facilidades para os desenvolvedores ASP, como:

a) editor integrado de código;

b) editor gráfico;

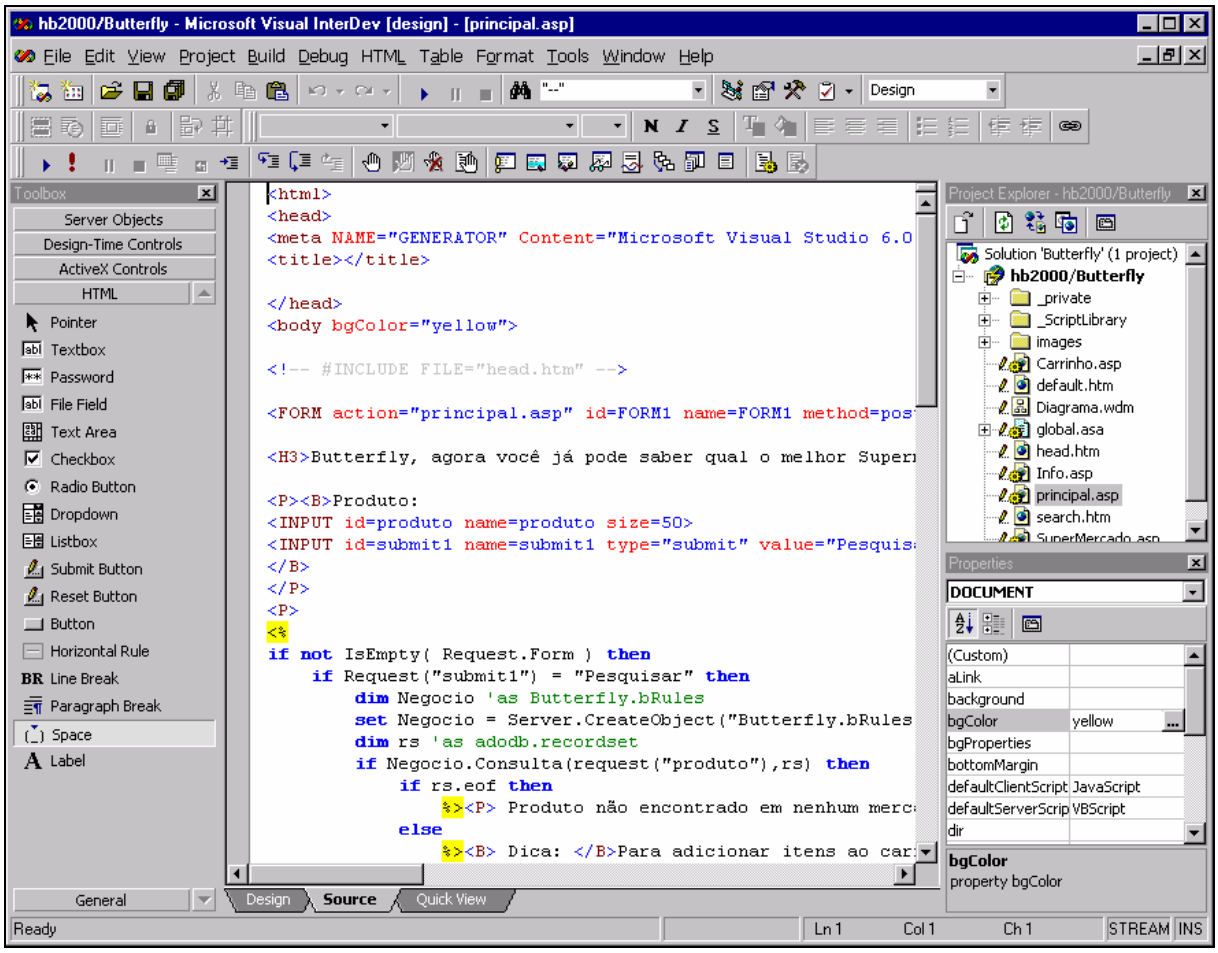
c) ferramentas de *debug*;

d) agrega uma série de componentes utilizáveis;

e) documentação integrada desde linguagens *script* até utilização de componentes e objetos padrão.

A figura 4 mostra o ambiente de desenvolvimento do Visual Interdev destacando a janela do editor de código.

Figura 4 – Editor de código do Visual Interdev



2.5 VBSCRIPT E JAVASCRIPT

Diferente de linguagens de programação mais complexas, as linguagens *script* são interpretadas. Elas são fáceis de aprender e, embora não sejam tão abrangentes quanto as outras linguagens, elas possuem funcionalidade bastante poderosa [MIC1999b].

As linguagem de *script* são utilizadas tanto na parte da aplicação que roda no servidor quanto na parte que roda no cliente. Dependendo da capacidade do cliente e do servidor podem ser utilizados diversos tipos de *script*. O Visual Interdev e o servidor de internet *Internet Information Server* dão suporte tanto a *VBscript* quanto a *JAVAScript*. O

desenvolvedor está livre para escolher o que ele mais se identifica e o que for mais compatível com sua aplicação.

Para o desenvolvimento desse trabalho foi utilizado o VBscript juntamente com os códigos em ASP que rodam no servidor. Para *scripts* que rodam em *Web browsers* no cliente é muito comum a utilização do JAVAscript por ser suportado pela maioria dos *browsers* do mercado, diferente do VBscript que em alguns casos necessita de *plug-in's* para poder ser executado.

2.6 OUTRAS TECNOLOGIAS

Além das tecnologias e produtos utilizados e demonstrados anteriormente, foram também utilizados alguns outros não tão relevantes para o projeto e que também não precisaram ser profundamente estudos para a elaboração do protótipo, são eles:

a) programação orientada a objetos (OOP);
b) ferramenta de desenvolvimento Desktop e de componentes Visual Basic;
c) ferramenta de desenvolvimento para Internet Visual Interdev;
d) linguagem SQL (*Structured Query Language*) ;
e) banco de dados Sql Server 7.0;
f) banco de dados Access;
g) sistemas operacional Windows NT Server 4.01;
h) modelo de componentes distribuídos DCOM;
i) softwares para desenvolvimento das especificações, conforme notação *Unified Modeling Language* (UML), Visual Modeler e Rational Rose;
j) software para modelagem de dados ERwin;
k) servidor de internet *Internet Information Server* 4 (IIS);
l) HTML;
m) ADO (ActiveX Data Objects): maiores referências podem ser encontradas em [MIC1999b] e [MIC1999f].

3 DESENVOLVIMENTO DO TRABALHO

Nesse capítulo serão demonstradas as especificações do sistema, as aplicações desenvolvidas, como foram utilizadas as tecnologias descritas no capítulo 2 e todos os passos para a elaboração do protótipo.

Por definição, esse projeto do protótipo de sistema foi chamado de "**Butterfly**" e poderá ser referenciado com esse nome daqui para frente.

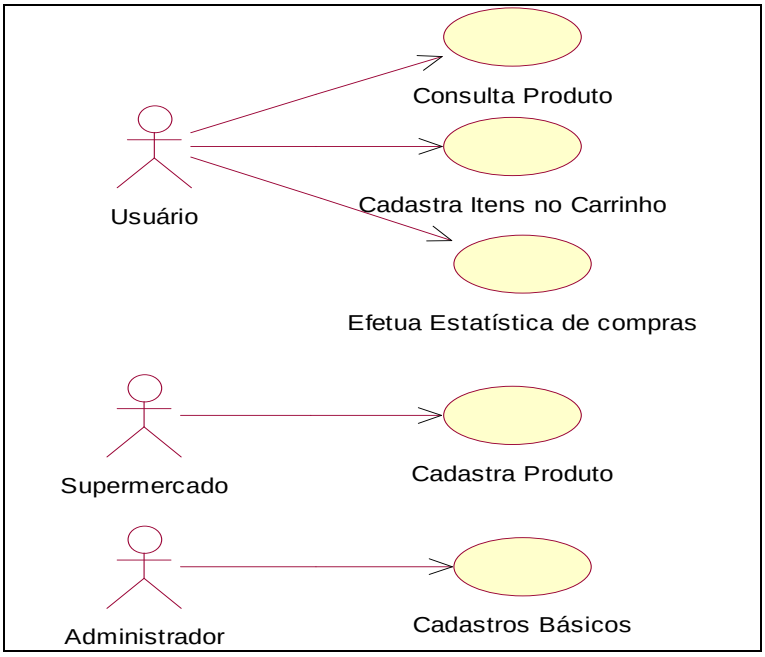
3.1 ESPECIFICAÇÃO

Serão demonstrados nessa seção os modelos desenvolvidos durante a fase de análise do software.

3.1.1 DIAGRAMA DE CASOS DE USO

Conforme a notação *Unified Modeling Language* (UML) ([UML1999]) e utilizando-se a ferramenta Rational Rose foi desenvolvido o diagrama de casos de uso conforme a Figura 5, onde destacaram-se três atores e cinco casos de uso principais do sistema.

Figura 5 - Diagrama de casos de uso do sistema Butterfly conforme notação UML



Três atores foram inicialmente identificados e são definidos na tabela 1.

Tabela 1 – Atores do sistema

Usuário	É o ator primário para o qual o sistema foi desenvolvido. Ele é o cliente efetivamente que irá utilizar e tirar benefícios do sistema. Ele fará uso das consultas e estatísticas.
Supermercado	É um ator secundário que irá alimentar e manter os produtos e seus atributos atualizados no sistema.
Administrador	É um ator secundário que tem por função administrar todo o sistema e efetuar os cadastros básicos para o funcionamento do mesmo.

Cinco casos de uso foram definidos para o sistema e são definidos na tabela 2.

Tabela 2 – Casos de uso do sistema

Consulta Produto	O Usuário efetua uma consulta para procurar por um determinado produto em todos os supermercados, a fim de descobrir qual tem o menor preço e o sistema retorna a ele todas as possibilidades de compra.
Cadastra Itens no Carrinho	O usuário adiciona itens ou exclui itens previamente colocados no carrinho.
Efetua Estatística de Compras	O usuário solicita relatórios estatísticos como, por exemplo, qual supermercado possui os itens selecionados pelo menor preço.
Cadastra Produto	O Supermercado inclui e dá manutenção sobre os produtos que ele disponibiliza.
Cadastros Básicos	O administrador cadastra as informações dos supermercados conveniados, alimenta a tabela de unidade de medida e cadastra a tabela de produtos, tudo isso para a operacionalização do sistema.

3.1.2 DIAGRAMA DE CLASSES

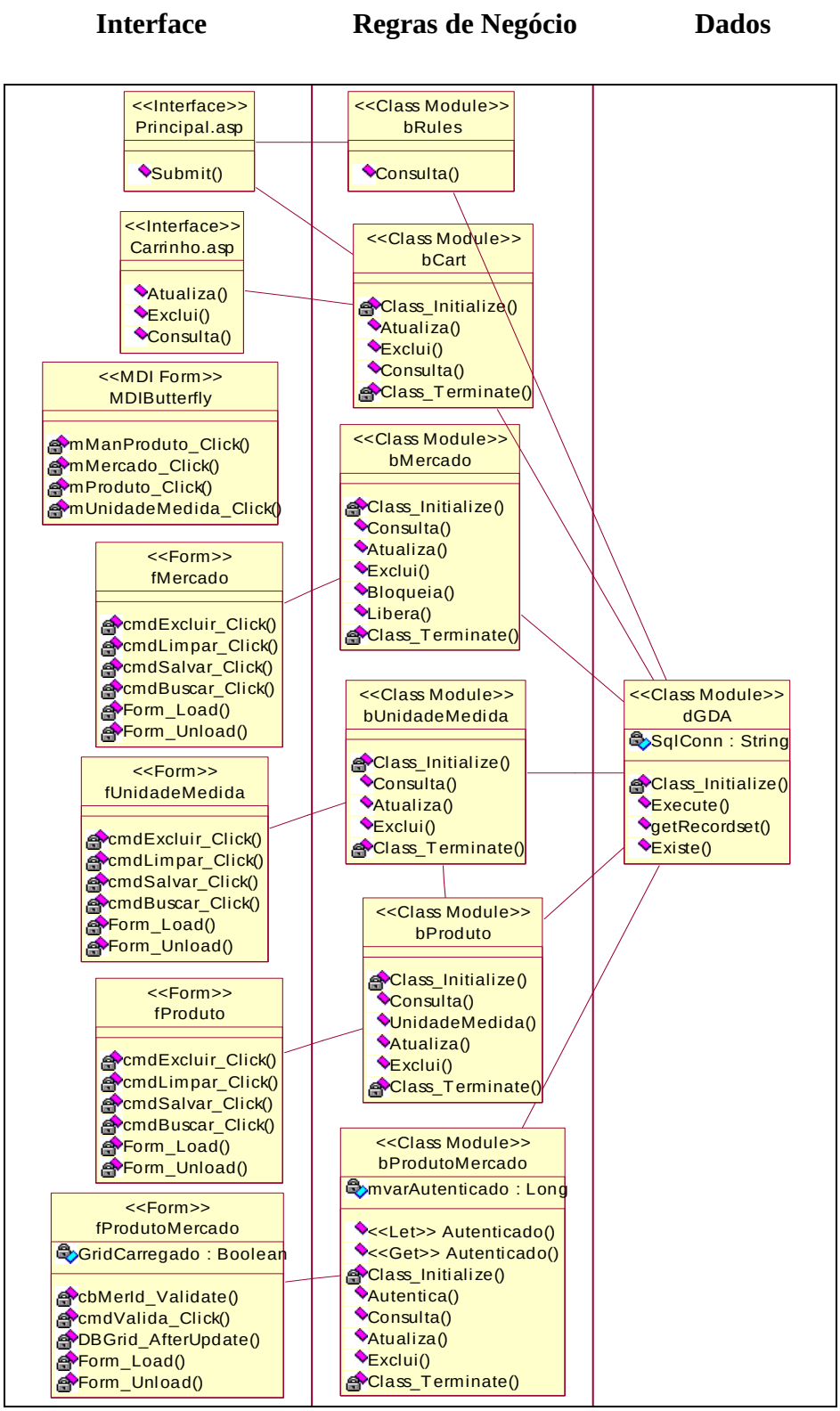
Foi convencionado para a nomenclatura das classes desse projeto que as classes que se iniciam pela letra “d” referem-se aos objetos da camada de dados e “b” (de *Business*) para as classes dos objetos da camada de regras de negócio.

É no diagrama de classes que se faz claramente visível a separação entre as 3 camadas lógicas de um sistema multicamadas, podem-se perceber na figura 6 que os formulários que interagem com os usuário do sistema estão na camada de Interface, as classes que possuem o conhecimento do sistema estão na camadas de Regras de Negócio e a classe que faz a persistência dos dados no banco está na camada de Dados.

