UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UMA APLICAÇÃO COMERCIAL, UTILIZANDO BANCO DE DADOS CACHÉ COM INTERFACE WEB

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

MAURÍCIO ROGÉRIO OBENAUS

BLUMENAU, JULHO/2000

2000/1-52

PROTÓTIPO DE UMA APLICAÇÃO COMERCIAL, UTILIZANDO BANCO DE DADOS CACHÉ COM INTERFACE WEB

MAURÍCIO ROGÉRIO OBENAUS

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. Luiz Bianchi — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Luiz Bianchi

_____

Prof. Oscar Dalfovo

_____

Prof. Everaldo Artur Grahl

DEDICATÓRIA

Dedico este trabalho aos meus pais pelo apoio que sempre me deram.

AGRADECIMENTOS

Agradeço a Deus,

que me deu paciência e tranqüilidade para elaboração deste trabalho.

Agradeço a meus familiares,

pelo apoio que recebi e por tudo que me ensinaram.

Agradeço ao professor Luiz Bianchi,

por ter me orientado neste trabalho.

Agradeço a meus amigos,

que sempre estiveram ao meu lado nos momentos que precisei de sugestões e críticas.

Agradeço a todos que de uma forma ou de outra contribuíram para a elaboração deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Este trabalho consiste no desenvolvimento de um sistema de compras pela Internet utilizando interface WWW e o banco de dados orientado a objetos Caché para avaliar a facilidade de desenvolvimento deste tipo de aplicação com a tecnologia de orientação a objetos e verificar a facilidade de integração do Caché com a Internet através do WebLink, ambos da InterSystems Co.

ABSTRACT

This work consist of a development of a purchase system using WWW interface and the objetc oriented database Caché for evaluate the facility of develeop this application type with object oriented technology and observe the facility to integrate Caché and Internet with WebLink, both from InterSystems Co.

1 INTRODUÇÃO

As vendas de produtos caracterizam-se em ser muito diretas. Normalmente o vendedor procura o consumidor ou o consumidor procura o vendedor para efetuar uma transação comercial. Este é com certeza o método mais utilizado ainda hoje, mas existem novas propostas e maneiras de vender um produto como tele-vendas, correio (mala direta) e mais recentemente, por meio da Internet.

E com a atual popularidade da Internet, abrem-se inúmeras oportunidades para aquelas empresas iniciarem a venda de seus produtos pela Internet e assim, começarem a elaborar planos para possibilitar a venda *on-line* de seus produtos. Para isto é necessário que estas empresas tenham as ferramentas corretas que façam com que estas operações sejam eficientes e que agradem o cliente.

Para que isto seja possível é preciso uma maneira de oferecer os produtos na Internet e permitir que o consumidor possa efetuar uma transação comercial e o vendedor tomar conhecimento desta transação. Por isso criaram-se tecnologias que integram a Internet a banco de dados aos quais o vendedor pode ter acesso e atender o pedido do consumidor. Para a implementação deste protótipo será utilizado um banco de dados que ofereça estes recursos, como o Caché que é um banco de dados orientado a objetos.

O banco de dados Caché possui uma linguagem própria de desenvolvimento chamada de *Caché Object Script,* que é derivada da linguagem MUMPS, mas com novos recursos e algumas modificações como a orientação a objetos, é nesta linguagem que o protótipo será desenvolvido.

A integração entre a WWW e o banco de dados Caché ocorre através de um produto chamado de WebLink, fabricado pela InterSystems, que é um servidor de aplicações que interliga o servidor de páginas ao banco de dados Caché, através das *Application Programming Interface* (API) *Netscape Server Application Programming Interface* (NSAPI) ou *Internet Server Application Programming Interface* (ISAPI), as API's dos servidores da Netscape e da Microsoft respectivamente, ou ainda através de *Common Gateway Interface* (CGI) para os servidores que não suportam estas API´s [INT2000].

O protótipo será implementado utilizando interface *World Wide Web* (WWW), que permite a qualquer pessoa conectada à Internet acessar as informações necessárias de uma

maneira organizada [SAV1997]. Para isto serão utilizadas as tecnologias de Internet, como *Hiper Text Markup Language* (HTML), que é uma linguagem de formatação de texto, e JavaScript, que permite um controle mais eficiente das informações para o usuário e que é um recurso hoje disponível nos navegadores de Internet mais utilizados, como o Internet Explorer e o Netscape Navigator.

JavaScript é uma linguagem que veio para solucionar os problemas da linguagem HTML que é completamente estática. JavaScript permite que o desenvolvedor coloque elementos dinâmicos em uma página HTML, possibilitando implementar animações e controle de fluxo de informações de forma que o usuário tenha uma interatividade maior com o página HTML [MCC1997].

Para a especificação do protótipo será utilizada a ferramenta Rational Rose, que utiliza a *Unified Modeling Language* (UML). Esta notação foi criada propondo atender às necessidades de modelagem de sistemas orientados a objeto, abrangendo todas as etapas de análise do sistema, desde os requisitos até o modelo de execução da aplicação [BAR2000]. Para a implementação do protótipo, será utilizada a linguagem nativa do banco de dados chamada de *Caché Object Script* e também suas ferramentas de desenvolvimento como o *Caché Studio*, e o *Object Architect* que são respectivamente um editor de programas e a ferramenta de auxílio a criação das classes no banco de dados. Será também utilizado o editor HTML Macromedia Dreamweaver para criação das páginas.

O protótipo é um modelo de sistema de vendas por Internet que se resume em um cadastro de pedido de compra onde o cliente pode escolher os produtos que lhe convenha e no fim do processo confirmar o pedido e efetuar a transação. O protótipo, também, permitirá que o cliente consulte informações sobre os pedidos anteriores e se for um cliente novo permitir que ele se cadastre no sistema.

1.1 MOTIVAÇÃO

A Internet vem crescendo muito intensamente e com o crescente número de pessoas com acesso a Internet surge a oportunidade para os fornecedores comercializarem seus produtos por esta nova mídia e o comércio eletrônico é tido como uma das tendências emergentes com maior poder potencial de inovação nos processos de negócio nos vários setores econômicos.

1.2 ÁREA

Este trabalho de conclusão de curso abrange as áreas de banco de dados orientado à objetos, desenvolvimento para Internet com HTML e JavaScript e análise orientada a objeto.

1.3 JUSTIFICATIVAS

A importância deste trabalho deve-se ao fato de que as aplicações comerciais estão em um caminho sem volta para a Internet porque fazem com que as vendas sejam facilitadas, evitando que o cliente tenha que se deslocar até o fornecedor permitindo que ele economize tempo para obter os produtos que necessita.

1.4 OBJETIVOS

O objetivo deste trabalho é especificar e implementar um protótipo básico para a venda de produtos pela Internet utilizando banco de dados orientado a objetos.

Como objetivos secundários pretende-se mostrar como se faz a programação orientada a objetos na linguagem *Caché Object Script* e como é feita a integração com a Internet.

1.5 ESTRUTURA DO TRABALHO

O trabalho subdivide-se em cinco grandes partes.

A primeira parte contextualiza o comércio eletrônico e de como ele evoluiu a partir do comércio tradicional com a chegada da Internet.

A segunda parte descreve os bancos de dados orientados a objetos e as diferenças entre os bancos de dados convencionais (relacionais) e o banco de dados Caché.

A terceira parte faz uma breve explanação sobre HTML e JavaScript para o desenvolvimento de produtos para Internet.

A quarta parte mostra a integração do banco de dados Caché com a Internet.

A quinta e última parte demonstra o desenvolvimento do protótipo e sua estrutura.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são mostrados conceitos necessários para melhor entendimento deste trabalho. Este conceitos são comércio, banco de dados, orientação a objetos, banco de dados orientado a objetos, Caché, HTML e JavaScript.

2.1 COMÉRCIO

Comércio é a troca de mercadorias ou serviços através da permutação[1]. É através do comércio que os produtos da indústria e da agricultura são postos em circulação de suas origens para os consumidores, o comércio existe desde a antigüidade quando os grupos mercadores da Mesopotâmia e do Egito praticavam o escambo de mercadorias como tintas, tecidos e metal através do deserto e os Fenícios por vias marítimas, após muito tempo depois disto com a introdução da moeda pelos gregos o comércio começou a ficar mais parecido com o comércio atual onde se trocam mercadorias por moedas e vice-versa ocasionando um crescimento do comércio na época.

Desde este tempo, era necessário a iteração de dois agentes para efetuar uma transação comercial, o vendedor e o comprador que deviam se encontrar fisicamente para obter sucesso na transação, depois com a invenção do correio o encontro entre os agentes já não era mais necessário, mas era um processo lento, inseguro e não confiável. Com a chegada do telefone o problema da lentidão foi resolvido, mas a confiabilidade e insegurança continuaram cada vez maiores, sem contar que o fato de um vendedor telefonar para alguém e oferecer produtos nem sempre é conveniente e pode causar insatisfação ao consumidor.

2.1.1 COMÉRCIO ELETRÔNICO

Com a chegada da Internet introduziu-se o conceito de comércio eletrônico (*e-commerce*) que se baseia em que o consumidor procura o *site* do fornecedor e preenche um pedido de compra, o fornecedor então atende o pedido envia o produto para o consumidor, concretizando a transação.

---

[1] Permutação - é o ato ou efeito de permutar; troca; câmbio.

Para que isto tudo seja possível é necessário que o comprador possa fazer o pedido pela Internet e o fornecedor possa saber o que o comprador deseja adquirir. Para isto criaram-se as tecnologias de Internet que solucionam este problema, que são os bancos de dados e os sistemas de comunicação entre o computador do comprador e o computador do fornecedor, que no caso deste trabalho serão utilizados HTML e seus recursos para a comunicação entre os computadores que farão parte da transação.

2.2 BANCO DE DADOS

A tecnologia de banco de dados é a que permite o armazenamento, recuperação e processamento de dados que descrevem entidades percebidas no mundo real. Os bancos de dados passaram por diversas mudanças até chegarem ao que são hoje.

Segundo [KOR1997] um Sistema de Gerenciamento de Banco de Dados (SGBD) (ou ainda DBMS do inglês *Database Management Systems*) consiste em uma coleção de dados inter-relacionados e um conjunto de programas para acessá-los.

Para [KER1994], SGBD é um sistema para gerenciar dados armazenados de maneira organizada, permitindo todo tipo de operação de manutenção e consulta nesses dados. Essas operações são realizadas através de ferramentas e instruções de alto nível onde o usuário diz o que fazer e o SGBD se encarrega de fazê-lo.