Inicialmente foram identificados alguns dos objetos que interagem diretamente com o sistema no mundo real, como o supermercado, produto e usuário, que depois acabaram se transformando em objetos da camada de negócio da aplicação conforme demonstrado na figura 6.

O diagrama de classes exposto na figura 6 demonstra como os objetos interagem entre si em suas diversas camadas. Pode-se perceber que os objetos localizados na camada de interface fazem chamadas aos objetos da camada de regra de negócios e esses objetos que detém o conhecimento de como funciona cada processo específico da aplicação faz uso do objeto, nesse caso genérico, de acesso à dados. Nesse protótipo optou-se pela implementação de um único objeto na camada de dados para gerenciar a chamada dos objetos de negócio. Outras abordagens de modelos multicamadas sugerem que sejam implementados objetos específicos para cada, ou um conjunto, de regras de negócio.

Figura 6 – Diagrama de classes em 3 camadas (projeto físico)



As tabelas que explicam o modelo de dados foram exportadas através das ferramentas do *case ER-win* e importadas nesse documento.

Na tabela 3 são listadas as classes responsáveis pelos serviços de interface com o usuário e suas funcionalidades.

Tabela 3 – Classes da camada de Interface

Principal.asp	Formulário de interação com o usuário. Essa tela está rodando na internet e é responsável pela consulta de produtos realizada pelos usuários do sistema e também por adicionar os itens no carrinho de compra.
Carrinho.asp	Formulário de interação com o usuário. Essa tela está rodando na internet e é responsável pela manutenção dos itens adicionados no carrinho de compras.
MDIButterfly	Janela principal do aplicativo desenvolvido em Visual Basic responsável pela interação do sistema com o Administrador e com o Supermercado. Essa tela, como trata-se de uma MDI, é responsável por abrigar as outras janelas da aplicação.
Fmercado	Janela do aplicativo em Visual Basic responsável pelo cadastro e manutenção dos Supermercados.
FunidadeMedida	Janela do aplicativo em Visual Basic responsável pelo cadastro e manutenção das unidades de medida.
Fproduto	Janela do aplicativo em Visual Basic responsável pelo cadastro e manutenção dos Produtos.
FProdutoUsuario	Janela do aplicativo em Visual Basic responsável pela manutenção dos preços e quantidades em estoque de cada produto de cada supermercado.

Na tabela 4 são listadas as classes responsáveis pelos serviços da camada de Regras de Negócio e sua funcionalidade.

Tabela 4 – Classes da camada de Regras de Negócio

bRules	Essa é a principal classe de regras de negócio do sistema. Nela estão contidas as regras de negócio da consulta de produtos efetuada pelo usuário.
bCart	Essa é a classe responsável por armazenar e manter os itens selecionados pelo usuário no carrinho de compras. Ela possui uma série de métodos destinados a isso.
bMercado	Essa classe é responsável por manter as informações da tabela de Mercado do banco de dados e possui uma série de métodos para tal fim.
bUnidadeMedida	Classe responsável pelas regras de negócio correspondentes a tabela de Unidade de Medida.
bProduto	Classe responsável pelas regras de negócio correspondentes a tabela de Produto.
bProdutoMercado	Classe responsável pelas regras de negócio correspondentes a tabela de Produto por Supermercado. Essa classe mantém o preço e a quantidade de produtos em estoque de cada supermercado.

Na Tabela 5, é listada classe responsável pelos serviços da camada de acesso a dados.

Tabela 5 – Classe da camada de acesso a dados

DGDA	Nesse modelo essa é a única classe na camada de acesso a dados. Ela é uma classe genérica que é utilizada para tornar os dados persistentes, ou seja, gravar as informações no banco de dados. Ela é responsável pela conexão com o banco e pode ser utilizada para conectar com outros bancos que suportem SQL ANSI simplesmente alterando-se a "String de Conexão" mantida no atributo "SqlConn".

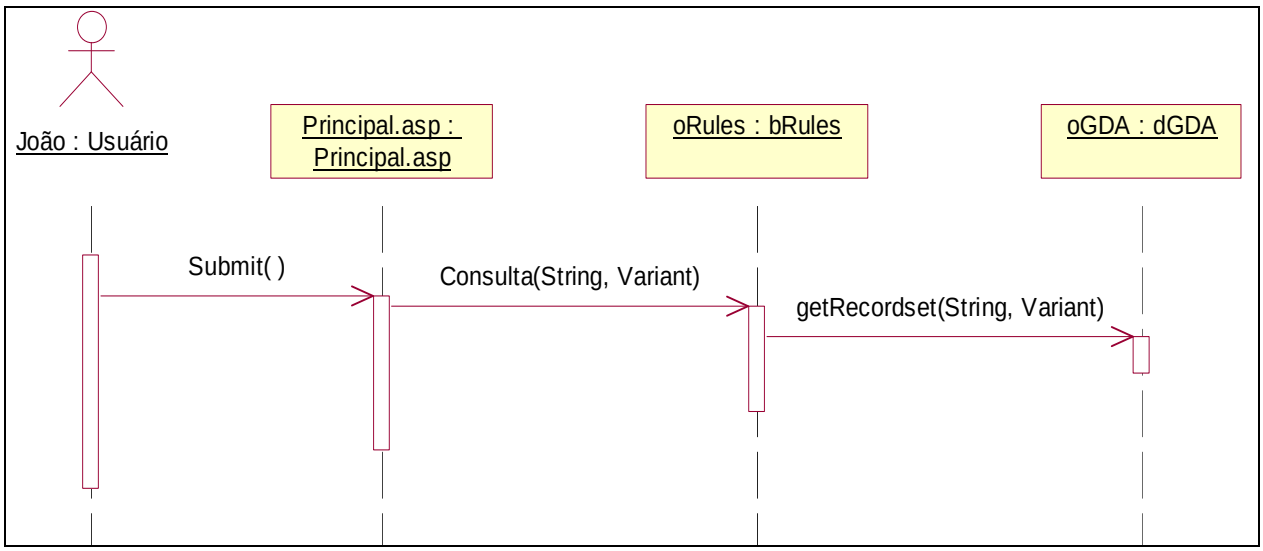
3.1.3 DIAGRAMA DE SEQUÊNCIA

Nos diagramas de sequência pode-se observar o tempo de execução de cada objeto e verificar as mensagens que são trocadas entre os objetos.

Os diagramas de sequências com base nos casos de uso do sistema são:

a) consulta dos produtos pelo usuário via Internet (figura 7). Quando o usuário efetua uma consulta no sistema, através do ButterflyQuery, o sistema dispara o método ***Submit*** da tela **Principal.asp** enviando os dados da consulta. Essa tela por sua vez chama o método **Consulta** do objeto **oRules** para que ele efetue a consulta propriamente dita, quando o mesmo dispara o método **getRecordset** do objeto **oGDA** responsável por buscar as informações no banco de dados.

Figura 7 – Diagrama de Sequência: consulta de produtos pelo usuário



b) manutenção do carrinho de compras pelo usuário (figura 8). O usuário pode solicitar três comandos:

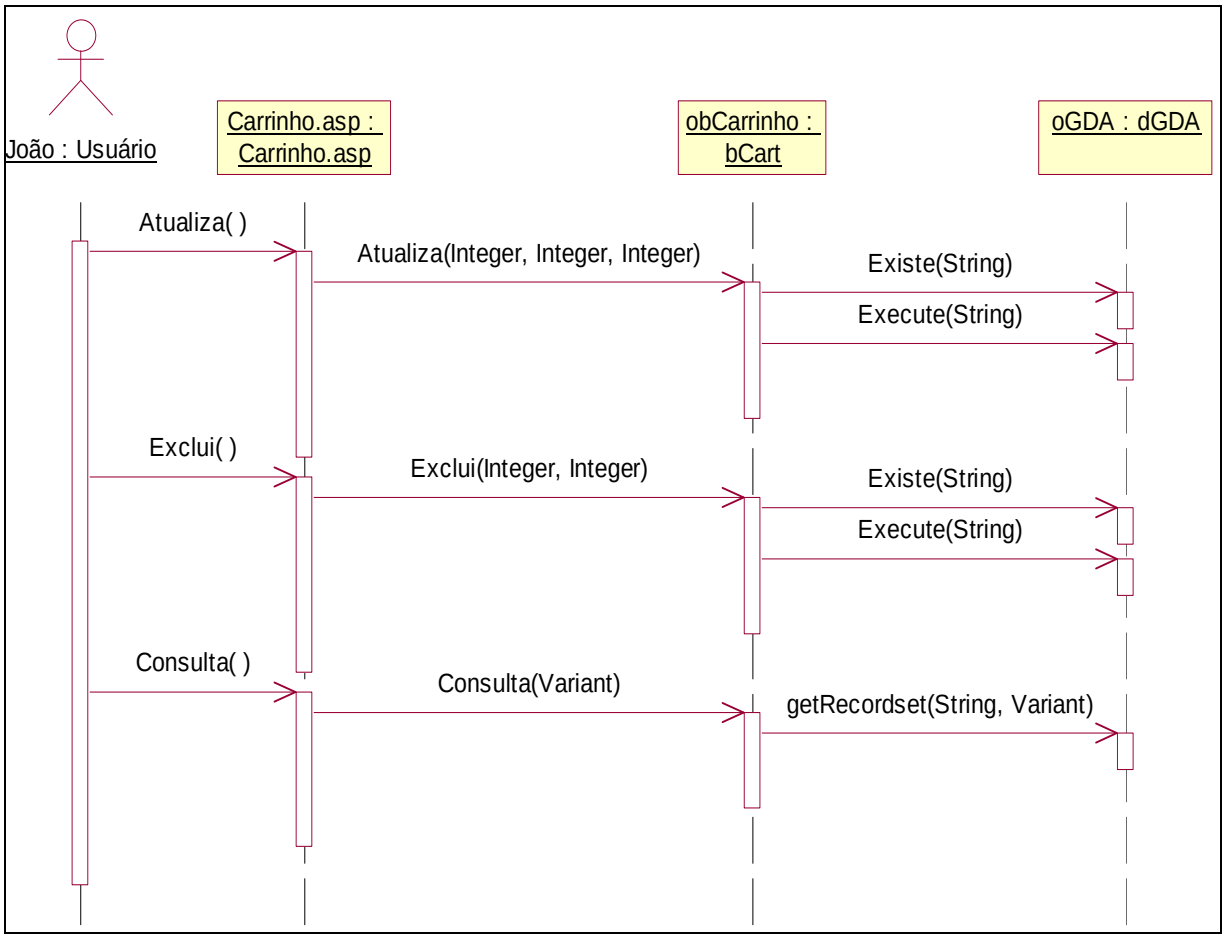
- Atualiza: o usuário passa as informações dos itens novos e dos itens que deseja alterar as quantidades contidas no carrinho, o sistema chama o método **Atualiza** do objeto **obCarrinho**, que por sua vez chama o método **Existe** do objeto **oGDA** e em seguida chama o método **Execute** desse mesmo objeto. Se o item já está no carrinho, o objeto que contém as regras de negócio da

manutenção do carrinho (**obCarrinho**) envia um comando para fazer uma alteração nesse item e se ele ainda não existir, ele envia um comando para adicionar um novo item;

- Exclui: o usuário passa ao sistema o código do item que deseja retirar do carrinho, o sistema chama o método **Exclui** do objeto **obCarrinho**, que chama os métodos **Existe** e, caso exista, o **Execute** do objeto **oDGA**;

- Consulta: o usuário envia um pedido de consulta, o sistema dispara o método **Consulta** do objeto **obCarrinho**, que dispara o método **getRecordset** do objeto **oGDA** retornando os dados da consulta.

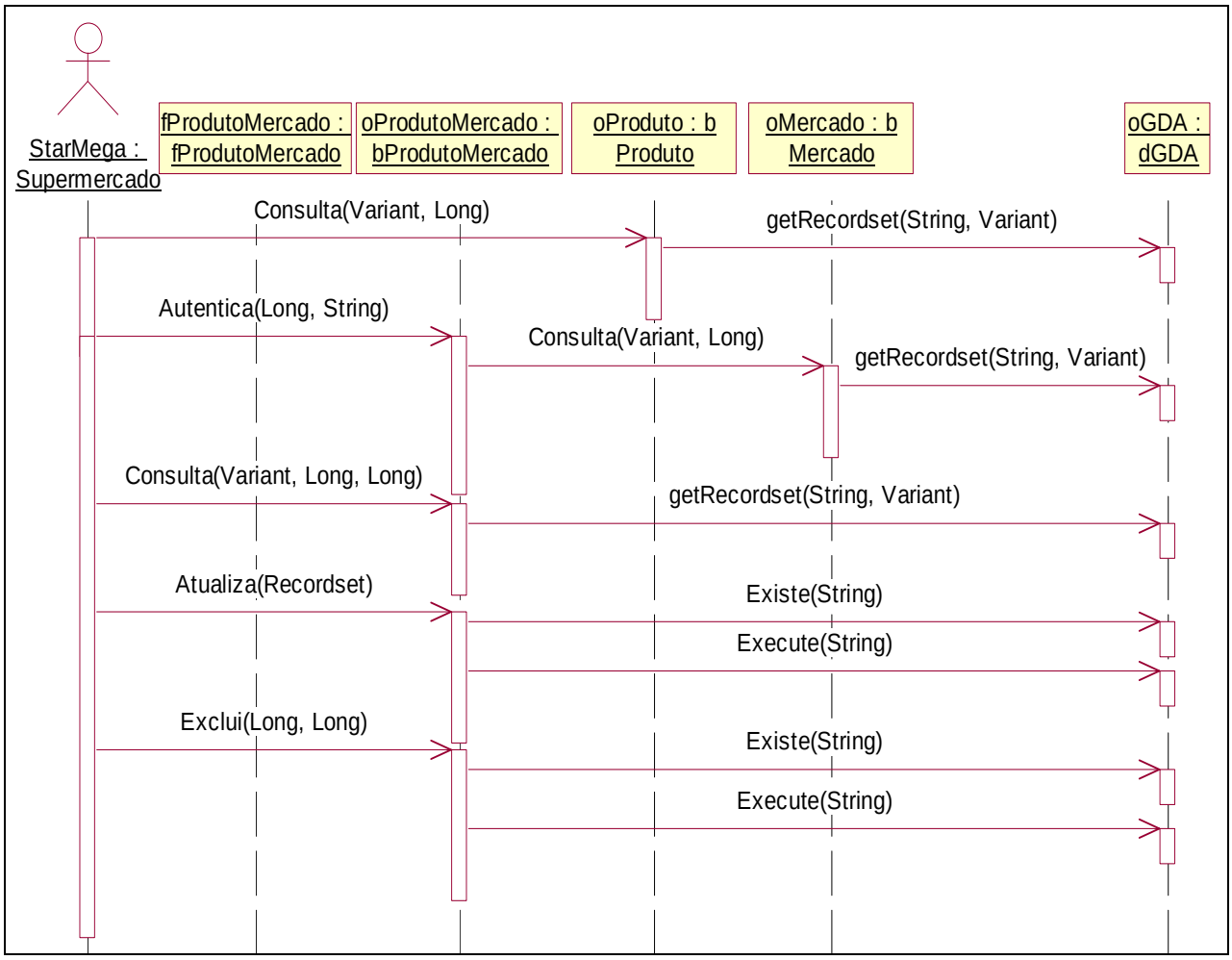
Figura 8 – Diagrama de sequência: manutenção do carrinho de compras pelo usuário



c) manutenção dos preços e quantidades dos produtos pelo supermercado (figura 9). O supermercado efetua a manutenção dos itens disponibilizados por ele no sistema para consulta. Ele tem a possibilidade de consultar esse produtos (comando

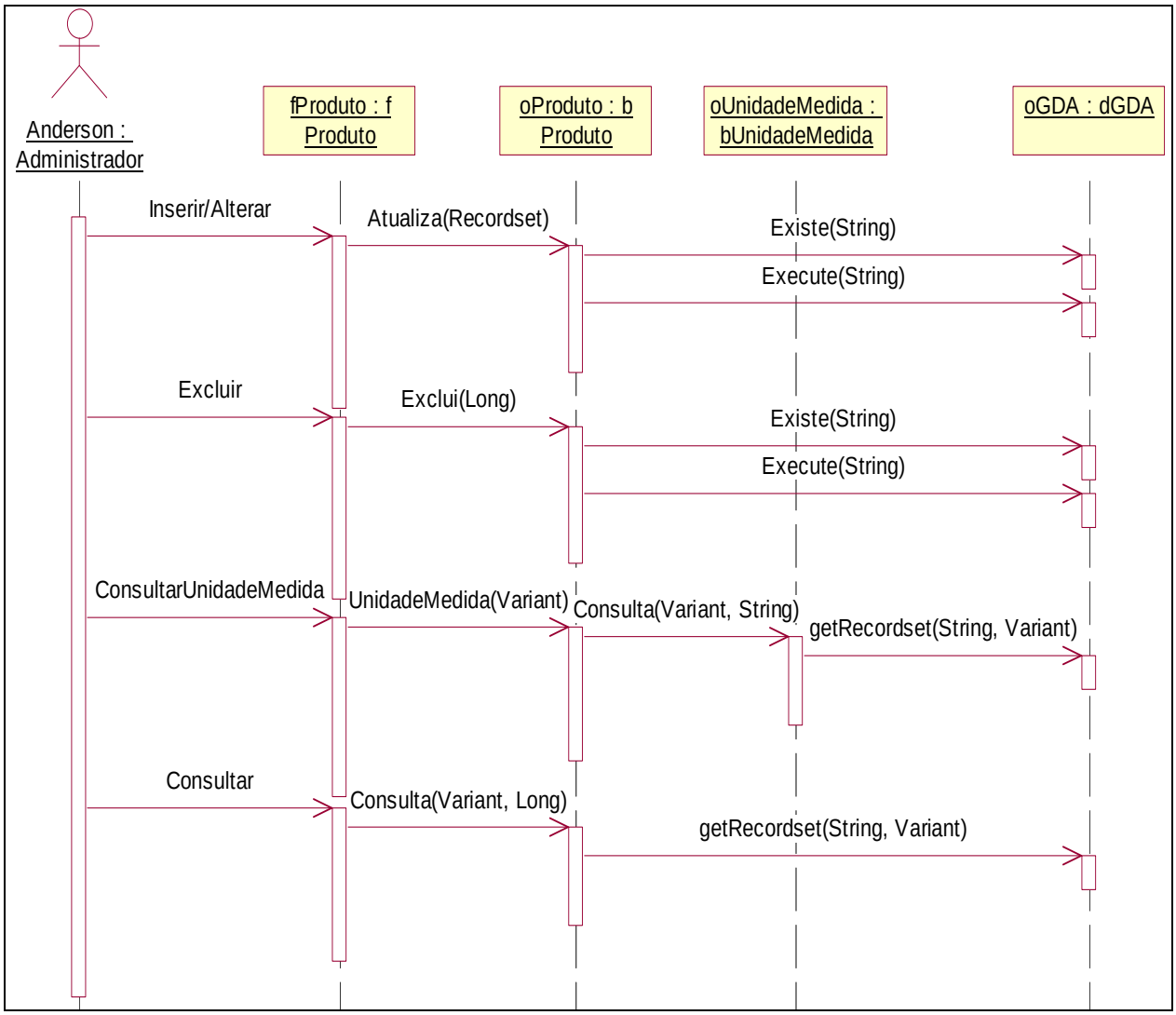
Consulta), incluir e alterar (comando **Atualiza**), excluir (comando **Exclui**) e se autenticar (comando **Autentica**) para que ele tenha permissão de efetuar as alterações e exclusões dos produtos disponibilizados.

Figura 9 – Diagrama de sequência: preços e quantidade pelo supermercado



d) manutenção de produtos pelo administrador (figura 10). O administrador do sistema é responsável por manter o cadastro de produtos. Para isso ele pode Inserir/Alterar os produtos (comando **Atualiza**), excluir (comando **Excluir**), consultar as unidades de medidas do produto (comando **UnidadeMedida**) e verificar os produtos existentes (comando **Consulta**).

Figura 10 – Diagrama de sequência: manutenção de produtos pelo administrador



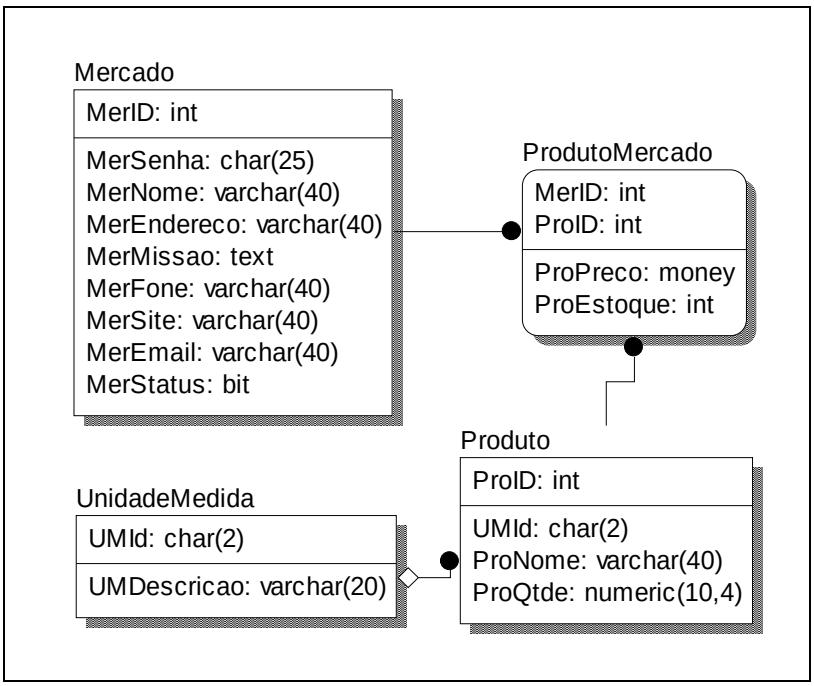
e) manutenção do cadastro (básico) de supermercados pelo administrador. O administrador pode consultar, incluir, alterar e excluir os supermercados cadastrados;

f) manutenção das unidades de medida (cadastro básico) pelo administrador. O administrador pode consultar, incluir, alterar e excluir as unidades de medida cadastradas.

3.1.4 MODELAGEM DE DADOS

A modelagem de dados Entidade-Relacionamento tanto para o Sql Server quanto para o Access foi feita utilizando a ferramenta ERwin da Platinum. O modelo de dados para o Sql Server 7 é demonstrado na figura 11.

Figura 11 – Modelagem de dados Entidade-Relacionamento física para Sql Server 7



O modelo de dados para o Access e para o Sql Server 7 diferem basicamente na nomenclatura dos tipos de dados, que são específicos de cada um. Por exemplo os campos “int” do Sql Server são equivalentes ao “long integer” do Access, o “varchar” ao “text”, o “text” ao “memo”, o “money” ao “currency” e assim por diante.

Essas tabelas serão utilizadas por todo o sistema para a persistência dos dados, sendo acessadas sempre por objetos pertencentes à camada de dados da aplicação. Abaixo são definidas a descrição e utilidade de cada tabela bem como dos seus atributos

A tabela 6 define as tabelas do modelo de dados.

Tabela 6 – Tabelas do modelo de dados

Mercado	Possui o cadastro dos supermercados que disponibilizam seus produtos no sistema.
Produto	Possui o cadastro dos produtos utilizados. Pode-se notar que os códigos dos produtos são os mesmos para todos os supermercados, por isso seria uma idéia interessante utilizar os números que estão no código de barras dos produtos como chave primária da tabela.
UnidadeMedida	Possui as unidades de medida que são utilizadas para quantificar os produtos.
ProdutoMercado	Possui as informações referentes aos produtos em cada supermercado, como o preço e a quantidade em estoque.

A tabela 7 demonstra o dicionário de dados dos atributos do modelo, que foi exportado da ferramente *case Ratinal Rose* para o *Microsoft Excel*, tratada e trazida para esse documento. Ela demonstra os tipos de dados relativos ao Sql Server.

Tabela 7 – Dicionário de dados do sistema

Entidade	Atributo	Definição	Tipo de Dado
Mercado	MerEmail	Email para contato.	varchar(40)
	MerEndereco	Endereço comercial do estabelecimento.	varchar(40)
	MerFone	Telefone.	varchar(40)
	MerID	Chave primária identificadora do estabelecimento.	Int
	MerMissao	Descrição da missão ou qualquer outro texto informado pelo supermercado.	Text
	MerNome	Nome Fantasia do estabelecimento.	varchar(40)
	MerSenha	Senha para acesso às informações administrativas.	char(25)
	MerSite	Endereço WWW do site do estabelecimento.	varchar(40)
	MerStatus	Atributo interno que indica se o supermercado está ativo ou inativo no sistema.	bit

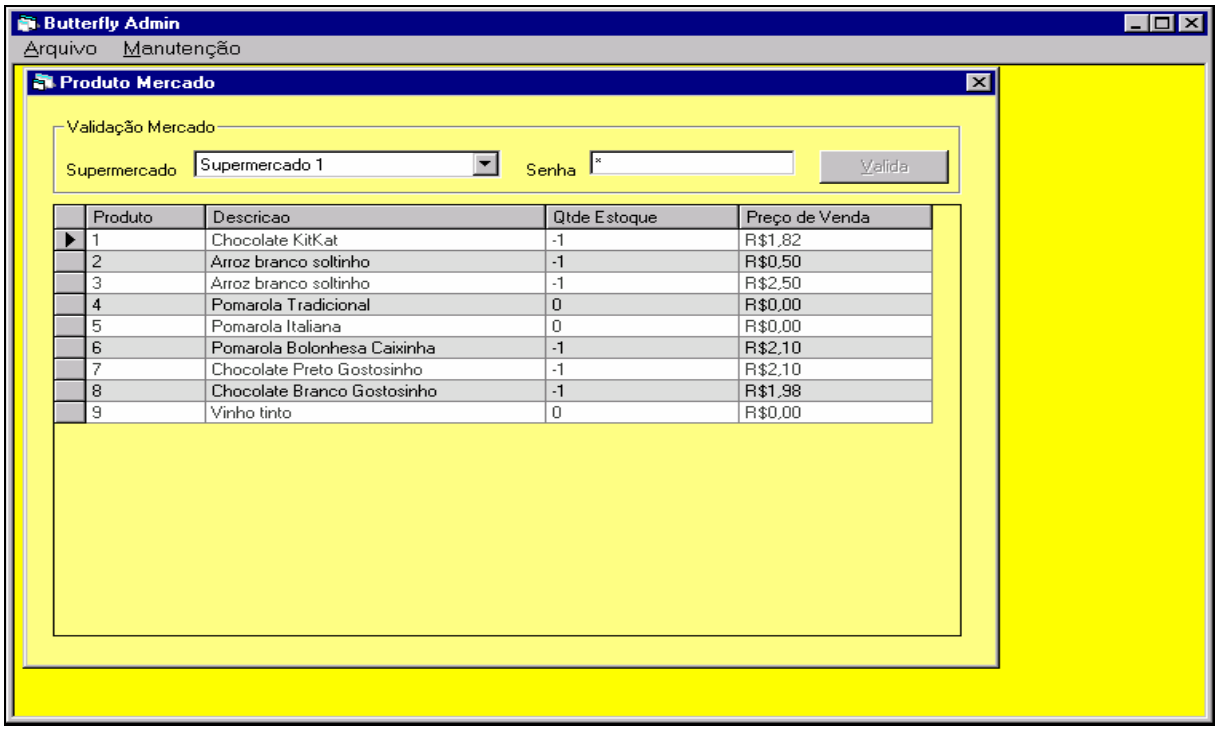
Produto	ProID	Chave primária identificadora do produto, deve preferencialmente ser igual ao próprio código de barra do produto.	Int
	ProNome	Nome/Descrição do produto.	varchar(40)
	ProQtde	Quantidade em relação às unidades de medida. Ex. 200 (gramas).	numeric(10,4)
	UMId	Identificador da unidade de medida do produto. Chave secundária.	char(2)
ProdutoMercado	MerID	Identificador do Mercado do Produto. Chave secundária.	int
	ProEstoque	Quantidade do produto que o supermercado dispõe em estoque.	int
	ProID	Identificador do Produto. Chave secundária.	int
	ProPreco	Preço do produto práticado pelo determinado supermercado.	money
UnidadeMedida	UMDescricao	Descrição da Unidade de Medida	varchar(20)
	UMId	Chave primária identificadora da unidade de medida. Deve ser a própria unidade de medida abreviada. Ex.: Kg, mm	char(2)

3.2 IMPLEMENTAÇÃO

Para resolver o problema foram desenvolvidas duas aplicações distintas que foram modeladas em multicamadas e utilizam objetos COM, implementados com o Visual Basic exclusivamente para o sistema, que podem ou não estar no *Application Server*:

1) **Butterfly Admin**: Aplicação desenvolvida em Visual Basic e que será utilizada pelos supermercados e administradores do sistema para a entrada e manutenção das tabelas do sistema, bem como as de produto e supermercados (figura 12).