Já [DAT1990] define um SGBD como um sistema de manutenção de dados por computador que tem por objetivo principal manter as informações e disponibilizá-las a seus usuários quando solicitadas.

2.2.1 EVOLUÇÃO DOS BANCOS DE DADOS

Os primeiros sistemas para armazenamento de informações foram os sistemas de gerenciamento de arquivos que realizavam ações simples como inclusões, manutenções e geração de relatórios. Os sistemas construídos com este modelo tinham o tipo de processamento *Batch*, depois nos anos 50 e 60 foram criados produtos para a definição dos

dados, como COBOL[2], que tinha a definição dos dados separada das rotinas que acessavam e atualizavam os dados [KHO1994].

O modelo de dados hierárquico provém uma visão em árvore dos dados, onde cada nó é um registro, e as linhas entre os nós são os relacionamentos entre os registros, estes relacionamentos são de um para muitos, onde um nó pode ter diversos nós filhos.

O modelo de dados em rede é muito parecido com o modelo hierárquico, mas com a possibilidade de qualquer nó estar relacionado com qualquer outro e assim como o modelo de dados hierárquico são um modelo basicamente navegacional, ou seja, o usuário pode começar de um registro e navegar para os nós relacionados.

Em 1970 E. F. Codd introduziu o modelo de dados relacional que se popularizou principalmente nos ambientes cliente/servidor, este modelo se baseia em tabelas, onde cada linha ou tupla de uma tabela corresponde a um registro, uma ocorrência da entidade que se quer armazenar, e as colunas da tabela são os atributos da entidade. O relacional vem da característica que as tabelas são relacionas por atributos em comum, por exemplo em uma tabela de cidades existe um atributo estado que corresponde a um estado na tabela de estados.

No fim da década de 80 a maior parte dos bancos de dados eram baseados no modelo relacional, surgiram então algumas propostas alternativas para banco de dados, mas nunca saíram dos laboratórios, estes eram os bancos de dados semânticos e os bancos de dados de objetos complexos que tinham como objetivo modelar o mundo real da melhor maneira possível.

Dentro deste contexto surgiram os bancos de dados orientados a objetos que ficaram mais fortes durante os anos 90 com a proliferação do conceito de orientação a objetos.

Mais informações dobre a evolução dos bancos de dados podem ser encontrados em [KOR1997], [KHO1994], [KER1994] e [DAT1990].

---

[2] COBOL (COmmon Business Oriented Language) – Linguagem desenvolvida no final dos anos 50 com iniciativa do governo americano, linguagem utilizada para sistemas de negócio.

2.3 ORIENTAÇÃO A OBJETOS

Segundo [KER1994] A orientação a objeto é um paradigma de programação que parte da idéia de que tudo no mundo é objeto. Comparando ao paradigma da programação estruturada, onde dados e procedimento estão separados, a principal diferença é que na orientação a objetos os procedimentos fazem parte de um objeto junto com os dados.

Para [FUR1998] a tecnologia de objetos oferece modularidade de seus elementos podendo-se tomar um subconjunto existente e integrá-lo de maneira diferente em outra parte de um sistema, também apresenta uma correspondência com o mundo real, visualizando objetos de natureza da maneira como são. Estes objetos podem ser utilizados em conjunto ou individualmente para a solução de um problema.

2.3.1 CONCEITOS BÁSICOS

Segundo [FUR1998], o primeiro conceito básico que é necessario saber sobre a orientação a objetos é o conceito de objeto. Um objeto é uma instância (ocorrência) específica de uma classe e é similar a uma entidade da tabela no modelo relacional. Por exemplo, nome e endereço são dados da entidade Pessoa ou do objeto João, João é uma Pessoa.

Outro conceito é o de **mensagem**. Os objetos se comunicam através de mensagens, ou seja pelos métodos deste, um objeto manda uma mensagem para outro através do acionamento de um método específico, o resultado do método é retornado ao objeto chamador.

A **herança** é o conceito de que um objeto pode herdar todos os métodos e atributos de outro objeto, isto significa que é possível criar classes complexas a partir de classes simples sem a repetição de código, este conceito é um dos pontos fortes da orientação a objetos, porque permite a reusabilidade do código que já foi programado.

Polimorfismo é o conceito de que vários objetos respondem a uma mensagem de maneira diferente, por exemplo, existem os objetos motor, lâmpada e televisão, todos possuem o método ligar mas o fazem de maneira diferente.

Classe é a definição de um objeto, ou seja, é uma forma de objeto, representa objetos similares que apresentam a mesma estrutura e comportamento, enquanto uma instância de

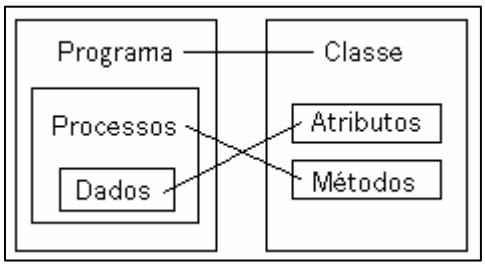
objeto contém informações de um objeto descrito através de uma classe. Por exemplo, uma galinha e um pato são dois objetos com atributos diferentes, mas ambos são da classe Ave.

2.3.2 ANÁLISE ORIENTADA A OBJETOS

Com o avanço das linguagens de programação no sentido da orientação a objetos, a análise não pode ficar de fora e para que se possa utilizar com sucesso os recursos desta tecnologia, nada melhor do que uma análise voltada para este caminho. Para a especificação do protótipo deste trabalho, será utilizada a notação *Unified Modeling Language* (UML) para a análise do problema proposto.

Segundo [FUR1998], o enfoque tradicional de modelagem baseia-se de que sistema é um conjunto de programas que executa processos sobre dados, o enfoque orientado a objetos vê o mundo como uma coleção de objetos que interagem entre si, e objetos são a união de métodos e atributos, processos e dados respectivamente, conforme a figura 1.

Figura 1 - Enfoque baseado em sistema versus enfoque baseado em objeto



Fonte: [FUR1998]

2.3.2.1 UNIFIED MODELING LANGUAGE

Ou Linguagem de Modelagem Unificada (UML), o termo Unificada é pelo fato de que os autores desta notação, Grady Booch e James Rumbaugh, pegaram tudo que existia de bom em modelagem de sistemas e definiram a primeira versão da notação da UML que se chamava Método Unificado. Em 1995 Ivar Jacobson juntou-se à equipe da definição da UML. Com o tempo, muitas empresas importantes reconheceram a importância da UML, que fêz com que esta linguagem se tornasse bem definida, expressiva, poderosa e de uso geral, não proprietária e aberta a todos. Com a aprovação da UML em 1997 pela *Object Management Group* (OMG) a disputa dos métodos Orientados a Objeto tinha chegado ao final [FUR1998].

A UML é uma linguagem de modelagem, não uma metodologia. Metodologias consistem de uma linguagem de modelagem e processos para utilizá-la, a UML não descreve explicitamente este processo, mas em muitos casos, a modelagem é a parte mais importante da metodologia.

A UML é uma linguagem padrão para especificar, visualizar, documentar, e construir sistemas e pode ser utilizada com todos os processos ao longo do ciclo de vida do projeto. Os objetivos da UML são:

a) fornecer aos usuários uma linguagem de modelagem visual expressiva e pronta para uso visando o desenvolvimento de modelos de negócio;

b) fornecer mecanismos de extensibilidade e de especialização para apoiar conceitos essenciais;

c) ser independente de linguagens de programação e processos de desenvolvimento;

d) prover uma base formal para entender a linguagem de modelagem;

e) encorajar o crescimento no número de ferramentas orientadas a objeto no mercado;

f) suportar conceitos de desenvolvimento de nível mais alto tais como colaborações, estrutura de trabalho, padrões e componentes;

g) integrar as melhores práticas.

A UML propõe os seguintes diagramas para a modelagem de diversas partes do sistema:

a) diagrama de classes;

b) diagrama de Casos de uso;

c) diagrama de seqüência;

d) diagrama de colaboração;

e) diagrama de estado;

f) diagrama de atividade;

g) diagrama de Componentes;

h) diagrama de implantação.

Outro conceito interessante da UML é o pacote, que tem como objetivo organizar o sistema colocando todos os diagramas que se relacionam em um pacote, como numa árvore de diretórios. Isto permite que se manipule grandes sistemas sem se confundir numa imensidão de classes a disposição.

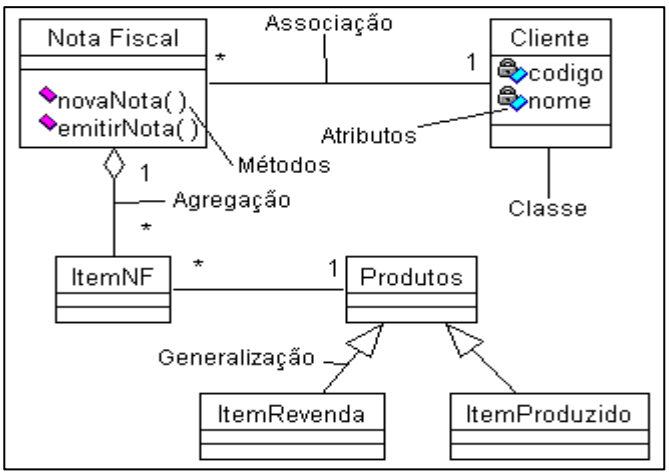
- **Diagrama de classe**

Este é o principal diagrama da UML, é uma estrutura lógica que representa uma coleção de elementos como classes, tipos e seus respectivos conteúdos e declarações. Os quatro principais tipos de relacionamento são:

a) generalização/especificação: indica a herança de um elemento mais geral e um mais especifico (respectivamente, superclasse e subclasse), a subclasse pode conter informação adicional acerca do elemento mais geral;
b) agregação: é utilizada para mostrar relacionamentos todo-parte, como o item de uma nota fiscal;
c) associação: é utilizada para mostrar relacionamento entre classes (uma nota fiscal tem um cliente).
d) dependência: é um relacionamento entre elementos, um independente e outro dependente, onde uma mudança no elemento independente afetará o elemento dependente.

No exemplo da figura 2, podemos visualizar um diagrama de classes demonstrando associações entre as classes “NotaFiscal” com “Cliente” e “ItemNF” com “Produto”, estas associações significam que uma nota fiscal esta relacionada com cliente e um item da nota fiscal possui um produto. Existe também uma agregação de “ItemNF” e “NotaFiscal” representando que a nota fiscal agrega vários itens. E neste caso também existe um exemplo de herança, onde “ItemRevenda” e “ItemProduzido” herdam as características de “Produto”, isto significa que itens de revenda e itens produzidos são produtos.

Figura 2 - Diagrama de classe



- **Diagrama de caso de uso**

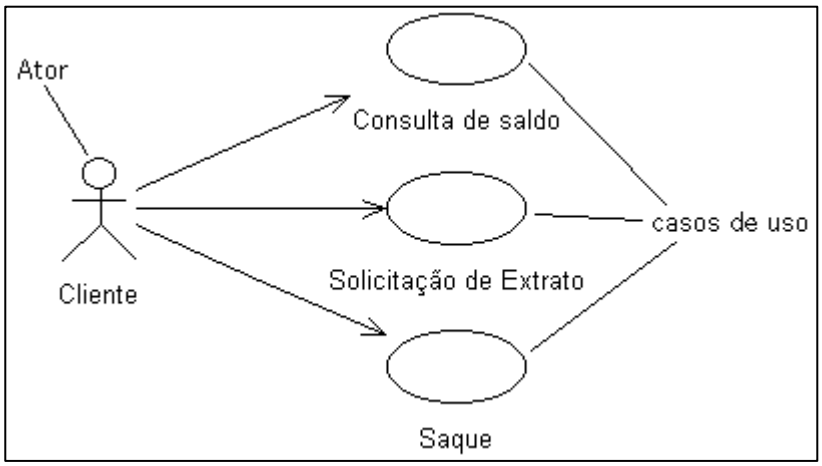
Os casos de uso descrevem como o sistema é visto por atores externos, o ator interage com o sistema podendo ser um usuário, um dispositivo ou outro sistema. Um caso de uso deve ter uma descrição completa do que faz, e deve retornar uma informação em resposta à uma solicitação.

O conceito importante do caso de uso na UML é o ator. O ator representa o mundo externo ao sistema que desempenha papéis relevantes ao sistema, atores típicos são cliente, usuário, gerente, computador, impressora, dispositivo de entrada de dados, etc. É importante ressaltar que o ator representa um papel e não um usuário do sistema.

O ator interage com o sistema através de envio e recebimento de mensagens para os casos de uso, estas comunicações são indicadas por setas que interligam o ator e o caso de uso, a seta é iniciada a partir de quem tomou a iniciativa da comunicação.

Por exemplo o diagrama de casos de usa da figura 3, possui tres casos de uso, “Consulta de saldo”, “Solicitação de Extrato” e “Saque”. Neste exemplo todos os casos de uso são acionados pelo ator “Cliente”.

Figura 3 - Diagrama de caso de uso



2.4 BANCOS DE DADOS ORIENTADOS A OBJETO

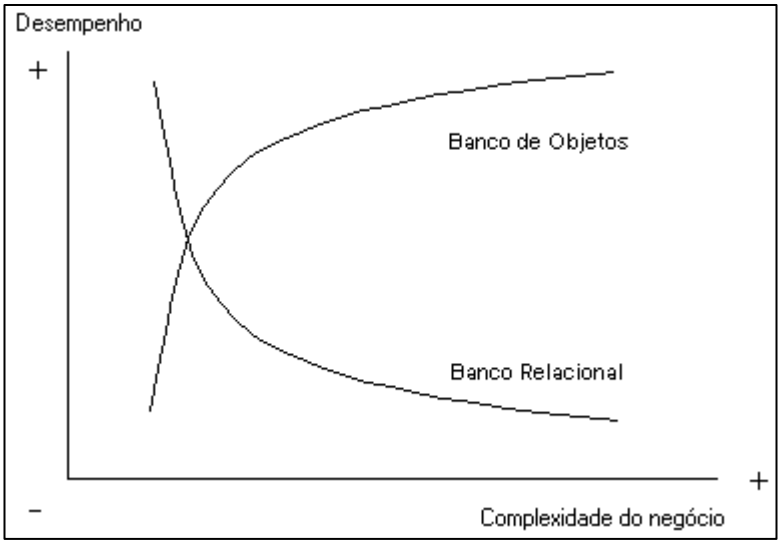
Assim que surgiram as linguagens orientadas a objeto também surgiu a necessidade de bancos de dados que tivessem as características dos bancos de dados relacionais e implementassem o conceito de orientação a objeto.

Segundo [KHO1994] as capacidades dos bancos de dados orientados a objeto são as seguintes:

a) persistência**:** a capacidade de objetos persistirem através de varias sessões da aplicação caracteriza a persistência, ou seja, os objetos persistentes são gravados em disco para que não se percam entre as varais chamadas de um programa;
b) transações**:** são conjuntos de atualizações relacionadas que devem ser executadas todas ou nenhuma, se uma das operações de uma transação falhar todas as operações anteriores são desfeitas;
c) controle de concorrência**:** os bancos de dados orientados a objetos são multiusuários, isto quer dizer que é possível que mais de um processo pode querer acessar um mesmo objeto, por isso é implementado o conceito de concorrência para evitar problemas de integridade no banco de dados;
d) recuperação: um banco de dados deve permitir a recuperação de dados em caso de acontecer alguma falha durante uma transação para que esta falha não seja propagada para o banco de dados, a maneira mais utilizada para implementar esta caraterística é o *log*, onde são gravadas as transações na ordem que acontecem, caso ocorram uma falha, estas são desfeitas uma a uma;
e) filtragem: filtros são utilizados para selecionar conjuntos de objetos dentro de uma coleção. Filtros são expressos em termos de construções declarativas de alto nível que permite ao usuário definir o que ele quer recuperar do banco de dados;
f) versões: um objeto pode ter muitas modificações dentro do banco de dados, o versionamento consiste de ferramentas e construções que automatizam a construção e a organização de versões de um mesmo objeto, permitindo aos usuários acesso tanto ao estado atual quanto a estados anteriores dos objetos;
g) integridade: refere-se a precisão ou a validade dos dados, isto significa que o banco de dados não deve permitir que dados inválidos ou inconsistentes sejam gravados;
h) segurança: é a proteção dos dados para que somente pessoas autorizadas possam acessar ou manipular certos objetos do banco de dados;

i) desempenho: os bancos de dados orientados a objetos tiveram uma reputação de serem ricos em funcionalidade, mas pobres em desempenho. Mas com a atual complexidade dos sistemas, este quadro se reverteu graças a estratégias para melhorar o desempenho, e agora, segundo [MAC1995], conforme figura 4, os bancos de dados relacionais é que sofrem com o desempenho em sistemas com grande complexidade.

Figura 4 - Banco de dados relacional *versus* objeto



Fonte: [FUR1998]

2.5 CACHÉ

Segundo [INT2000] com a crescente complexidade dos sistemas é necessário que os bancos de dados consigam armazenar a complexidade destes sistemas, os bancos de dados relacionais são muito eficientes com seus modelos de tabelas e fáceis de entender, mas a sua simplicidade dificulta a modelagem de problemas mais complexos. Para suprir esta deficiência entram os bancos de dados orientados a objetos que com esta tecnologia se propõe a resolver tais problemas.

Outro fator importante é que as aplicações estão incrementando muito a quantidade de transações, nestas aplicações vêem-se que os bancos de dados relacionais tem sua performance e escalabilidade comprometida, este tipo de problema o banco de dados Caché se destaca, por ter um modelo baseado em objetos executado sobre um banco de dados multi-dimensional e voltado a transações.

2.5.1 ELEMENTOS DO CACHÉ

Conforme [INT1998b],o Caché possui diversas ferramentas para o desenvolvimento de aplicações orientadas a objeto, são elas:

a) *Object Architect* (Arquiteto de Objetos) é uma ferramenta para desenvolver as classes que são utilizadas pela aplicação;
b) *Class Definition Language* (CDL) é a linguagem de definição de classes do Caché, esta definição de classe é um arquivo texto que pode ser criada ou gerada por uma ferramenta e é utilizada pelo Caché para importar e exportar definições de classe;
c) *Cache Object Script* (COS) é a linguagem orientada a objetos do Caché com que é feita a implementação dos métodos das classes;
d) Caché SQL é a implementação da Intersystems do SQL para o Caché. O Caché SQL pode ser utilizado dentro do COS, ou por aplicações externas através do *Caché SQL Server;*
e) *Caché Object Server for Java* é ferramenta que permite que classes Java possam visualizar as classes do Caché. Permite acessar e modificar valores das propriedades e acionar os métodos das classes Caché;
f) *Caché Object Server for ActiveX* permite que qualquer ferramenta de desenvolvimento que possua suporte a *ActiveX* possa manipular as classes do Caché;
g) *Class Dictionary*, que contém as definições de todas as classes de uma aplicação *Caché Objects;*
h) *Class Compiler* que compila as definições de classe e gera o código da aplicação (Rotinas Caché).

2.5.2 CACHÉ OBJECT SCRIPT

Segundo a documentação do Caché [INT1997a] [INT1997b] [INT1998b], o *Caché Object Script* (COS) é a linguagem nativa do Caché, ela é orientada a objetos, suporta completamente o modelo de objetos do Caché. E possui um grande conjunto de comandos, funções e operadores para confecção de programas e métodos de classes. A linguagem também suporta macros, que são elementos que são substituídos por código pré-definido durante a compilação de um programa, por exemplo:

```
#define ISUNIX          ($zv["UNIX")
#define ISWINDOWS       ($zv["Windows")
...
i $$$ISWINDOWS w !,"plataforma Windows"
i $$$ISUNIX w !,"plataforma Unix"
```

Antes da compilação final do programa, as macros são substituidas:

```
i ($zv["Windows") w !,"plataforma Windows"
i ($zv["UNIX") w !,"plataforma Unix"
```

O COS é basicamente formado por comandos, funções e variáveis especiais, O comando é a unidade básica do COS. Os principais comandos do Caché utilizados na implementação são:

a) Do – executa uma rotina do COS ou um método de uma classe;
b) Set – atribui um valor qualquer a uma variável na memória do processo ou fisicamente no disco;
c) Write – comando de saída de informações que exibe resultados na tela do computador, impressoras, arquivos ou nas conexões entre o servidor WWW e o navegador do usuário;
d) Quit – finaliza a execução do método ou do programa/sub-programa atualmente na pilha de execução e volta para o programa/método que o chamou;
e) If – comando de condição do Caché;
f) For – repetição que define os *Loop's* do COS;
g) Kill – este comando "mata" variáveis da memória ou do disco;
h) New - coloca uma variável numa pilha em memória que não pode ser mais acessada até que seja encontrado um **Quit** quando volta com o valor original, ela pode ser definida novamente no bloco de código atual e é retirada da memória no final.