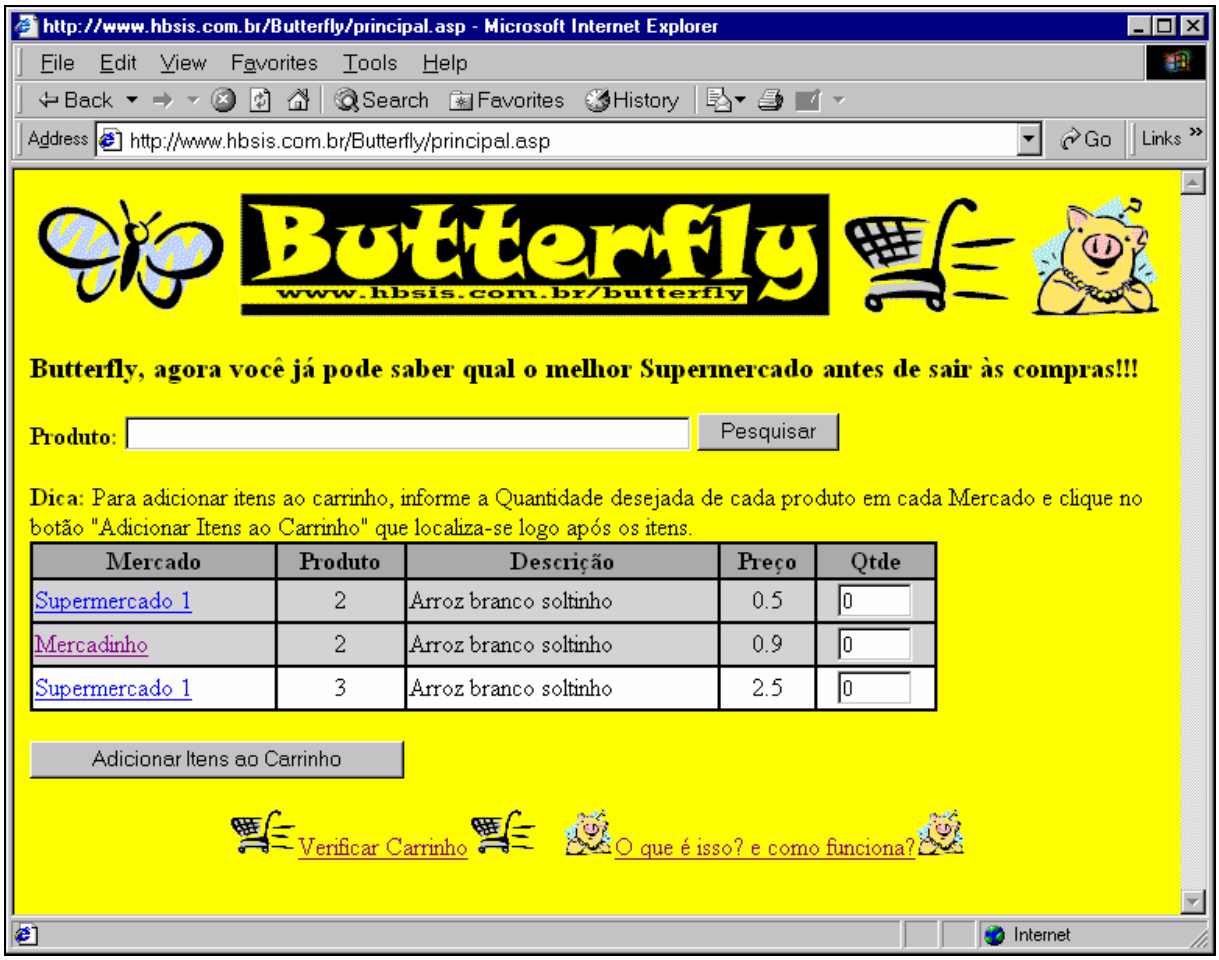
Figura 12 – Butterfly Admin



	Produto	Descricao	Qtde Estoque	Preço de Venda
▶	1	Chocolate KitKat	-1	R$1,82
	2	Arroz branco soltinho	-1	R$0,50
	3	Arroz branco soltinho	-1	R$2,50
	4	Pomarola Tradicional	0	R$0,00
	5	Pomarola Italiana	0	R$0,00
	6	Pomarola Bolonhesa Caixinha	-1	R$2,10
	7	Chocolate Preto Gostosinho	-1	R$2,10
	8	Chocolate Branco Gostosinho	-1	R$1,98
	9	Vinho tinto	0	R$0,00

2) **Butterfly Query**: Aplicação desenvolvida em ASP com auxílio do Visual Interdev e que será utilizada pelos usuários do sistema para fazer as devidas consultas na base de dados a fim de descobrirem informações sobre os produtos do supermercado. Essa parte do sistema está temporariamente disponibilizada na Internet (figura 13).

Figura 13 – Butterfly Query



3.2.1 DESENVOLVIMENTO DOS OBJETOS

Conforme a especificação do sistema, detalhada na seção anterior, para resolver alguns problemas específicos da aplicação foram desenvolvidos alguns objetos. Esses objetos foram desenvolvidos no padrão especificado pelo modelo COM, utilizando-se o Visual Basic que encapsula todos os detalhes da implementação desse modelo.

A figura 14 mostra a tela do Visual Basic utilizada para a criação de novos projetos. Nesse caso utiliza-se um projeto do tipo “*ActiveX DLL*” (utilizado para gerar objetos COM no Visual Basic) que irá conter os objetos do sistema.

Figura 14 – Criação de um projeto no Visual Basic

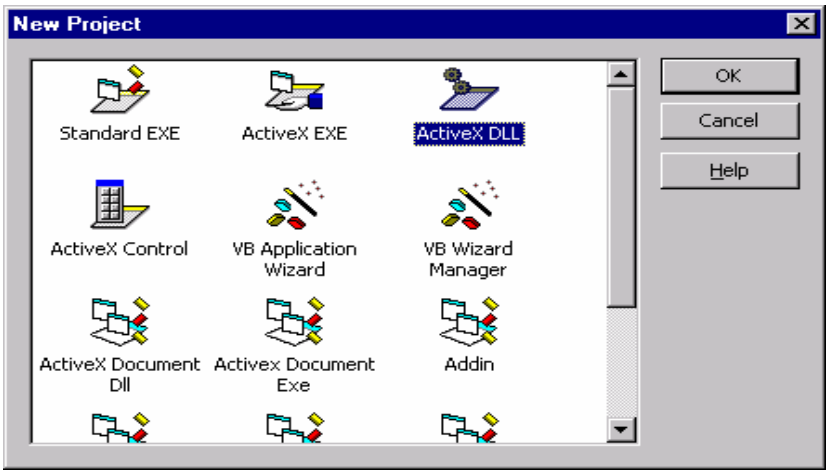
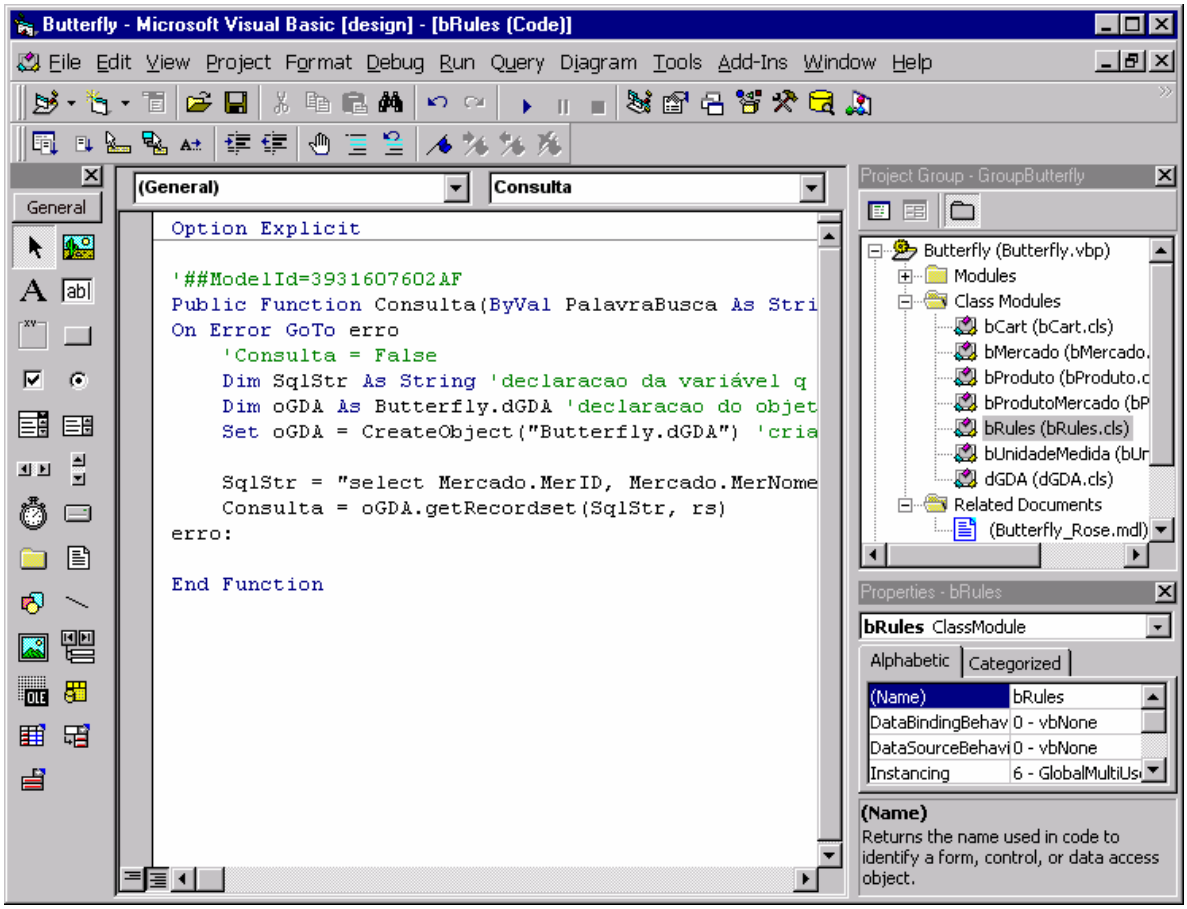


Figura 15 – Ambiente de desenvolvimento do Visual Basic, exemplo de implementação de classes e classes do projeto Butterfly



A figura 16 demonstra 3 coisas:

a) o ambiente de desenvolvimento do Visual Basic;

b) um exemplo de codificação de uma classe no Visual Basic;

c) as classes desenvolvidas no projeto Butterfly que é um projeto do tipo COM ActiveX DLL e que possui as classes dos objetos das camadas de negócio e de dados.

3.2.1.1 DETALHES DE IMPLEMENTAÇÃO DOS OBJETOS

Nessa seção serão demonstrados os trechos de código relevantes referentes ao desenvolvimento das classes negócio e dados do protótipo.

Conforme a arquitetura de 3 camadas determina, para que os objetos de negócio acessem os dados, ele necessitam sempre enviar mensagens para os objetos da camada de dados. Sendo assim, foi desenvolvido um objeto genérico na camada de dados que recebe como parâmetro as *strings* SQL que serão executadas no banco, conforme será mostrado mais adiante nesse capítulo.

Por definição os métodos *Class_Initialize()* e *Class_Terminator()* são utilizados no VB como construtores e destrutores das classes. O quadro 3 exemplifica um construtor e o quadro 4 exemplifica um destrutor típicos no protótipo.

Quadro 3 – Construtor de classes em VB

```
Private Sub Class_Initialize()
    Set oGDA = New Butterfly.dGDA
End Sub
```

Quadro 4 – Destrutor de classes em VB

```
Private Sub Class_Terminate()
    Set oGDA = Nothing
End Sub
```

3.2.1.1.1 DETALHES DOS OBJETOS DE DADOS

Todo o acesso a dados da aplicação, como criar a conexão com o banco de dados, efetuar solicitações SQL e armazenar os dados em estruturas para o acesso da aplicação, são feitos utilizando-se o componente ActiveX ADO (*ActiveX Data Objects*) 2.5. O componente

ADO possui classes específicas para a criação de conexões e *recordsets* que são utilizados no protótipo.

Conforme visto anteriormente o único objeto que existe na camada de acesso a dados do protótipo é o objeto **dGDA** (de *Generic Data Object*). Ele é responsável pela persistência, ou seja, a gravação efetiva das informações no banco de dados e é um objeto genérico podendo ser acessado por qualquer outro objeto para essa finalidade.

Esse objeto de dados, consequentemente, é responsável também por abrir uma conexão com o banco de dados. Para facilitar mudanças, foi criado o módulo **Global.bas** no VB para armazenar a *string* de conexão do banco de dados. O quadro 5 mostra como foi implementada a interface de acesso e a *string* de conexão para o Sql Server 7 e para o Access. Para alternar entre um banco e outro basta descomentar a linha da *string* de conexão relativo ao banco que se deseja conectar, comentar a outra e redistribuir o objeto.

Quadro 5 – *String* de conexão com o SQL Server

```
Function StringConexao() As String
    'SqlServer 7
    StringConexao = "Provider=SQLOLEDB.1;User ID=sa;Initial
Catalog=Butterfly;Data Source=HB2000"
    'Access
    'StringConexao = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\Arq\Tcc\butterfly.mdb"
End Function
```

O quadro 6 mostra como foi implementada a conexão com o banco de dados no método **Execute** do objeto **dGDA**. Os comentários que iniciam com **##ModelId** foram incluídos pelo Rational Rose durante as engenharias reversas e inversas para seu próprio controle.

Quadro 6 – Conexão com o banco no método Execute do objeto dGDA

```
'##ModelId=393160770379
Private conn As ADODB.Connection 'Declara o objeto de conexão ao banco
'##ModelId=393160770397
Private SqlConn As String

'##ModelId=3931607800B4
Private Sub Class_Initialize()
    SqlConn = StringConexao() 'busca a string de conexão do módulo
    Set conn = New ADODB.Connection 'Instancia o objeto
```

```
End Sub

'##ModelId=393160780186
Public Function Execute(ByVal SqlString As String) As Boolean
    Dim errConn As ADODB.Error
    On Error GoTo erh
    conn.Open SqlConn
    Dim x As Variant
    x = conn.Execute(SqlString)
    conn.Close
    Execute = True
    Exit Function
erh:
    Execute = False
    conn.Close
End Function
```

Para que se possa passar o objeto *recordset* do ADO como referência entre os métodos do objeto, é necessário declará-lo como *variant*. O quadro 7 exemplifica essa passagem de parâmetros através da variável **RsRetorno** e como o mesmo deve ser instanciado para tornar-se um objeto do tipo *recordset*.