As principais funções utilizadas no protótipo são:

a) $Data – esta função retorna informações a respeito da existência de variáveis;
b) $Extract – retorna uma subcadeia de uma cadeia de caracteres informada;
c) $Fnumber – retorna o valor formatado de um número;
d) $Get – retorna o conteúdo de uma variável ou nulo se esta não existir;
e) $Piece – retorna o n-ésimo elemento de uma cadeia com um separador específico;
f) $Order – retorna o próximo índice de uma variável indexada.
g) A única variável especial utilizada no protótipo é **$Horolog** que contém a data e a hora atual do sistema separado por vírgula.

2.5.3 ACESSANDO UM OBJETO

No Caché os objetos persistentes podem estar em duas situações, gravados ou carregados, gravados significa que o objeto está gravado no banco e carregado significa que ele esta na memória e pode ser manipulado. Segundo [INT1998a], o objeto pode ser referenciado de duas maneiras dependendo da situação, pelo OREF (*object* reference) ou pelo OID (*object identifier*).

Um OREF aponta para um objeto na memória, toda vez que um objeto é carregado do banco é assumido um OREF diferente.

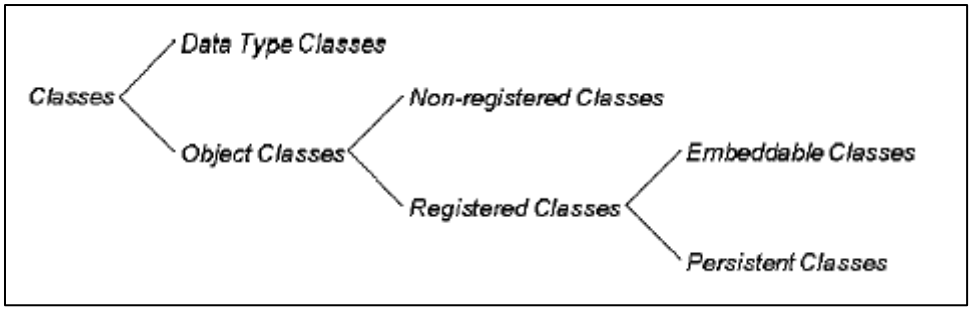
Um OID aponta para um objeto que está no banco, o OID é único para cada objeto de uma classe e nunca muda, é a partir o OID que se referencia a objetos que estão gravados no banco.

Para objetos que possuem associações com outros objetos, os atributos do objeto que está carregado possuem o OREF do objeto associado. Quando este objeto é gravado no banco, os valores destes atributos são substituídos pelos OIDs dos objetos associados.

2.5.4 TIPOS DE CLASSE

Conforme [INT1998a], no Caché existe uma hierarquia de objetos suportados pelo Caché com os seguintes tipos de classe conforme a figura 5.

Figura 5 - Tipos de Classes do Caché



Fonte: [INT1998a]

Classes do tipo *Data Type Classes* são classes para representar tipos de dados do Caché, como inteiros, datas e *strings*, enquanto classes *Object Classes* são utilizadas para outros fins. A maioria das outras classes derivam da classe de sistema *%RegisteredObject* que

disponibiliza muitas ações básicas de um objeto e são chamadas *Registered Classes*. Classes que não derivam de *%RegisteredObject* são chamadas de *Non-registered Classes*, suportam apenas os recursos mais básicos de objetos e é necessário implementar outras facilidades para utilizá-la na aplicação.

a) *Data Type Classes* – este tipo de classe define e controla valores literais, diferentemente de *Object Classes,* não possui identificador e não pode ser instanciado explicitamente. Portanto, eles só existem como atributos de objetos;
b) *Object Classes* – classes deste tipo são utilizadas para implementar a aplicação, existem dois tipos de *Object Classes*, as *Registered Objects* e *Non-registered Objects*;
c) *Registered Objects* – estas classes são derivadas de *%RegisteredObject* que possui uma série de facilidades que incluem métodos para alocar e desalocar memória para seus atributos e controlar seu OREF;
d) *Non-registered Objects* – estas classes não são derivadas de *%RegisteredObject* e não são *Data Type*, isso significa que o desenvolvedor precisa implementar toda a lógica para o objeto poder ser manipulado, como os métodos de alocar e desalocar memória, este tipo de classe não é muito utilizado;
e) *Persistent Class* – classes deste tipo são derivadas de *%Persistent* estas classes são *Registered Classes* que possuem todo o mecanismo de persistência implementado e são as classes que serão armazenadas no banco;
f) *Embeddable Classes* – estas são classes embutidas, estas classes podem existir independentemente na memória, mas quando são gravadas, só podem existir embutidas em outra classe, ou seja, como um atributo de uma classe persistente. Classes embutidas são diferentes gravadas ou carregadas, na memória são objetos comuns com OREF e atributos, mas quando o objeto que as contém é gravado, estas classes são gravadas junto e não existem separadas.

2.5.5 HERANÇA

De acordo com [INT1998a], o modelo de objetos do Caché permite que o desenvolvedor crie classes com herança simples e herança múltipla. Uma classe derivada herda todas as especificações de sua superclasse, incluindo propriedades, métodos e parâmetros. Exceto quando uma classe é definida como “final”, então esta classe não pode ser

derivada. Uma classe pode sobrescrever (mas não pode apagar) os componentes herdados, exceto quando o componente for definido como “final”.

Quando uma classe tem diversas superclasses, ela herda os membros de todas as superclasses, se existirem membros com o mesmo nome em mais de uma superclasse é herdado o membro da última superclasse com este membro.

2.5.6 ESPECIFICAÇÃO DE CLASSES

Uma definição de classe determina o que é uma classe e o que ela possui. Conforme [INT1998a], uma definição de classe to tipo *Object Classes* inclui parâmetros, métodos e propriedades, enquanto classes *Data Type* só possuem métodos e parâmetros.

Para definir uma classe no Caché é utilizado o *Object Architect* ou utilizando CDL. Os elementos para uma definição de classe são:

- Um nome único de classe;
- *Class Keywords* – um conjunto de modificadores que atuam na definição da classe, podem controlar tanto a classe toda ou um membro da classe, os seguintes membros são definidos utilizando *Class Keywords:*
- *Class Parameters* – são parâmetros de classe que servem para controlar o comportamento de uma classe durante a compilação da classe, usado normalmente por métodos geradores de código;
- *Class Properties* – são as propriedades ou atributos da classe, os dados associados a uma instância da classe;
- *Class Methods* – são os métodos da classe;
- *Class Queries* – são consultas SQL para filtrar objetos que satisfaçam a um critério;
- *Class Indexes* - são os índices que são estruturas para otimizar o acesso aos dados.

2.5.6.1 PARÂMETROS

De acordo com [INT1998a], os parâmetros podem também ser chamados de propriedades de classe, isto significa que é um valor fixo para todos os objetos da classe. Comparando com Java ou C++ seria uma propriedade estática. Uma subclasse pode substituir um parâmetro se ele não for “final” e pode acrescentar novos parâmetros.

Parâmetros podem ser utilizados para qualquer fim, normalmente são utilizados por métodos geradores durante a compilação para definir o comportamento deste método em *run-time*.

Classes *Data Type* utilizam parâmetros para fornecer diversas funcionalidades como validação ou conversão de valores, por exemplo, o *Data Type %Integer* possui um parâmetro chamado **MAXVAL** que restringe o valor do atributo definido como *%Integer*. Este parâmetro é utilizado pelo método *IsValidDT()* da classe *%Integer* para fazer a validação do atributo na classe que contém o atributo.

2.5.6.2 MÉTODOS

Métodos são as operações do objeto que podem ser invocados. Segundo [INT1998a], todo método possui um nome único na classe, uma lista formal de argumentos e o código do método.

Todo método deve especificar um tipo de valor de retorno que pode ser um *Data Type*, ou outra classe registrada do Caché. Classes podem utilizar o *Data Type %Status* para incorporar tratamento de erros no método.

Os métodos podem receber qualquer quantidade de argumentos, os argumentos são especificados na definição do método que também deve definir o tipo de cada argumento. Na definição podem ser definidos valores *default* para o argumento e se este argumento é passado por valor ou por referencia.

Na maior parte do tempo os métodos se referenciam a métodos de um objeto (instância da classe). Métodos de instância são métodos que são invocados para uma instância específica da classe e para isso o objeto deve estar carregado na memória. Métodos de classe são métodos que podem agir sobre todas as instâncias da classe.

2.5.6.3 PROPRIEDADES

Propriedades definem o estado de um objeto. Existem dois tipos de propriedades: **atributos** que contém valores e **relacionamentos** que contém associações entre objetos. Segundo com [INT1998a], o Caché, atualmente suporta somente propriedades do tipo **atributo**.

Muitas linguagens como Java e C++ não possuem propriedades na realidade, mas possuem atributos privados que são expostos por métodos públicos de acesso. No Caché existe um modelo de propriedades onde:

a) são valores literais, referencias a objetos, objetos embutidos, etc.;
b) possuem, automaticamente, métodos para acesso que suportam validação;
c) podem invocar métodos de conversão de dados durante a leitura ou gravação dos dados, automaticamente;
d) carregam, automaticamente, objetos embutidos e persistentes para a memória.

As propriedades podem ser declaradas públicas ou privadas, se elas forem privadas, só podem ser acessadas pela classe em que foram definidas ou por suas subclasses; se forem públicas podem ser acessadas de qualquer lugar. Se uma propriedade for declarada privada, esta propriedade não é considerada na projeção SQL da classe, ou seja, não é possível fazer comandos SQL que utilize esta propriedade.

Para acessar o conteúdo de propriedades na memória o Caché possui a *Dot Sintax* e para cada propriedade, o Caché gera, automaticamente, um método *Set()* e *Get()* para acessar este conteúdo, por exemplo:

```
Set OREF.atributo=”valor”
```

Invoca:

```
Do OREF.atributoSet(“valor”)
```

Enquanto:

```
Write OREF.atributo
```

Invoca:

```
Write OREF.atributoGet()
```

Para acessar as propriedades de dentro dos métodos de instância, existem três sintaxes:

a) **`Write ..atributo`** - Esta sintaxe chama o atributo do objeto acessado pelo método deste objeto;

b) **`Write %this.atributo`** - Funciona de maneira idêntica a sintaxe anterior, mas neste caso não é possível acessar propriedades privadas;

c) **`Write i%atributo`** - Esta sintaxe é utilizada para acessar o conteúdo da propriedade na memória sem utilizar os métodos *Get()* e *Set()* da propriedade.