Quadro 7 – Passagem de objetos por referência via parâmetro

```
Public Function getRecordset(ByVal SqlString As String, ByRef RsRetorno As
Variant) As Boolean
On Error GoTo erro
    GetRecordset = False
'    Dim rs As ADODB.Recordset
    conn.Open SqlConn
    Set RsRetorno = New ADODB.Recordset 'instancia como um recordset
```

3.2.1.1.2 DETALHES DOS OBJETOS DE NEGÓCIO

Todo o código referente as lógicas do negócio de um sistema fica armazenado nos objetos da camada de regras de negócio.

Normalmente comandos SQL são muito relevantes e possuem boa parte do conhecimento de muitas aplicações. Algumas vezes, bem como nesse protótipo, eles são encapsulados nos objetos das regras de negócio. O quadro 8 mostra uma instrução *select* de SQL utilizada pelo método **Consulta** da classe **bRules**, responsável pela consulta de produtos pelo usuário.

Quadro 8 – Instrução *select* SQL encapsulada no método Consulta da classe dGDA

```
Public Function Consulta(ByVal PalavraBusca As String, ByRef rs As Variant)
As Boolean
```

```
On Error GoTo erro
    'Consulta = False
    Dim SqlStr As String 'var. q recebe a SQL string
    Dim oGDA As Butterfly.dGDA 'objeto de acesso a dados
    Set oGDA = CreateObject("Butterfly.dGDA") 'criação do objeto de acesso
a dados

    SqlStr = "select Mercado.MerID, Mercado.MerNome, Mercado.MerSite,
Produto.ProID, Produto.ProNome, ProdutoMercado.ProPreco from
ProdutoMercado, Produto, Mercado Where ProdutoMercado.ProID = Produto.ProID
and ProdutoMercado.MerID = Mercado.MerID and Produto.ProNome like '%" &
PalavraBusca & "%' Order by Produto.ProID, ProdutoMercado.ProPreco"

    Consulta = oGDA.getRecordset(SqlStr, rs)
erro:

End Function
```

O quadro 9 exemplifica uma instrução *insert* e uma *update* de SQL utilizadas no método **Atualiza** da classe **bProduto**, responsável por manter a tabela de cadastro de produtos do banco de dados.

Quadro 9 – *insert* e *update* SQL encapsulados no método Atualiza da classe bProduto

```
Public Function Atualiza(ByVal rs As Recordset) As Boolean
On Error GoTo erh
    Atualiza = False
    Dim SqlStr As String
    While Not rs.EOF
        If oGDA.Existe("Select 1 from Produto where ProID = " &
rs("ProID")) Then
            SqlStr = "Update Produto Set ProNome = '" & rs("ProNome") & "'
,UMID = '" & rs("UMID") & "' ,ProQtde = " & Replace(CStr(rs("ProQtde")),
",", ".") & " where ProID = " & rs("ProID")
        Else
            SqlStr = "Insert into Produto (ProID, ProNome, UMID ,ProQtde)
values (" & rs("ProID") & ", '" & rs("ProNome") & "' ,'" & rs("UMID") & "'
, " & rs("ProQtde") & ") "
        End If
        OGDA.Execute SqlStr
        rs.MoveNext
    Wend
    Atualiza = True
    Exit Function
erh:
End Function
```

O quadro 10 exemplifica uma instrução *delete* de SQL encapsulada do método **Exclui** do objeto **bProduto.**

Quadro 10 – *delete* SQL encapsulado no método Exclui da classe bProduto

```
Public Function Exclui(ByVal ProID As Long) As Boolean
On Error GoTo erh
```

```
    Exclui = False
    Exclui = oGDA.Execute("delete from Produto where ProID = " & ProID)
    Exit Function
erh:
End Function
```

Para realizar a consulta das unidades de medida do produto, o método **UnidadeMedida** da classe **bProduto,** que está na camada de negócios, instancia e faz uso de métodos da classe **bUnidadeMedida,** também da camada de negócios. O código desse método encontra-se no quadro 11.

Quadro 11 – Método UnidadeMedida da classe bProduto

```
Public Function UnidadeMedida(ByRef rs As Variant) As Boolean
On Error GoTo erh
    UnidadeMedida = False
    Dim oUnidadeMedida As Butterfly.bUnidadeMedida
    Set oUnidadeMedida = New Butterfly.bUnidadeMedida
    If oUnidadeMedida.Consulta(rs) Then
        UnidadeMedida = True
    End If
    Set oUnidadeMedida = Nothing
    Exit Function
erh:
End Function
```

Para se manter os itens do carrinho de compras enquanto o usuário estiver navegando na aplicação Butterfly Query, atribuição do objeto **bCart**, é feito uso de um *recordset* do ADO. O quadro 12 demonstra como o *recordset* é instanciado e inicializado no método construtor da classe.

Quadro 12 – Instanciação e inicialização do *recordset* do objeto bCart

```
'##ModelId=3931607A0297
Private Carrinho As ADODB.Recordset
'Esse objeto (bcar) será instanciado um para cada secção (1 para cada
usuário)
'Note q o carrinho possui apenas os campos de Id do Mercado e do Produto e
a Qtde, para economizar memória....

'##ModelId=3931607A02B5
Private Sub Class_Initialize()
    Set Carrinho = New ADODB.Recordset
    Carrinho.Fields.Append "ProId", adInteger
    Carrinho.Fields.Append "MerId", adInteger
    Carrinho.Fields.Append "Qtde", adInteger
    Carrinho.Open
End Sub
```

3.2.2 DESENVOLVIMENTO DA APLICAÇÃO BUTTERFLY ADMIN

Essa aplicação pertencente ao sistema possui várias classes, em geral formulários do VB, pertencentes à camada de interface com o usuário e faz uso de vários dos objetos pertencentes à camada de negócio, conforme a definição demonstrada na figura 6.

3.2.2.1 CADASTROS BÁSICOS

Ele possui as telas ou classes referentes aos cadastros básicos de produtos (**fProduto**)(figura 16) , unidades de medida (**fUnidadeMedida**) (figura 17), supermercados (**fMercado**) (figura 18), bem como a tela de manutenção dos preços e quantidades dos produtos de cada supermercado (**fProdutoMercado**) que interagem com o Administrador e com o Supermercado.

Figura 16 – Cadastro de produtos

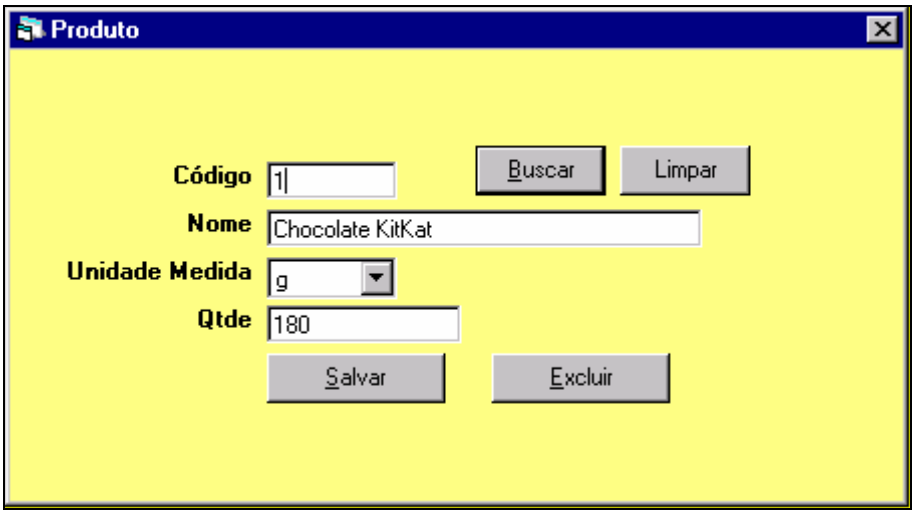


Figura 17 – Cadastro de unidades de medida

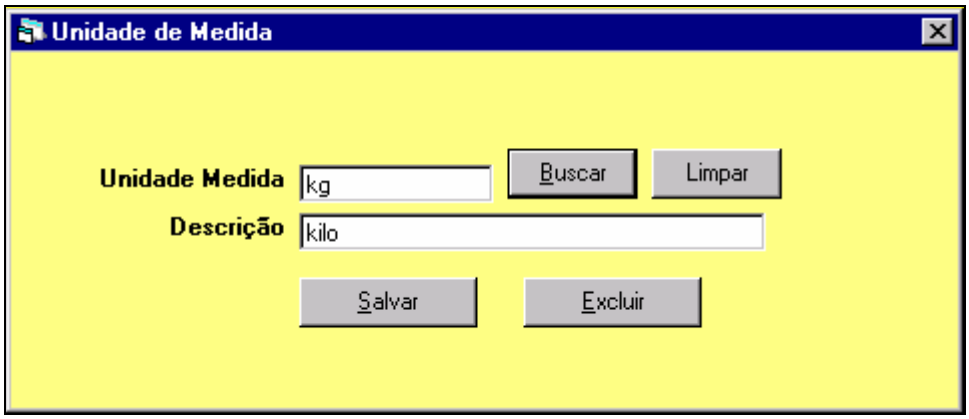
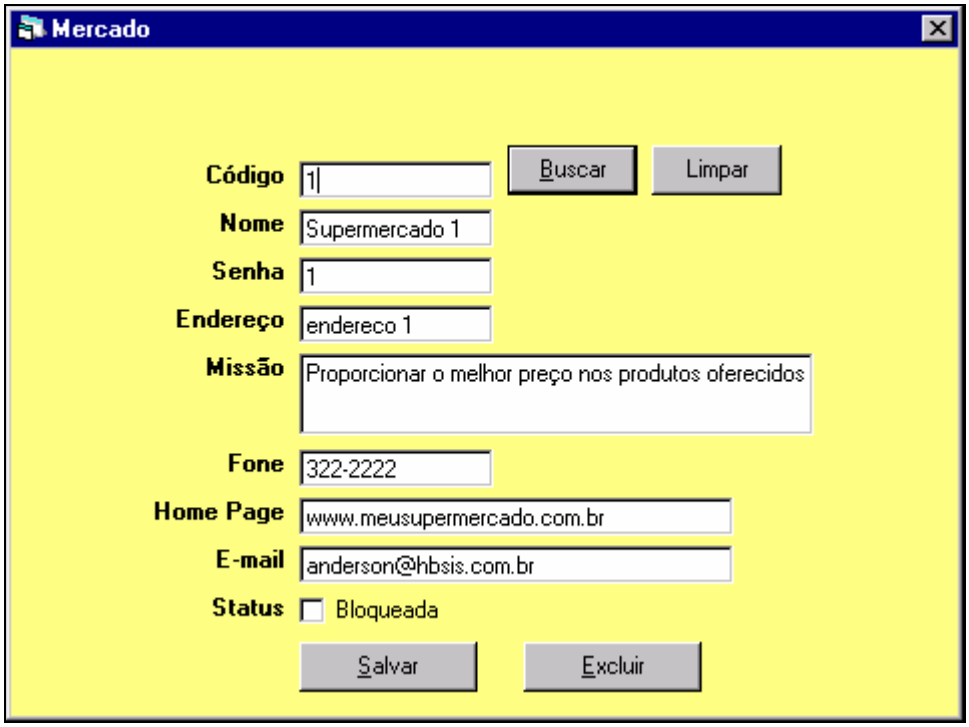


Figura 18 – Cadastro de supermercados



Essas três telas de cadastros básicos possuem praticamente a mesma lógica de funcionamento. Elas possuem um botão **Buscar** que pesquisa no banco de dados pelo código informado no campo texto imediatamente antes dele, um botão **Limpar** que simplesmente inicializa os campos que estão na tela, um botão **Salvar** que salva os dados digitados no banco de dados conforme o código informado e um botão **Excluir** que exclui o registro no banco, conforme o código digitado.

Será mostrado a seguir como foi implementado o cadastro básico de produto, sendo que os outros seguiram a mesma lógica de desenvolvimento.

Inicialmente na tela de produtos é criada uma referência para classe **bProduto** à qual é responsável por manter esse cadastro e outra para o objeto *rs* da classe *recordset* do ADO que será utilizado para ler e gravar os dados do banco (quadro 13).

Quadro 13 – Criação de referências para o cadastro básico de produtos

```
Option Explicit

'##ModelId=393187020042
Private oProduto As Butterfly.bProduto
'##ModelId=39318702006A
Private rs As ADODB.Recordset
```

No quadro 14 são mostrados os eventos *Form_Load* e *Form_unload* que são disparados quando se carrega e quando se fecha a janela, respectivamente. No evento de carga, o objeto **bProduto** é efetivamente instanciado e o *combo-box* que contém as unidades de medidas é carregado com base nas informações retornadas pelo método **UnidadeMedida** do objeto **oProduto**.

Quadro 14 – Eventos de carga e descarga do *form* de produtos

```
'##ModelId=393187030043
Private Sub Form_Load()
    Set oProduto = New Butterfly.bProduto

    '--- Alimento o combo de unidade de medida
    Dim rsUM As ADODB.Recordset
    Set rsUM = New ADODB.Recordset
    If oProduto.UnidadeMedida(rsUM) Then
        While Not rsUM.EOF
            cbUMID.AddItem rsUM("UMID")
            rsUM.MoveNext
        Wend
    End If
    Set rsUM = Nothing
    '------------------------
End Sub

'##ModelId=39318703013E
Private Sub Form_Unload(Cancel As Integer)
    Set oProduto = Nothing
End Sub
```

Quando o usuário dessa aplicação clica no botão de **Excluir** é chamado o método **Exclui** do objeto **oProduto** e passado como parâmetro o código para que ele seja excluído

(quadro 15). A responsabilidade pela exclusão efetiva do produto é do objeto **oProduto** que é uma instância da classe da camada de negócio **bProduto**.

Quadro 15 – Botão Excluir do cadastro de produtos

```
Private Sub cmdExcluir_Click()
    If Not oProduto.Exclui(tProID) Then
       MsgBox "Problemas tentantando excluir Produto. Objeto Produto"
    End If
End Sub
```

O botão **Limpar** do cadastro simplesmente atribui um valor vazio para os *edits* que estão na tela (quadro 16).

Quadro 16 – Botão Limpar do cadastro de produtos

```
Private Sub cmdLimpar_Click()
    TproID = Empty
    TproNome = Empty
    CbUMID.Text = Empty
    TProQtde = Empty
End Sub
```

Quando clica-se sobre o botão salvar, a aplicação monta uma estrutura de dados com um *recordset* e chama o método **Atualizar** do objeto **oProduto** (quadro 17). Esse método faz a inclusão ou, no caso desse código já existir, faz a alteração dos dados do cadastro de produtos.