De acordo com [INT1998a], existem ainda as classes persistentes, que alem dos elementos de classes *Object Classes* possuem ainda os seguintes elementos:

a) Queries

Queries no Caché são métodos de classe especiais que retornam uma consulta do tipo *%ResultSet*, as *queries* podem ser escritas tanto com COS ou com SQL, *queries* criadas a partir do arquiteto de objetos utilizam SQL.

Na definição de *queries* é necessário a definição das colunas da linha SQL, esta linha pode conter o ID do objeto ou não. Na definição das colunas é possível também definir nomes para os títulos das colunas que devem aparecer para o SQL e os tipos de dados das colunas.

b) Índices

Índices são utilizados para otimizar a performance durante a consulta de dados em objetos.

Nos índices são incluídos, automaticamente, os objetos das subclasses, isto significa que a classe pessoa que tem um índice por nome e a classe estudante que deriva de pessoa, as instâncias de pessoa e estudante estão no índice.

Os índices podem ser formado por um ou mais atributos da classe, e podem conter dados adicionais que são freqüentemente acessados para aumentar a performance de consultas SQL baseadas nos atributos que estão no índice.

c) *Triggers* e eventos

Triggers são segmentos de código que são acionados na execução de comandos SQL de manipulação (INSERT, UPDATE e DELETE). Estes códigos podem ser executados antes ou depois do comando SQL dependendo da definição da *Trigger*.

Eventos do *Caché Objects* são métodos executados por classes do sistema, para permitir processamento extra durante eventos específicos.

Os eventos para classes derivadas de *%RegisteredObject* são:

a) *%OnNew()* – é executado por *%New()* quando o objeto é criado e alocado na memória;
b) *%OnClose()* – é executado por *%Close()* quando o objeto é fechado e liberado da memória.

Os eventos para classes derivadas de *%Persistent* são:

a) *%OnOpen()* – é executado por *%Open()* quando o objeto é carregado do banco para a memória;
b) *%OnBeforeSave()* – é executado por *%Save()* antes do objeto ser gravado no banco;
c) *%OnAfterSave()* – é executado por *%Save()* após o objeto ter sido gravado no banco;
d) *%OnValidateObject()* – é executado por *%ValidateObject()* quando o objeto é validado para gravação;
e) *%OnDelete()* – é executado por *%Delete* quando o objeto é excluído do banco.

Por exemplo para impedir que um objeto seja apagado, ele deve possuir o seguinte código:

```
...
Write “Este objeto não pode ser excluído!”
Quit $$$ERROR(1)
```

2.5.7 DATA TYPES

Conforme [INT1998a], o Caché suporta diversos tipos de dados para atributos e variáveis através de *Data Types*. Cada *Data Type*, que também é conhecido como *Abstract Data Types* (ADT), é uma classe que controla as ações dos atributos nos objetos, como validações, conversões e a maneira como são expostos para tabelas relacionais.

Os *Data Types* possuem os seguintes recursos:

a) Eles gerenciam a conversão entre os dados literais armazenados no banco, dados lógicos na memória e *Display Formats*;

b) Eles permitem mecanismos de validação de dados literais que o desenvolvedor pode estender e customizar.

Com relação à classes, os *Data Types* diferem porque não contém propriedades, suportam um conjunto específico de métodos e não podem ser instanciados independentemente.

A seguir, são relacionados as principais classes *Data Type* fornecidos com o Caché:

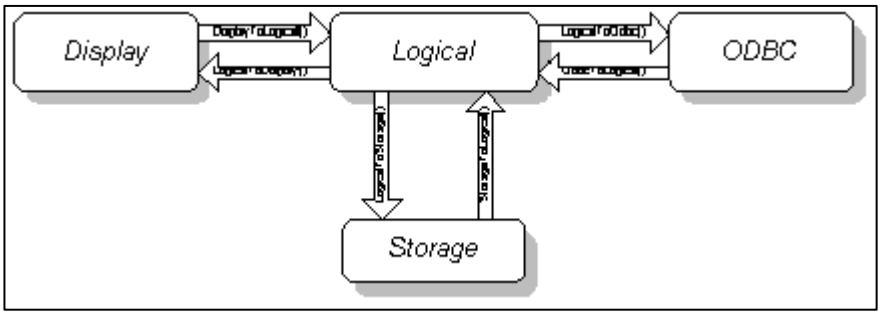
a) *%Binary* – representa dados binários;
b) *%Boolean* – representa um valor lógico;
c) *%Currency* – representa um valor monetário;
d) *%Date* – representa uma data;
e) *%Float* – representa um valor de ponto flutuante;
f) *%Integer* – representa um valor inteiro;
g) *%List* – representa dados em formato de lista;
h) *%Name* – representa um nome no formato “Sobrenome, Primeiro nome”;
i) *%Numeric* – representa um valor numérico com precisão variável;
j) *%Status* – representa um código de erro;
k) *%String* – representa uma cadeia de caracteres;
l) *%Time* – representa uma hora;
m) *%TimeStamp* – representa uma data e hora.

Quando Caché esta manipulando dados, este pode estar em quatro formatos diferentes:

a) *Display* – formato em que os dados são mostrados para o usuário;
b) *Logical* – formato em que os dados são manipulados na memória;
c) *Storage* – formato em que os dados são gravados no disco;
d) *ODBC* – formato em que os dados são apresentados via ODBC/SQL.

Seundo [INT1998a] os dados são úteis apenas se são acessíveis, por isso os métodos são parte importante da funcionalidade dos *Data Types*, eles servem de base para a criação de um *Data Type* específico, os métodos são: *DisplayToLogical(), LogicalToDisplay(), LogicalToOdbc(), LogicalToStorage(), OdbcToLogical(), StorageToLogical()*. Estes métodos são utilizados para conversão entre os diferentes formatos que a informação pode ter, conforme figura 6.

Figura 6 - Formato de dados e métodos de conversão



Fonte: [INT1998a]

Por convenção atributos de dados suportam valores enumerados (múltipla escolha). Este recurso é implementado por dois parâmetros dos *Data Types*: ***VALUELIST*** e ***DISPLAYLIST***.

Por exemplo, na definição do atributo “atualizado” no exemplo seguinte, estão definidos os parâmetros ***VALUELIST*** e ***DISPLAYLIST***, que restringem o conteúdo do atributo “atualizado” e pode-se utilizar o método *LogicalToDisplay()* para mostrar “Não” ou “Sim” dependendo do valor de “atualizado”.

```
...
attribute atualizado { type = %String( DISPLAYLIST=”,Não,Sim”,
VALUELIST=”,0,1”); ... }
...
```

2.5.8 MANIPULAÇÃO DE OBJETOS

Este item mostra como fazer a manipulação de objetos do Caché, executar métodos, gerenciar classes através dos métodos, criar novos objetos, fechar, abrir, salvar e apagar objetos e executar *Queries.*

2.5.8.1 EXECUTANDO MÉTODOS

A criação e manipulação de objetos é feita através de métodos. Segundo [INT1998a] o Caché suporta dois tipos de métodos: o método de instância que atua sobre um objeto específico que está carregado na memória e o método de classe que pode atua sobre todos os objetos de uma classe.

Para executar métodos de instância utiliza-se a seguinte sintaxe:

```
Do OREF.NomeDoMetodo(ListaDeArgumentos)
```

```
Set Retorno=OREF.NomeDoMetodo()
```

Para executar métodos de classe utiliza-se a seguinte sintaxe:

```
Do ##class(NomeDaClasse).NomeDoMetodo(ListaDeArgumentos)
Set Retorno=##class(NomeDaClasse).NomeDoMetodo()
```

Argumentos para métodos podem ser passados por valor e por referência conforme a seguinte sintaxe:

```
Do OREF.Metodo(PorValor)
Do OREF.Método(.PorReferencia)
```

2.5.8.2 MÉTODOS PARA GERENCIAMENTO DE CLASSES

Todos as classes derivadas de *%RegisteredObject* herdam dois métodos para a criação e fechamento de objetos que respectivamente alocam e liberam memória do objeto, estes métodos são:

a) *%New()* – este método cria um OREF e aloca memória para o objeto;

b) *%Close()* – este método fecha um objeto e libera a memória.

Sintaxe:

```
Set OREF=##class(NomeDaClasse).%New()
Do OREF.%Close()
```

Para as classes persistentes derivadas de *%Persistent* existem ainda os métodos para abrir um objeto no disco, salvar e apagar:

a) *%Open()* – este método de classe recebe como parâmetro um OID e retorna um OREF com o objeto carregado;

b) *%Save()* – este método salva o objeto que está na memória;

c) *%Delete()* – este método apaga um objeto no disco.

Sintaxe:

```
Set OREF=##class(NomeDaClasse).%Open(OID)
Do OREF.%Save()
Do ##class(NomeDaClasse).%Delete(OID)
```

2.5.8.3 EXECUTANDO *QUERIES*

Queries são métodos que tem como retorno um objeto do tipo *%ResultSet* e existem duas maneiras de visualizar o resultado de uma *Query*. A primeira delas é executar o método ***RunQuery*** que executa a Q*uery* e mostra o resultado de uma maneira muito simples:

```
Do ##class(%ResultSet).RunQuery(“Classe”,”Query”,”Arg1”...)
```

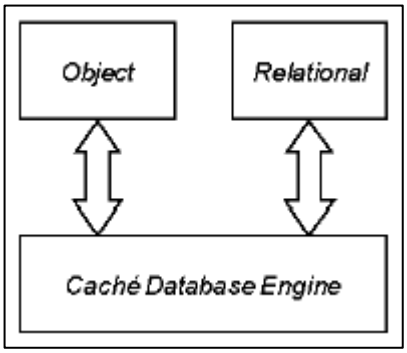
Outra maneira, mais complexa mas com mais recursos são através dos outros métodos de *%ResultSet* que permitem que o desenvolvedor mostre o resultado da *Query* como quiser. Abaixo um exemplo utilizando um *%ResultSet* de maneira mais complexa:

```
NEW col,rset,sc,colunas
SET rset=##class(%ResultSet).%New()
SET rset.ClassName="Classe"
SET rset.QueryName="Query"
SET colunas=rset.GetColumnCount()
SET sc=rset.Execute("Arg1")
IF $$$ISERR(sc) ; Falha na execução da Query
FOR QUIT:'rset.Next() DO
. FOR col=1:1:colunas DO
. . WRITE rset.GetColumnName(col),": "
. . WRITE rset.GetData(col),!
DO rset.Close() ; Fecha a execução da Query
DO rset.%Close() ; Fecha o objeto rset
```

2.5.9 PROJEÇÃO SQL

Segundo [INT1998a], o Caché permite que se tenha um visão relacional dos objetos persistentes, o *Caché SQL Server* é a ferramenta que permite isto. A projeção SQL é possível graças a *Caché Unified Data Architecture* (Arquitetura de Dados Unificada Caché) mostrada na figura 7 e ao *Class Dictionary*. Desta maneira o desenvolvedor pode manipular objetos de forma relacional no *Caché Database Engine* com alta performance.