Quadro 17 – Botão Salvar do cadastro de produtos

```
Private Sub cmdSalvar_Click()
    Set rs = New ADODB.Recordset
    Rs.Fields.Append "ProID", adInteger
    Rs.Fields.Append "ProNome", adVarChar, 40
    Rs.Fields.Append "UMID", adChar, 2
    rs.Fields.Append "ProQtde", adDouble
    rs.Open

    rs.AddNew
    rs("ProID") = tProID
    rs("ProNome") = Trim(tProNome)
    rs("UMID") = Trim(cbUMID.Text)
    rs("ProQtde") = CDbl(tProQtde)
    rs.Update

    If Not oProduto.Atualiza(rs) Then
        MsgBox "Problemas na atualização. Objeto Produto."
    End If
    Set rs = Nothing
End Sub
```

E finalmente o botão **Buscar** do cadastro de produtos chama o método **Consulta** do objeto **oProduto** e alimenta os *edits* da janela com as informações consultadas (quadro 18).

Quadro 18 – Botão Buscar do cadastro de produtos

```
Private Sub cmdBuscar_Click()
    Set rs = New ADODB.Recordset
    If Not oProduto.Consulta(rs, Val(tProID)) Then
        MsgBox "Erro na consulta. Objeto Produto."
    Else
        If rs.EOF Then
            MsgBox "Produto ainda não cadastrado."
        Else
            tProID = rs("ProID")
            tProNome = rs("ProNome")
            cbUMID.Text = rs("UMID")
            tProQtde = rs("ProQtde")
        End If
    End If
    Set rs = Nothing
End Sub
```

Uma pequena variação do cadastro básico que acontece no cadastro de supermercados é o fato dele possuir um campo para a liberação do supermercado para que ele possa efetivamente ser incluído nas consultas realizadas pelo aplicativo **Butterfly Query**. Então no método **Salvar** do *form* de produtos (fProduto) é verificado o *check-box* de *status* e chamado os métodos para liberar ou bloquear o supermercado (quadro 19).

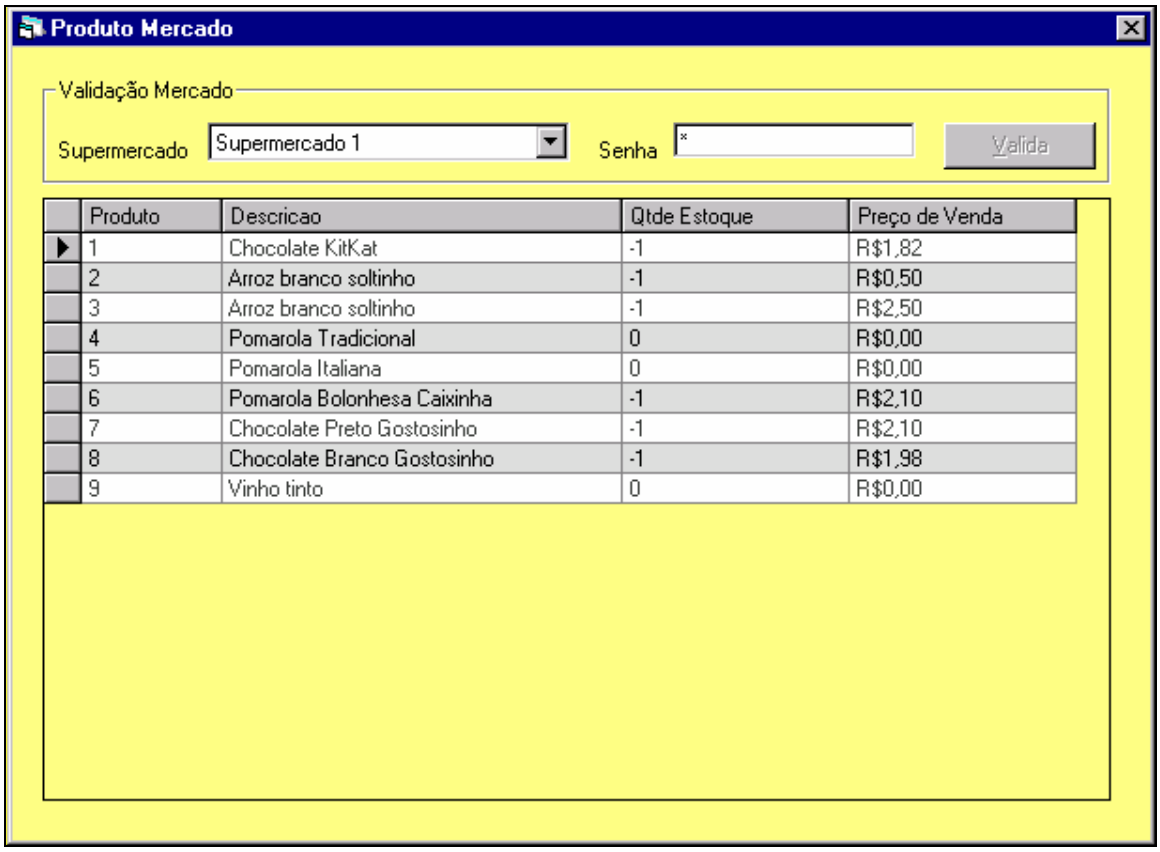
Quadro 19 – Chamada aos métodos para bloquear ou liberar os supermercados

```
Set rs = Nothing
    If cMerStatus.Value Then
        OMercado.Bloqueia tMerID
    Else
        OMercado.Libera tMerID
    End If
```

3.2.2.2 MANUTENÇÃO DE PRODUTOS

Para dar manutenção nos preços e quantidades do produtos de cada supermercado é utilizado o *form* de **Produto**/**Mercado** (figura 19).

Figura 19 – Form de Produto/Mercado



Nele pode-se consultar os dados dos produtos por supermercado e fazer as alterações no valor mediante uma autenticação para cada supermercado.

Essa janela faz uso de um objeto da classe **bProdutoMercado** pertencente à camada de negócios, o qual é responsável pela manutenção dessas informações. Seu funcionamento é muito parecido com o funcionamentos dos cadastros básicos, lendo, gravando e excluindo dados.

A validação da senha do supermercado é feita quando o botão **Verifica** é acionado e no caso da autenticação ser válida, o *grid* é carregado com as informações do produto, conforme código fonte demonstrado no quadro 20.

Quadro 20 – Código referente a validação do supermercado na tela de Produto/Mercado

```
Private Sub cmdValida_Click()
    If Not oProdutoMercado.Autentica(cbMerId.ItemData(cbMerId.ListIndex),
tMerSenha) Then
        MsgBox "Senha não confere..."
```

```
    Else
        'carrega preco e qtde no grid
        GridCarregado = False
        cmdValida.Enabled = False
        Dim rsPM As ADODB.Recordset
        Set rsPM = New ADODB.Recordset
        Dim i As Long
        DBGrid.Redraw = False
        For i = 0 To DBGrid.Rows – 1
            DBGrid.Row = i
            If oProdutoMercado.Consulta(rsPM,
cbMerId.ItemData(cbMerId.ListIndex), DBGrid.Columns(0).Value) Then
                If Not rsPM.EOF Then
                    DBGrid.Columns(2).Value = CInt(rsPM("ProEstoque"))
                    DBGrid.Columns(3).Value = CCur(rsPM("ProPreco"))
                Else
                    DBGrid.Columns(2).Value = 0
                    DBGrid.Columns(3).Value = 0
                End If
                rsPM.Close
            End If
        Next i
        DBGrid.Redraw = True
        If DBGrid.Rows > 0 Then
            DBGrid.Row = 0
        End If
        GridCarregado = True
        '-------------
    End If
End Sub
```

3.2.3 DESENVOLVIMENTO DA APLICAÇÃO BUTTERFLY QUERY

Essa aplicação pertencente ao sistema possui várias classes, em geral páginas ASP geradas com o auxílio do *Visual Interdev*, pertencentes à camada de interface com o usuário e faz uso de vários dos objetos pertencentes à camada de negócio.

3.2.3.1 PÁGINA DE CONSULTA

A janela principal desse aplicativo, que inclusive possui o nome **Principal.asp**, é a tela responsável pelas consultas que o usuário faz à base de dados para verificar os preços praticados pelos diversos supermercados participantes (figura 20).

Figura 20 – Tela de consulta do aplicativo Butterfly Query

Essa janela possui código HTML (quadro 21) e ASP quando é montada e possui um botão chamado **Pesquisar** responsável pelo *submit* do campo de consulta do produto para o servidor que roda a aplicação.

Quadro 21 – Html da página de consulta

```
<html>
<head>
<meta NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<title></title>

</head>
<body bgColor="yellow">

<!-- #INCLUDE FILE="head.htm" -->

<FORM action="principal.asp" id=FORM1 name=FORM1 method=post>

<H3>Butterfly, agora você já pode saber qual o melhor Supermercado antes de
sair às compras!!!</H3>

<P><B>Produto:
<INPUT id=produto name=produto size=50>
<INPUT id=submit1 name=submit1 type="submit" value="Pesquisar">
</B>
</P>
<P>
```

Ela faz uso de uma diretiva *include* de HTML que serve para incluir uma página HTML dentro de outra para se definir as opções de navegação da aplicação que serão utilizadas por todas as janelas. O arquivo **Head.htm** é um simples arquivo HTML que possui a navegação que deve ser utilizada por todas as páginas da aplicação. Foi utilizado esse recurso ao invés da utilização de *frames* HTML pela dificuldade de garantir que os *frames* da aplicação sempre estarão carregados corretamente. O quadro 22 mostra o código HTML contido no arquivo **Head.htm**.

Quadro 22 – Conteúdo do arquivo Head.htm

```
<html>
<head>
<meta NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<title></title>
</head>
<body BGCOLOR="yellow">
<center>
<a href="Principal.asp"><img border="0" src="images/Bfly1.gif" WIDTH="130"
HEIGHT="79"></a>
<img border="0" src="images/Butterfly.gif" WIDTH="380" HEIGHT="79">
<a href="Carrinho.asp"><img border="0" src="images/Cart.gif" WIDTH="118"
HEIGHT="79"></a>
<a href="Info.asp"><img Border="0" src="images/Info1.gif" WIDTH="88"
HEIGHT="79"></a>
</center>
</body>
</html>
```

Quando a página é enviada ao servidor e reprocessada, o código ASP (quadro 23) instancia o objeto **bRules** e executa o método **Consulta** responsável pela consulta aos produtos, bem como monta uma tabela HTML para mostrar os dados encontrados e inclui um botão para adicionar os itens selecionados ao carrinho (figura 21).

Quadro 23 – ASP para montagem do retorno da consulta

```
<%
if not IsEmpty( Request.Form ) then
      if Request("submit1") = "Pesquisar" then
            dim Negocio 'as Butterfly.bRules
            set Negocio = Server.CreateObject("Butterfly.bRules" )
            dim rs 'as adodb.recordset
            if Negocio.Consulta(request("produto"),rs) then
                  if rs.eof then
                        %><P> Produto não encontrado em nenhum mercado.
</P><%
                  else
                        %><B> Dica: </B>Para adicionar itens ao carrinho,
informe a Quantidade desejada de cada produto em cada Mercado e clique no
botão "Adicionar Itens ao Carrinho" que localiza-se logo após os itens.
<br>
```

Figura 21 – Resultado de uma consulta por “pomarola” na tela de consulta

Quando o usuário informa as quantidades de um ou de vários produtos e clica no botão “**Adicionar Itens ao Carrinho**” o aplicativo roda o código ASP para instanciar o objeto **bCart** e fazer a adição dos itens no carrinho (quadro 24).

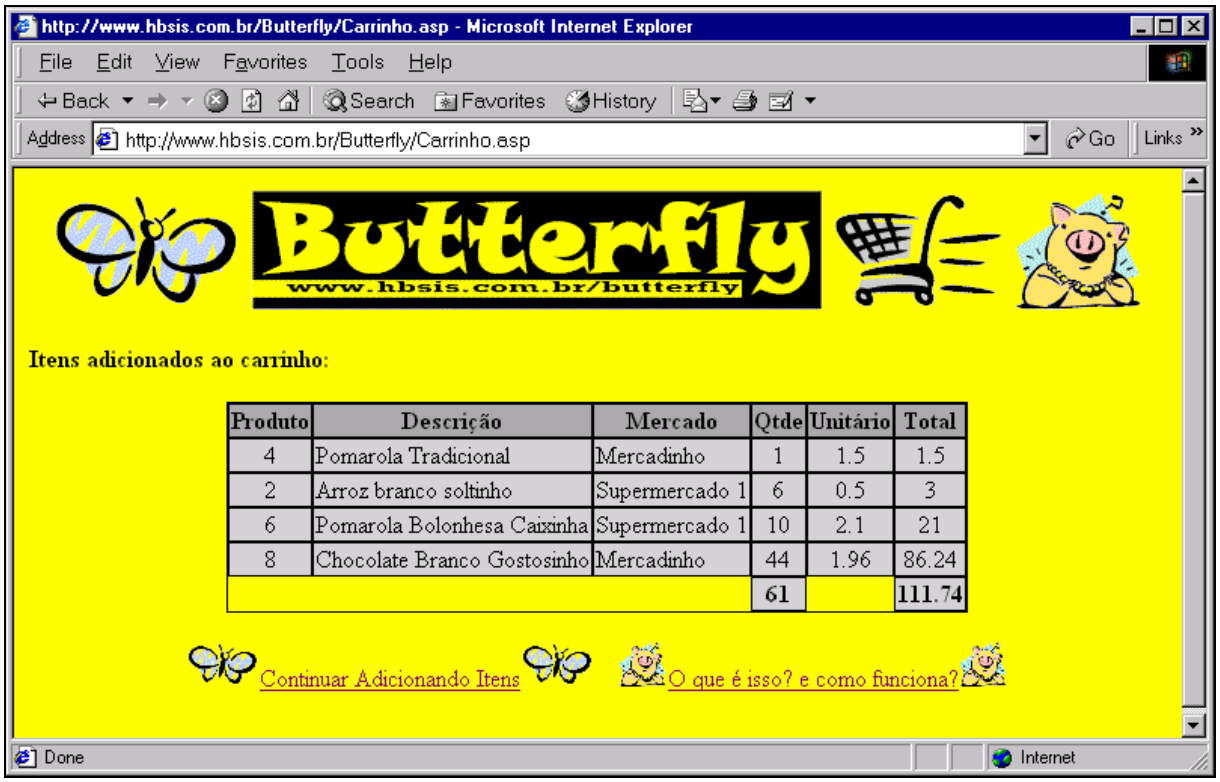
Quadro 24 – Código ASP para adicionar itens ao carrinho

```
            Linha=1
            while not isempty( Request( "parQtde" & Linha ) )
                  if int(Request( "parQtde" & Linha )) > 0 then
                        session("Carrinho").Atualiza Request( "parPro" &
Linha ), Request( "parMer" & Linha ), Request( "parQtde" & Linha )
                  end if
                  Linha=Linha+1
            Wend
            %><Br><B> Dica: </B>Itens adicionados ao Carrinho. Para
visualizar os itens no carrinho, clique em "Verificar Carrinho" ou continue
pesquisando para incluir mais itens. <br> <%
```

3.2.3.2 PÁGINA DE MANUTENÇÃO DO CARRINHO

Após terem sido adicionados itens ao carrinho pode se ter acesso ao carrinho de compras (figura 22) para verificar os itens escolhidos anteriormente, alterá-los ou excluí-los, através do *link* "**Verificar Carrinho**".

Figura 22 – Itens do carrinho de compras



Essa janela utiliza uma instância da classe **bCart**, responsável por fazer a manutenção dos itens no carrinho. Os itens do carrinho são todos mantidos na memória utilizando-se um *recordset* no servidor, então para cada usuário que estiver ativo no sistema será criada uma estrutura desse tipo.

No arquivo **Global.asa,** que é um arquivo que contém código fonte de programas e que é executado pelo servidor de internet, no caso o *Internet Information Server*, foram colocados os códigos para a instanciação e destruição do objeto responsável pelo carrinho. O quadro 25 ilustra como isso foi feito. Os eventos *Session_OnStart* e *Session_OnEnd* são executados pelo IIS sempre que uma nova seção é iniciada e terminada respectivamente. Uma

seção para o servidor de internet é um tempo programado em que o servidor mantém o estado das variáveis para cada usuário que acessa a aplicação.

Quadro 25 – Instanciação e destruição do objeto Carrinho

```
Sub Session_OnStart
      set Session("Carrinho") = server.CreateObject( "Butterfly.bCart")
End Sub

Sub Session_OnEnd
      set Session("Carrinho") = Nothing
End Sub
```

3.3 DISTRIBUIÇÃO FÍSICA DO SISTEMA

Como um dos objetivos desse trabalho era o de desenvolver as aplicações utilizando objetos distribuídos, a título de exemplificação faz-se necessária uma abordagem de como o mesmo ficou distribuído fisicamente.

Nessa seção será demonstrado como o sistema como um todo ficou distribuído fisicamente. Por questões dos recursos disponíveis foi escolhida essa distribuição, porém diversas outras abordagens poderiam ter sido utilizadas para disponibilizá-lo.

3.3.1 DISTRIBUIÇÃO DO BUTTERFLYQUERY

A aplicação ButterflyQuery, que está disponível via Internet, faz uso de dois servidores. O primeiro deles, aqui chamado de **Servidor 1,** é responsável por executar os componentes da camada de negócio e também os componentes da camada de dados, serviço que é feito pelo *Application Server* MTS instalado nela. Esse servidor também é responsável por rodar o banco de dados Sql Server 7 (figura 24) e o banco de dados Access (figura 25), utilizados pelo sistema. O **Servidor 2** por sua vez ficou encarregado de executar o servidor *Web*, que processa a camada de interface da aplicação e a disponibiliza na internet (figura 26). Ele também tem um *Application Server* instalado, pois embora ele não execute os objetos, o servidor Web irá procurá-los nesse servidor e cabe ao *Application Server* direcionar a chamada a esses objetos para o *Application Server* do **Servidor 1**. Para o **Cliente** conectado à Internet e executando um *Web Browser* qualquer, toda essa distribuição física fica transparente, conforme figura 23.

Figura 23 – Distribuição física da aplicação ButterflyQuery

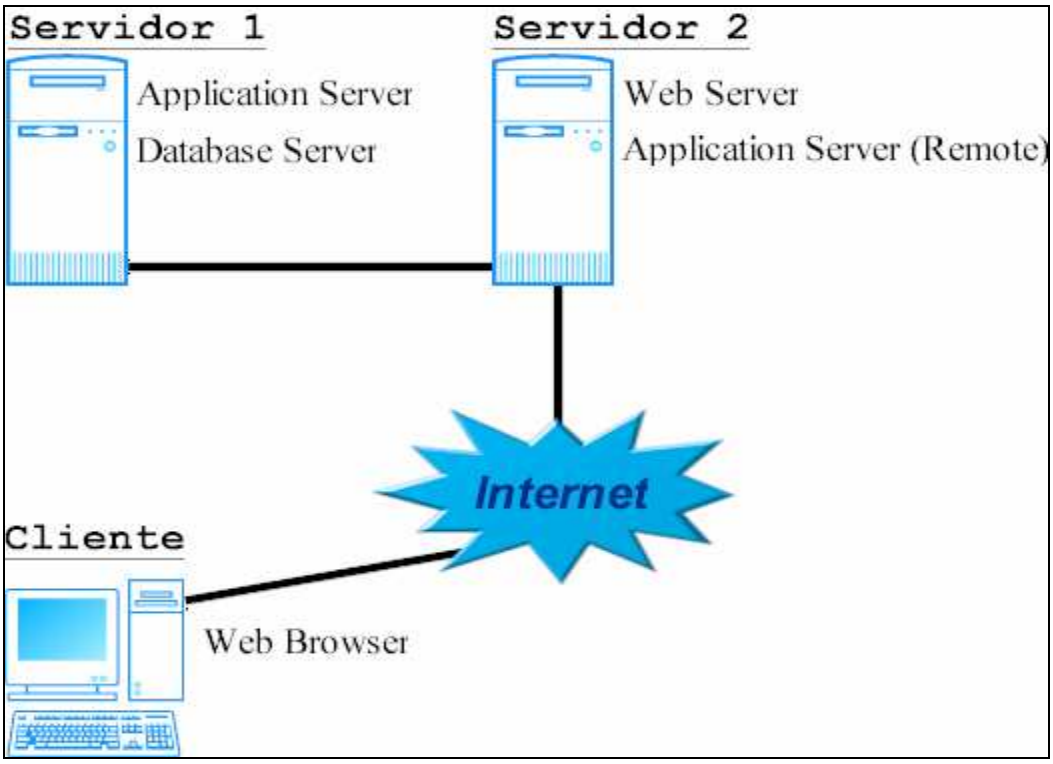


Figura 24 – Tabelas no banco de dados Sql Server 7

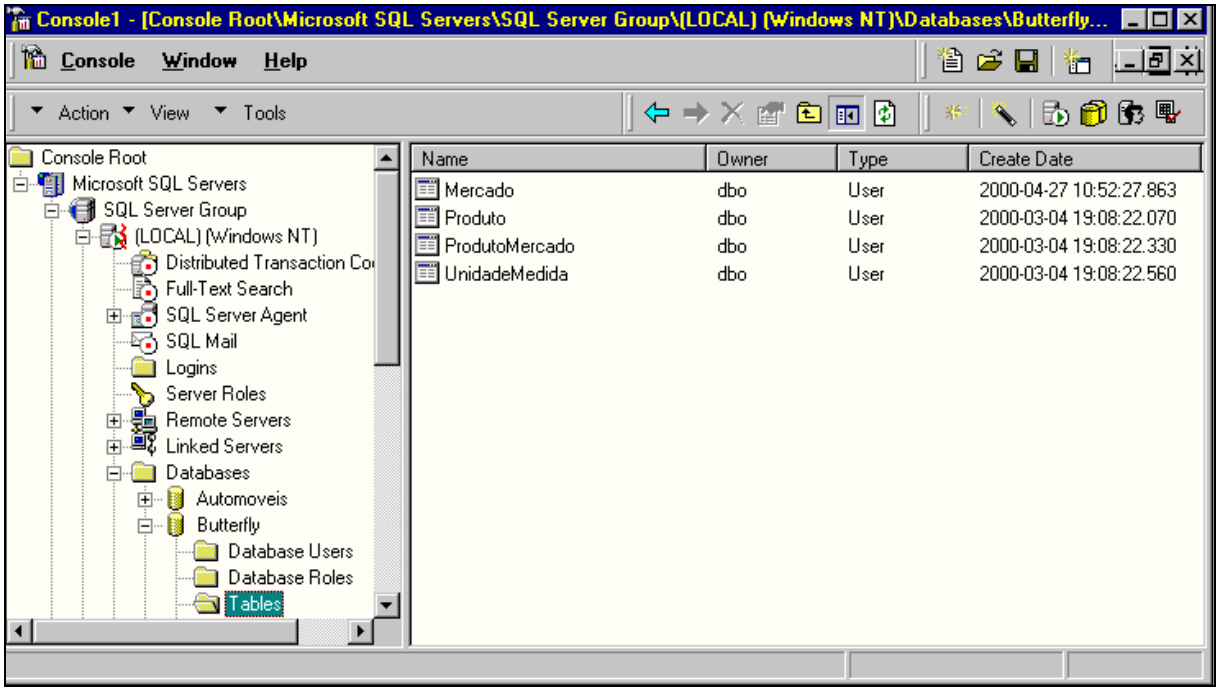


Figura 25 – Tabelas no banco de dados Access

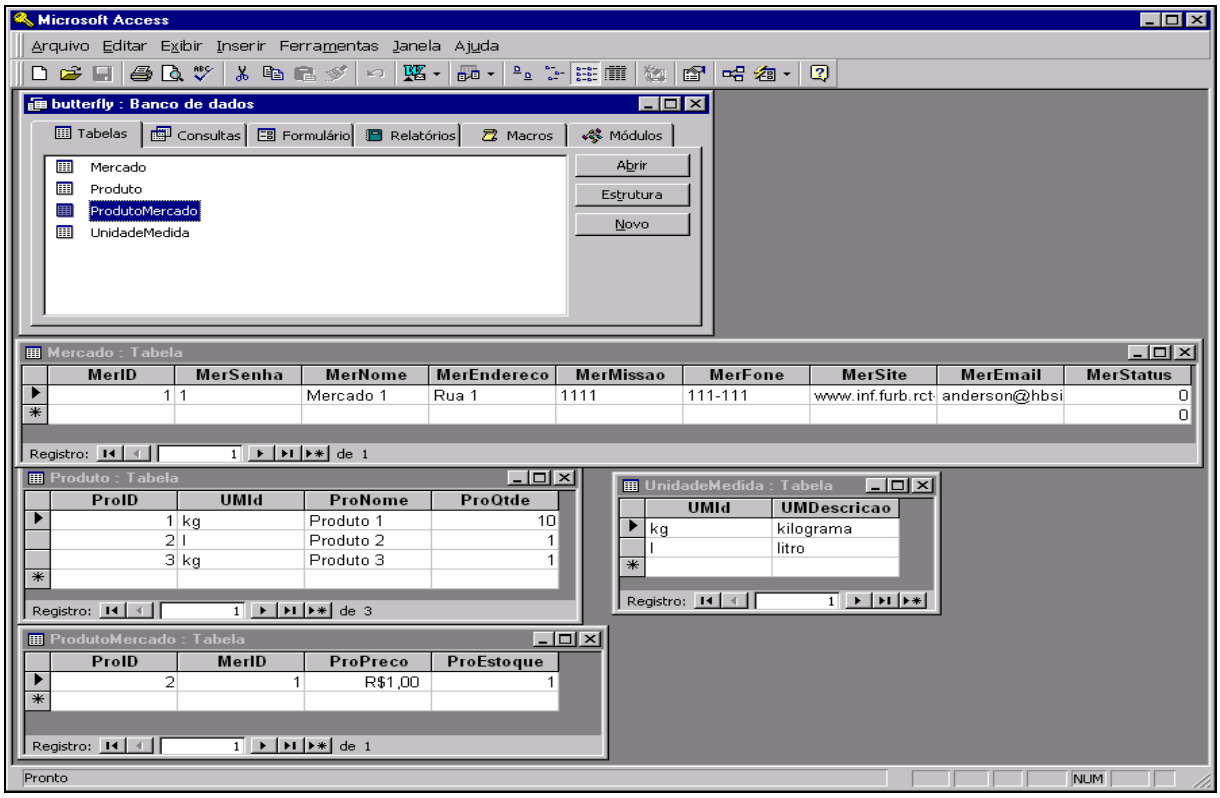
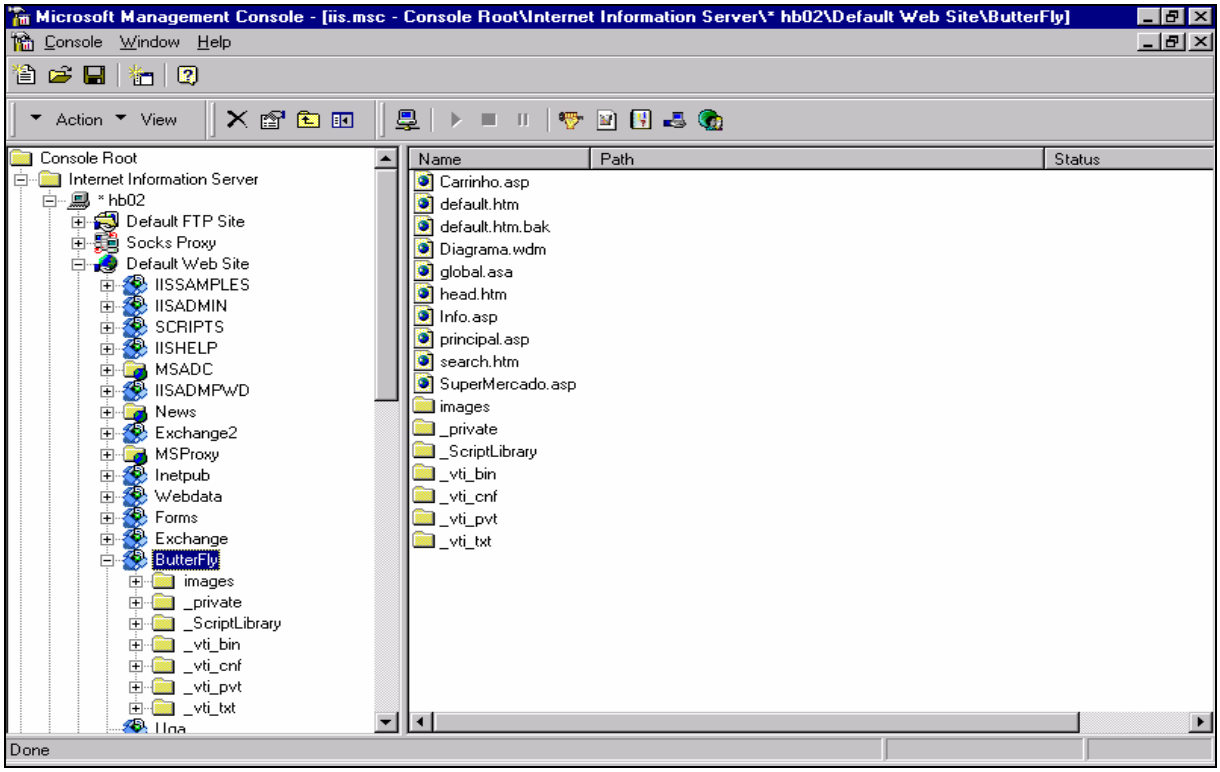
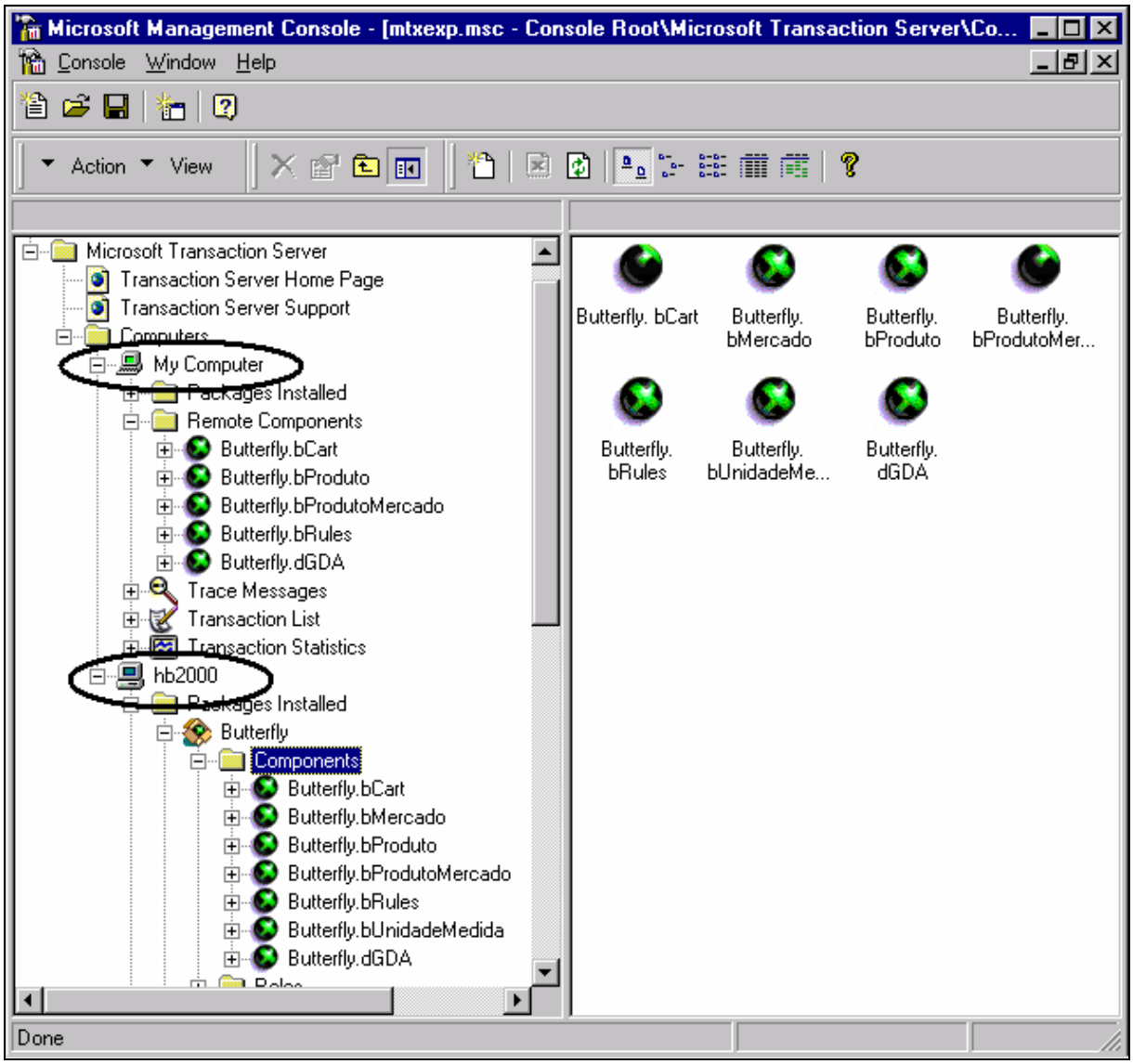