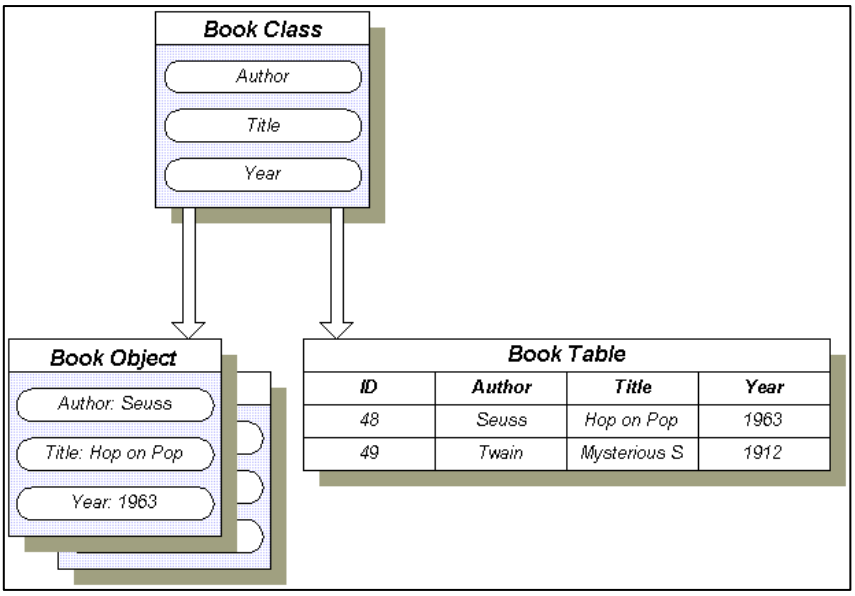
Figura 7 - Caché Unified Data Architecture



Fonte: [INT1998a]

Normalmente, a projeção SQL vê uma classe como uma tabela, e cada instância da classe como uma linha na tabela e cada atributo da classe como uma coluna. Como por exemplo a figura 8 mostra a projeção SQL de dois objetos da classe "Book".

Figura 8 - Projeção SQL de uma definição de classe



Como o SQL não suporta a herança, a projeção SQL do Caché produz uma tabela das classes derivadas com as colunas dos atributos da própria classe mais os atributos herdados. Por exemplo, uma classe “estudante” que é uma subclasse de “pessoa”, produz uma tabela “estudante” com todos os atributos de pessoa e todos os atributos de estudante como colunas.

As classes no Caché podem ter atributos com tipos que não são literais, estes atributos são projetados para SQL de acordo com a tabela 1:

Tabela 1 - Projeção SQL dos elementos OO

Conceito de objeto	Conceito relacional
OID	Campo identidade
Atributo *Data Type*	Campo normal
Atributo de referencia	Campo de referencia
Objeto embutido	Campos normais
Atributo lista	Campo lista
Atributo *Array*	Tabela filha

Atributo *Stream*	*BLOBs*
Índice	Índice
Método de classe	*Stored Procedure*

2.5.10 SQL EMBUTIDO

Para simplificar o processo de desenvolvimento de aplicações de banco de dados, o Caché suporta SQL embutido em códigos de métodos. Esta linguagem de consulta embutida permite que se combine a tecnologia de objetos do Caché com a flexibilidade do SQL.

Segundo [INT1998a], para utilizar o SQL embutido o COS oferece a palavra chave **&sql**. A sintaxe é a seguinte:

```
METHOD ... {
    CODE =
    {
        ...
        &sql(SELECT * into :T() FROM table )
        ...
    }
}
```

Para saber se o comando SQL executou com sucesso ou com erro, o SQL embutido gera uma variável que contém o código do erro. O conteúdo desta variável pode ser:

SQLCODE=0	Comando executado com sucesso
SQLCODE=100	Comando executado com sucesso, mas não existem mais dados
SQLCODE<0	Ocorreu um erro na execução do comando

O SQL fornece o recurso de cursores para obter dados de comandos SQL que resultam em mais de uma linha de informação, para manipulação de cursores existem os comandos SQL mostrados no exemplo abaixo:

```
; declaração de cursor
;
&sql( DECLARE NomeCur CURSOR FOR SELECT col1, col2 FROM Table )
;
...
; abrindo o cursor
```

```
;
&sql( OPEN NomeCur )
;
...
; pegando dados de uma linha do SELECT
;
&sql( FETCH NomeCur INTO :col1, :col2)
;
...
; fechando o cursor
;
&sql( CLOSE NomeCur )
```

2.5.11 HTML EMBUTIDO

Para a criação de aplicações baseadas em Web, conforme [INT1998a], o COS permite que se coloque HTML embutido no código dos métodos. Para isto o COS oferece a palavra chave **&HTML** e a sintaxe é a seguinte:

```
METHOD ... {
    CODE =
    {
        ...
        &HTML<<HTML><HEAD><TITLE>Ola mundo</TITLE>
        ...
        </HTML>>
        ...
    }
}
```

Esta sintaxe é simples e poderosa, pois permite que o desenvolvedor possa colocar no HTML embutido qualquer recurso do HTML, como, por exemplo, JavaScript.

Para incluir conteúdo dinâmico no HTML embutido são utilizadas duas sintaxes, uma para execução de comandos COS e outra para inclusão do resultado de expressões. Estas sintaxes são as seguintes:

Para executar comandos COS:

```
...
&HTML<
  ...
  Algo HTML<BR> <% Do OREF.Método() %>
  Mais HTML
  ...
>
```

Para inclusão do resultado de expressões COS:

```
...
&HTML<
  ...
```

```
    A Resposta é: <%=OREF.Atributo%>
    Mais HTML
    ...
>
```

2.6 HTML

Hiper Text Markup Language (HTML) é a linguagem utilizada para formatação dos documentos da *World Wide Web* (WWW) que tem por objetivo a distribuição de informações através da rede Internet, a WWW é constituída de um conjunto de protocolos que tem por objetivo padronizar a distribuição destas informações que podem ser de qualquer tipo, texto, imagens, áudio, vídeo, etc.

A HTML funciona através de *Tag´s* ou marcas que definem como o texto será visualizado pelo usuário leitor. Derivada da *Structed Generalized Markup Language* (SGML) que foi criada originalmente pela *International Standards Organizations* (ISO) para documentação de equipamentos e foi adotada pelo departamento de defesa dos Estados Unidos para a padronização de documentos técnicos criados pelos seus fornecedores e prestadores de serviços [SAV1997].

Atualmente quem cuida da padronização dos protocolos utilizados pela WWW é a *World Wide Web Consortium* (W3C) (http://www.w3c.org) que tem como objetivo padronizar e definir as novas versões dos protocolos utilizados na WWW inclusive o HTML.

Os recursos mais utilizados do HTML neste protótipo são os formulários pelo fato de ser um sistema com grande interatividade é necessário que haja comunicação entre o sistema e o usuário, para isto os formulários possuem uma serie de *Tag's* para a criação de campos de diversos tipos e um protocolo de comunicação com o servidor WWW que ativa o sistema desenvolvido no protótipo.

2.7 JAVASCRIPT

JavaScript é uma linguagem derivada do Java desenvolvida pela Netscape que é utilizada para criar documentos dinâmicos em documentos HTML. É uma linguagem interpretada, embutida no código do documento HTML e é executada no computador do usuário. Desta maneira é possível que o próprio cliente realize algumas tarefas sem que se necessite que seja feita nova requisição da página atualizada para o servidor WWW. Assim

obtêm-se uma resposta mais rápida para ações que necessitem de respostas simples como validação de datas em campos de formulários e alteração automática do conteúdo ou características do documento HTML.

Segundo [RIT1996] JavaScript é fácil pelo fato de ser uma linguagem derivada do Java, o que torna seu aprendizado rápido para as pessoas que já tiveram contato com esta linguagem, JavaScript é dinâmico e a vantagem do JavaScript é a sua capacidade de tratar eventos com elegância ao interagir com os elementos do HTML como as entradas de dados, botões, imagens e *links*, conforme exemplo 1, JavaScript é baseado em objetos, embora não totalmente, isto significa que é uma linguagem moderna e permite a comunicação com miniaplicativos Java que podem estar referenciados na pagina HTML.

Exemplo 1 - Fonte de HTML com JavaScript

```
<HTML><HEAD>
<TITLE>Alo Mundo</TITLE>
</HEAD>
<SCRIPT LANGUAGE=”JavaScript”>
function aloMundo() {
   alert(“Alo Mundo”);
}
</SCRIPT>
<BODY>
<FORM>
<INPUT TYPE=”button” VALUE=”Diga Ola” OnClick=”aloMundo()”>
</FORM>
</BODY>
</HTML>
```

3 DESENVOLVIMENTO DO TRABALHO

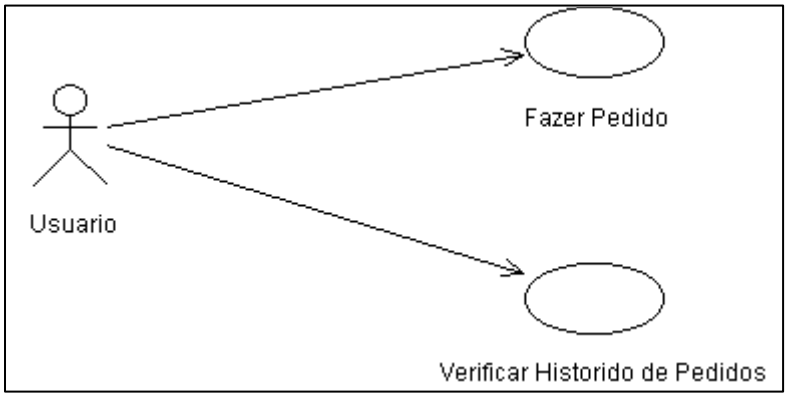
Neste capítulo são descritas todas as etapas e ferramentas utilizadas na especificação e implementação.