Figura 26 – Aplicativo ButterflyQuery no Servidor Web (*Internet Information Server*)



A figura 27 demonstra como foi configurado o *Application Server* para os dois servidores que executam a aplicação. "**HB2000**" é o nome do **Servidor 1** e "**My computer**" do **Servidor 2**, citados acima. Todos os componentes, conforme já comentado, executam no **Servidor 1**, onde eles estão fisicamente. No **Servidor 2**, que roda junto com o servidor *Web*, estão colocadas, na seção **Remote Components**, apenas as referências aos objetos necessários e quando a aplicação **ButterflyQuery** executa, ela utiliza essa referência para encontrar os componentes.

Figura 27 – Componentes no *Application Server* dos servidores

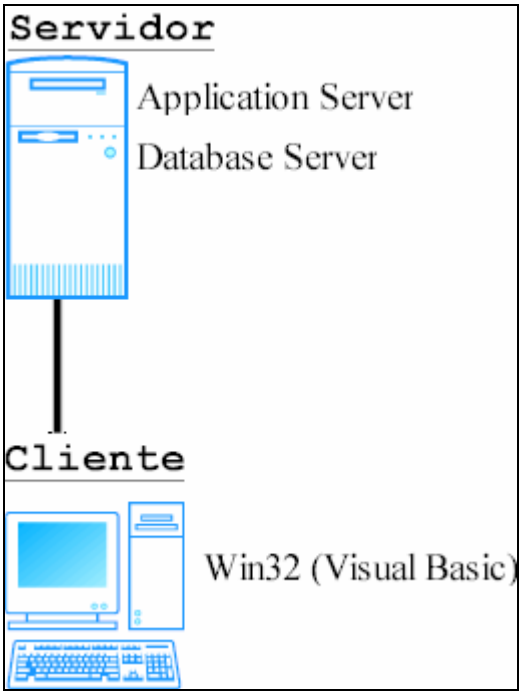


3.3.2 DISTRIBUIÇÃO DO BUTTERFLYADMIN

Por sua vez, a aplicação ButterflyAdmin que possui um cliente *fat*, embora desenvolvido em 3-camadas, utiliza apenas um servidor físico, que é o mesmo utilizado pelo ButterflyQuery e é responsável pelos banco de dados (Sql Server 7 e Access) e também pelo *Application Server* que executa os componentes de negócio e dados. O **Cliente**, por sua vez, é responsável apenas por executar a interface da aplicação e fazer as chamadas aos objetos do **Servidor**, conforme demonstrado na figura 28.

Pelas características dos componentes, esses mesmos objetos que executam no *Application Server* podem também ser utilizados como bibliotecas COM estacionadas na máquina local e serem acessados diretamente pelo *Visual Basic*.

Figura 28 – Distribuição física da aplicação ButterflyAdmin



4 CONCLUSÕES

O desenvolvimento utilizando-se a arquitetura em multicamadas vem se tornando uma tendência cada vez mais forte tendo em vista que supre e suporta boa parte das necessidades trazidas pelo grande e emergente mercado de Internet, como escalabilidade das aplicações, suporte à componentes e objetos distribuídos, métodos mais rígidos no desenvolvimento de componentes a fim de auxiliar na sua reutilização e distribuição, e flexibilidade na manutenção dos sistemas entre outras, necessidades essas que já não podiam ser supridas na sua íntegra pelas aplicações monolíticas e pelas aplicação puramente cliente/servidor.

Essa arquitetura divide claramente a aplicação em três camadas, a camada de interface que possui os serviços de interação com usuários e outros sistemas, a camada de lógica de negócios que é responsável pelos serviços referentes as regras de negócio da aplicação e a camada de dados, que é responsável pelos serviços que tratam do gerenciamento dos dados.

O fato de separar as lógica de negócios em uma camada específica torna flexível a manutenção sempre que uma regra do negócio muda. Da mesma maneira, essa arquitetura torna flexível a interface da aplicação, permitindo que vários componentes de interface acessem as mesmas regras de negócio e sempre de um mesmo lugar. A camada de dados torna flexível, e muitas vezes até independe, a implementação para os diversos servidores de banco de dados.

Muito se especula hoje ainda nas tendências para qual o mercado de internet caminha. Muitos pretendem estabelecer negócios lucrativos nessa nova e incerta área e para isso eles precisam dessa flexibilidade que a arquitetura em multicamadas oferece. E-business, e-commerce, e-solutions, muitas terminologias e “E’s” surgem hoje para definir esse novo modelo emergente de negócios, porém, certamente, apenas as empresas preparadas prosperarão.

Embora tenham sido utilizadas várias tecnologias e conceitos dos quais muitos ainda não amplamente difundidos, e do tempo de desenvolvimento e aprendizado tenderem a ser maiores, depois de aprendidos os conceitos básicos, torna-se bastante fácil e empolgante a tarefa de projetar e desenvolver aplicações utilizando essa arquitetura e esses conceitos aqui apresentados.

Quanto ao contexto social, a idéia do protótipo, se for possível colocá-la em prática, é muito interessante para o público usuário de internet em geral e potenciais consumidores, pois permite a toda uma comunidade se beneficiar das consultas aos preços dos supermercados, permitindo que possam escolher mais fácil e claramente qual o estabelecimento que possui os melhores preços.

4.1 DIFICULDADES ENCONTRADAS

Pode-se citar alguns pequenos problemas e contratempos encontrados durante o desenvolvimento do protótipo:

a) passagem de objetos por parâmetro, por referência, entre componentes: quando tenta-se por exemplo passar um *recordset* por referência para um outro componente como VBscript ocorrem alguns problemas. Para solucionar basta passar o *recordset* como *variant*, conforme documentado no banco de dados de suporte da Microsoft ([MIC1999e]);

b) propriedade **Instancing** das classes dos objetos COM: deve-se sempre ter em mente se o objeto será *stateless* ou *stateful*.

c) propriedade **MTSTransactionMode** das classes dos objetos COM: deve-se sempre observar se o objeto vai trabalhar com transações no *Application Server*;

d) demora na primeira instanciação dos componentes no *Application Server*: o tempo de criação e instanciação dos componentes no *Application Server* tende a ser bastante alto. Para minimizar esse problema configurou-se para que o MTS trabalhasse com tempos maiores de *pooling* para o objeto, ou seja, quando os objetos deixam de ser utilizados pelo sistema eles permanecem na memória por um determinado tempo e se alguma outra conexão for solicitada, os mesmos já terão uma cópia instanciada o que torna muito mais rápido para utilizar esse componente.

4.2 SUGESTÕES

Algumas sugestões de pesquisa em áreas correlatas são:

a) segurança em internet: como acesso aos objetos, transferências de dados seguras utilizando encriptação e segurança e vulnerabilidade dos servidores de internet;

b) *design* de interface para internet: estudos e buscas de interfaces mais atraentes e intuitivas para o usuário ;

c) metodologias de análise e desenvolvimento em multicamadas: buscar um maior aprofundamento nessa arquitetura, detalhando mais profundamente a função de cada camada;

d) implementação de transações com *Application Servers*: estudar e verificar o comportamento de transações entre vários bancos de dados utilizando *Application Servers*;

e) troca de mensagem com servidores de enfileiramento: verificar o funcionamento e utilização de servidores de enfileiramento para envio e recebimento de mensagens assíncronas em uma rede.

REFERÊNCIAS BIBLIOGRÁFICAS

[ANG2000] **Angeloni supermercados.** Jul. 2000. Endereço Eletrônico: http://www.angeloni.com.br

[D-T1998] D-TEC Distributed Technologies GmbH. **3- and n-tier architetures.** 1998. Endereço Eletrônico: http://www.corba.ch/e/3tier.html

[LOJ2000] **LOJAS Americanas.** Mar. 2000. Endereço Eletrônico: http://www.americanas.com.br

[MEN2000] MENDES, João Ricardo Barroca. Discutindo o Mercado e a evolução para objetos distribuídos. **Developers CIO Magazine.** Rio de Janeiro : Axcel Books do Brasil, mar. 2000.

[MIC1996] MICROSOFT Corporation. **Microsoft Solutions Framework.** [S.l. : s.n.], 1996.

[MIC1999a] MICROSOFT Corporation. **Comércio.** 1999. Endereço Eletrônico: http://www.microsoft.com/brasil/comercio

[MIC1999b] MICROSOFT Corporation. **MSDN Library.** [S.l. : s.n.], julho de 1999.

[MIC1999c] MICROSOFT Corporation. **Comércio: revolucionando as relações comerciais.** 2ª Edição. Série Soluções. [S.l. : s.n.], abril de 1999.

[MIC1999d] MICROSOFT Corporation. **Windows DNA** : Windows Distributed interNet Architecture. 1999. Endereço Eletrônico: http://www.microsoft.com/brasil/dna

[MIC1999e] MICROSOFT Corporation. **PRB: Passing parameters by reference to a VB COM object.** Dez. 1999. Endereço Eletrônico: http://support.microsoft.com/support/kb/articles/Q197/9/56.ASP

[MIC1999f] MICROSOFT Corporation. **Microsoft ADO.** Nov. 1999. Endereço Eletrônico: http://www.microsoft.com/data/ado/default.htm

[PAO2000] **PÃO de Açucar.** Mar. 2000. Endereço Eletrônico: http://www.paodeacucar.com.br

[PEN2000] PENNA, Manoel Camillo; Meinetti Jr., Sergio; Malucelli, Vinicius Vendrami. O mercado para projetos com objetos distribuídos. **Developers CIO Magazine.** Rio de Janeiro: Axcel Books do Brasil, mar. 2000.

[SES1997] SESSIONS, Roger. Microsoft's component tier. **Software Development Magazine.** Dez. 1997. Endereço Eletrônico: http://www.sdmagazine.com/breakrm/features/s97df1.shtml

[SES2000] **SÉ Supermercados.** Mar. 2000. Endereço Eletrônico: http://www.sesupermercados.com.br

[UML1999] UML.PDF. **OMG Unified Modeling Language Specification :** v. 1.3. Rational Software Corporation. Jun. 1999. Acrobat Reader. Endereço Eletrônico: http://www.rational.com/uml