3.1 APRESENTAÇÃO DA ESPECIFICAÇÃO

O protótipo é um sistema de loja virtual onde o cliente deve ter acesso restrito. O sistema deve permitir que o cliente entre no site e já possa preencher o pedido ou então consultar o seu histórico de pedidos e situação dos mesmos. No final da compra o sistema deve pedir a confirmação do pedido e dos dados cadastrais do cliente e informar-lhe o número do pedido gerado no sistema.

O sistema, possui dois casos de uso (Figura 9) e dois diagramas de classe, um para a definição dos objetos de negócio (Figura 10) que são persistentes e um diagrama para as classes de interface com o usuário (Figura 11).

Figura 9 - Diagrama de casos de uso do protótipo

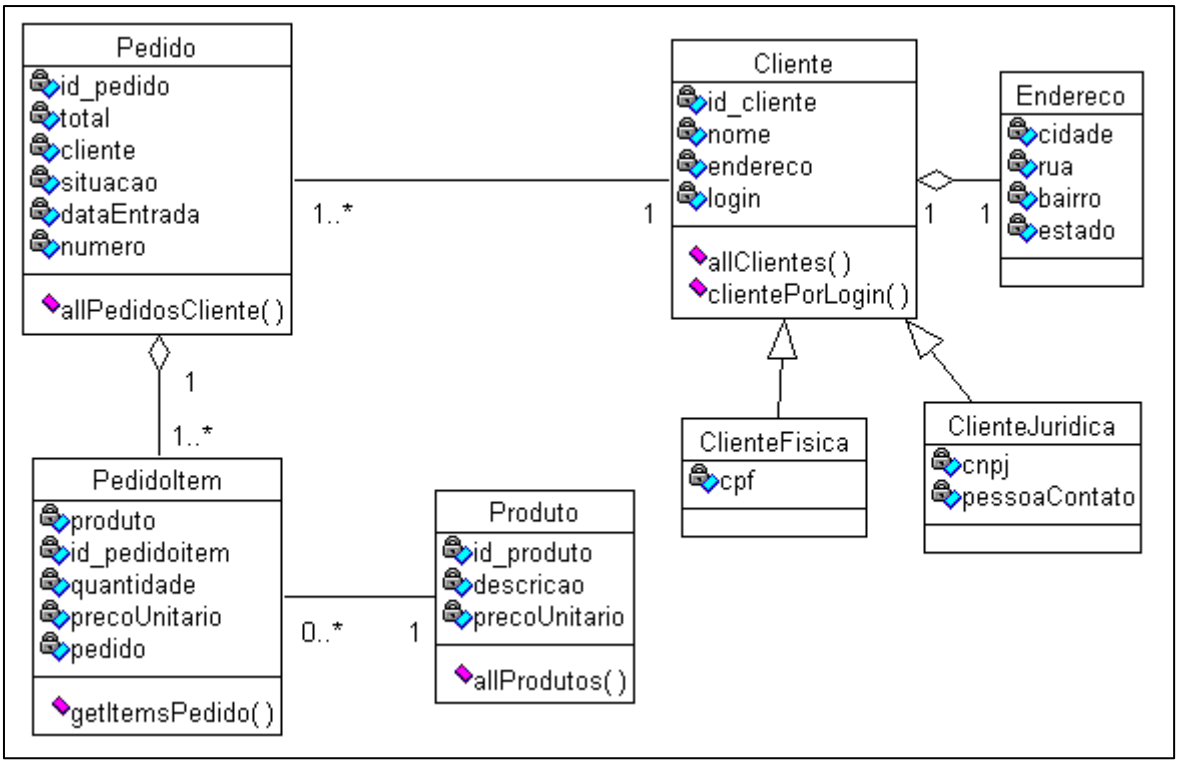


O caso de uso “Fazer Pedido” descreve o processo de fazer o pedido, que se resume em o cliente informar ao sistema a quantidade dos produtos que ele quer comprar e confirmar o pedido e suas informações cadastrais.

A descrição do caso de uso “Verificar Histórico de Pedidos” permite ao cliente consultar todos os pedidos realizados por ele, e os produtos que formam um pedido específico.

O diagrama de classes mostrado na figura 10 mostra as classes que são necessárias para o sistema de pedidos em si, este diagrama possui as seguintes classes:

Figura 10 - Diagrama de classes de negócio



a) Pedido – esta classe é responsável por armazenar e controlar os pedidos que são incluídos pela interface WWW. Esta classe possui a *Query* “allPedidosCliente” que retorna todos os pedidos de um cliente;
b) Produto – esta classe controla e armazena os produtos disponíveis para fazer pedidos pela Internet. esta classe possui a *Query* “allProdutos” que retorna todos os produtos cadastrados;
c) PedidoItem – esta classe é responsável pelo relacionamento entre a classe Pedido e a classe Produto. Esta classe possui a *Query* “getItemsPedido” que retorna os itens de um pedido;
d) Cliente – é a classe que contém dados genéricos sobre clientes. Esta classe possui a *Query* “allClientes” que retorna os clientes cadastrados e um método “clientePorLogin” que retorna o cliente a partir do nome de usuario;
e) ClienteFisica – é a classe especializada da classe Cliente que trata de pessoas físicas;
f) ClienteJuridica – é a classe especializada da classe Cliente que trata de pessoas jurídicas;

g) Endereço – é uma classe embutida que agrupa dados de endereço.

O diagrama mostrado na figura 11 mostra as classes responsáveis pela interface com o usuário na WWW. Todas as classes derivadas da classe **HTMLCache** devem implementar os métodos **getHTML** e **post.**

O método **getHTML,** gera código HTML que é enviado ao usuário e o método **post,** trata o retorno das informações do formulário HTML.

As classes que formam o diagrama da figura 11 são:

a) HTMLCache

Esta a classe que será herdada por todas as outras classes que fazem interface com o usuário, estas classes devem implementar os métodos **getHTML** e **post.**

O método **getHTML,** gera código HTML que é enviado ao usuário. O método **post,** trata o retorno das informações do formulário HTML, chamando o método **getHTML** de uma classe que será o resultado da ação do usuário. E o método **getForm,** que gera o código HTML necessário para o WebLink invocar o método **post** da classe correta.

b) HTMLCarrinho

Esta classe tem como objetivo gerar o código HTML para o usuário consultar os produtos já selecionados. Em determinado momento o método **getHTML** invoca o método **getHTMLLista**, que código HTML para os produtos que estão selecionados.

c) HTMLCliente

Esta classe é responsavel de cadastrar novos clientes. O método **getHTML** gera o formulário HTML onde o usuário irá informar os seu dados pessoais. Quando o usuário clicar no botão “confirma”, o método **post** invoca o método **salvarCliente** que cria uma nova instância de **Cliente** com os dados do formulário.

d) HTMLCompra

Esta classe é a classe encarregada de gerenciar a seleção dos produtos do pedido. Esta classe tem os métodos **getHTMLLista** e **getScript** que são chamados por **getHTML,** que geram a paginá com todos os produtos retornados pela *Query* “allProdutos” da classe **Produto**.

Se o cliente clicar no botão “Todos”, ou em “Adicionar”, o método **post** irá chamar o método **adicionar,** que irá colocar os produtos selecionados no carrinho de compras e chamar o método **getHTML.**

Se o cliente clicar o botão “Consulta” o método **post** irá chamar o método **getHTML** da classe **HTMLConsulta** ou **HTMLLogin** se o cliente ainda não se identificou para o sistema.

Se o cliente clicar no botão “encerra” e o cliente ainda não se identificou o método **post** irá chamar o método **getHTML** da classe **HTMLTipoCliente**, senão o método **getHTML** da classe **HTMLEncerrar** é chamado.

Se o cliente clicar outro botão, o método **post** irá chamar o método **getHTML** da classe correspondente.

e) HTMLConsultar

Esta classe gera HTML para o cliente consultar os pedidos que já foram cadastrados por ele. Esta classe possui o método **getHTMLLista** que gera HTML para os pedidos retornados pela *Query* “allPedidosCliente” da classe **Pedido**.

Se o cliente clicar em um botão “Verificar” ao lado de cada pedido, o método **post** irá chamar o método **getHTML** da classe **HTMLVerificaPedido.**

f) HTMLEncerrar

Esta classe gera a última página antes de efetivar a gravação do pedido, os métodos **getHTML** e **getHTMLLista** geram esta página com os produtos selecionados e um formulário com os campos para o cliente confirmar seus dados cadastrais.

Quando o cliente clica no botão “Confirmar” o método **post** chama o método **gravarPedido** que salva as informações do cliente, gera um novo pedido e chama o método **getHTML** da classe **HTMLFinaliza**.

g) HTMLErroSenha

O método **getHTML** desta classe gera uma página de erro com um botão para o cliente voltar à página anterior e repetir a operação de *login*.

h) HTMLErroUsuario

O método **getHTML** desta classe gera uma página de erro com um botão para o cliente voltar à página anterior e repetir a operação de *login*.

i) HTMLFinaliza

Esta classe gera uma página contendo o número do pedido que foi gerado e um botão para o cliente sair do sistema, ao clicar o botão, o método **post** chama o método **getHTML** da classe **HTMLSair**.

j) HTMLLogin

Esta classe é responsável pela identificação do cliente no sistema quando o cliente quer consultar o seu histórico de pedidos, o método **post** desta classe faz a validação pelo método "clientePorLogin" da classe **Cliente**, e caso a identificação for validada é chamado o método **getHTML** da classe HTMLConsultar, caso contrário é chamado o método **getHTML** da classe **HTMLErroUsuario** ou **HTMLErroSenha** dependendo do motivo de não validação da identificação do cliente.

k) HTMLSair

Esta classe possui em seu método **getHTML** código HTML para fechar a conexão com o Caché, junto com uma frase de agradecimento e um botão para sair. Quando o cliente clica neste botão, a conexão com o banco é fechada e é chamado o método **getHTML** da classe **HTMLVolta**.

l) HTMLTipoCliente

Esta classe é apresentada ao cliente quando do encerramento do pedido e o cliente ainda não foi identificado, a página gerada por esta classe possui campos para o usuário e senha do cliente e botões para ele se cadastrar como cliente físico ou cliente jurídico.

Se o cliente se identificar corretamente, o processo continuará no seu fluxo normal, levando o usuário para a página de confirmação de pedido. Caso contrário o sistema irá abrir a página de cadastro de clientes.

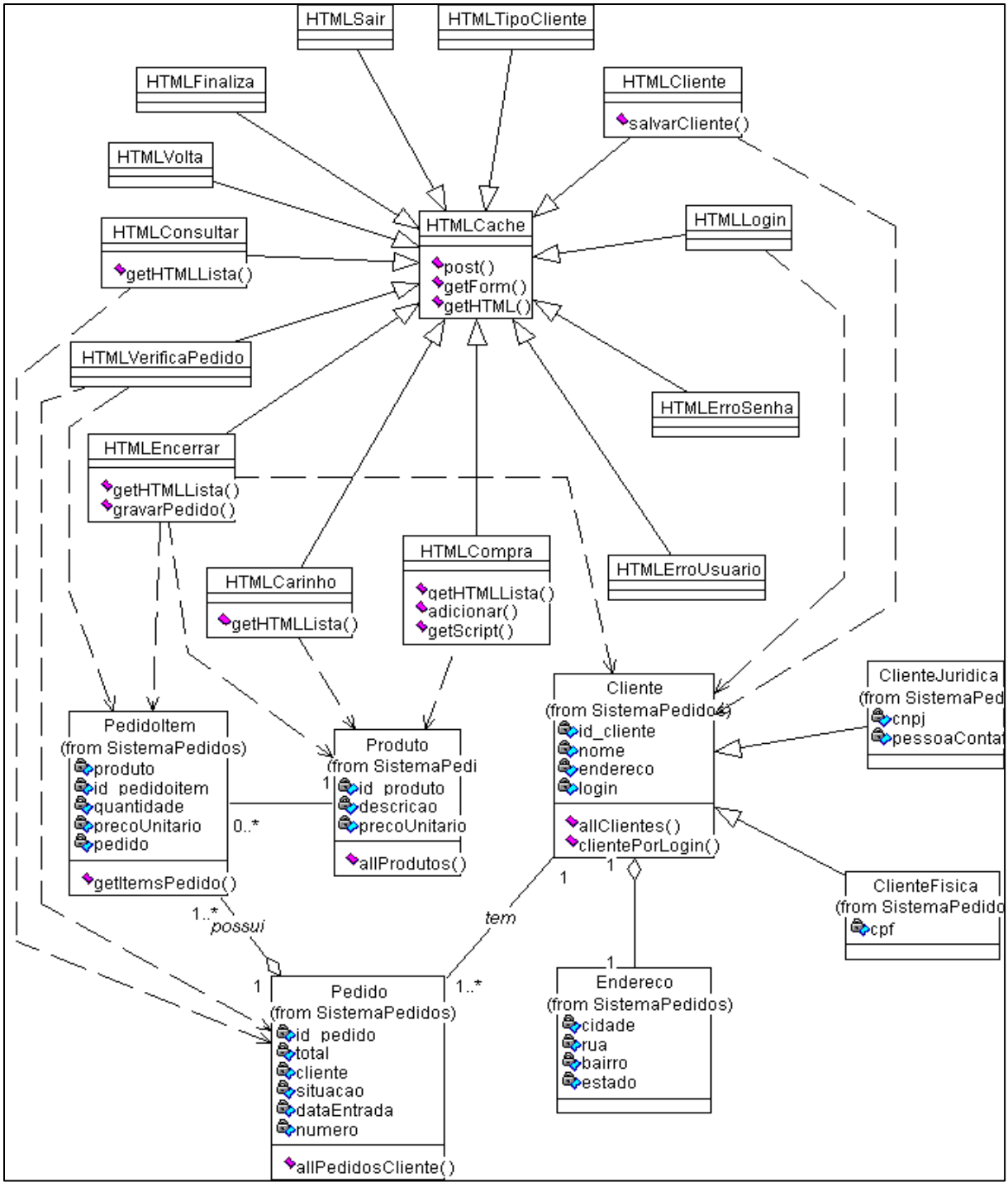
m) HTMLVerificaPedido

Esta classe que gera uma página HTML contendo todas as informações do pedido, é chamada pela classe HTMLConsultar.

n) HTMLVoltar

Esta classe possui código HTML para redirecionar o *browser* do cliente para a página inicial do sistema.

Figura 11 - Diagrama de classes de interface



3.2 IMPLEMENTAÇÃO

Neste item são descritas as ferramentas utilizadas para a implementação e é feita uma apresentação do protótipo.

3.2.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

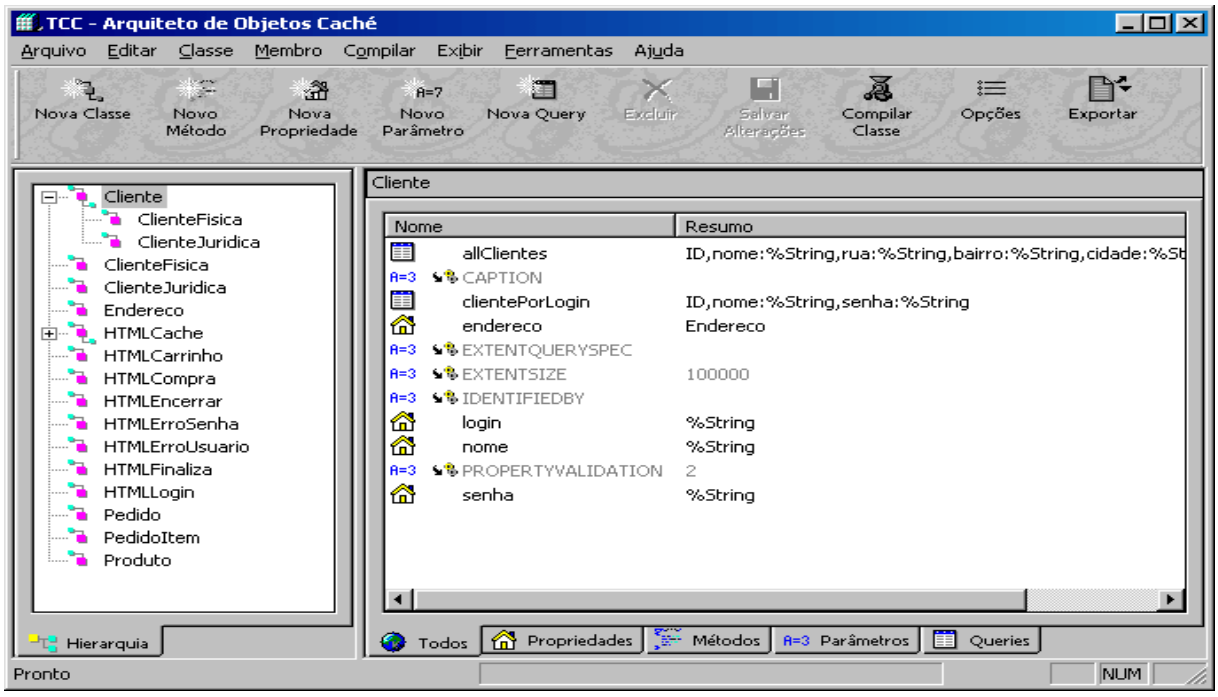
Esta parte fornece uma breve descrição das ferramentas utilizadas para a implementação do protótipo, são elas: Rational Rose, Cache Object Script, Arquiteto de Objetos, Cache Studio, Macromedia Dreamweaver e WebLink.

3.2.1.1 ARQUITETO DE OBJETOS

O arquiteto de objetos (Figura 12) é a ferramenta visual do Caché para a criação e manutenção das classes do banco de dados, nele é possível fazer toda a definição dos atributos da classe e dos objetos, dos métodos e definição de consultas SQL.

A janela do arquiteto de objetos possui duas partes principais, a janela de classes à esquerda, que mostra todas as classes disponíveis com sua hierarquia e à direita, a janela de membros, que mostra as propriedades, métodos, parâmetros e consultas da classe.

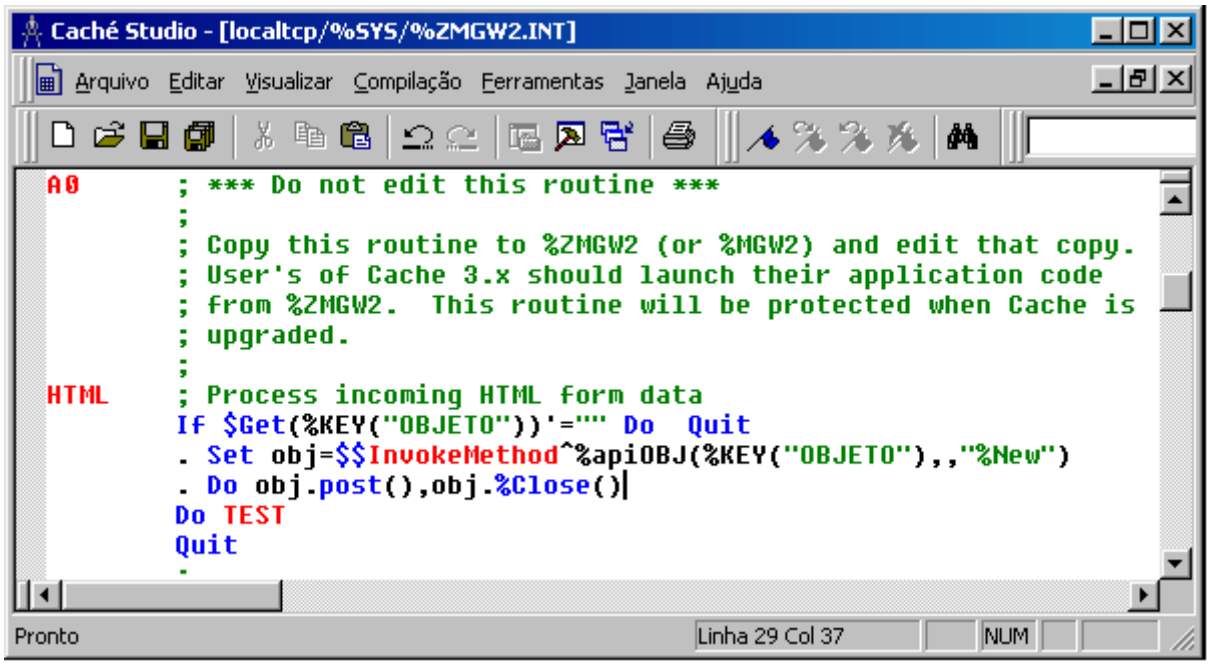
Figura 12 - Arquiteto de objetos



3.2.1.2 CACHÉ STUDIO

O Caché Studio (Figura 13) é a ferramenta que permite a manutenção de rotinas e é como um editor de texto especial que somente manipula rotinas do Caché. Possui verificação de sintaxe e mostra os elementos do COS com cores distintas.

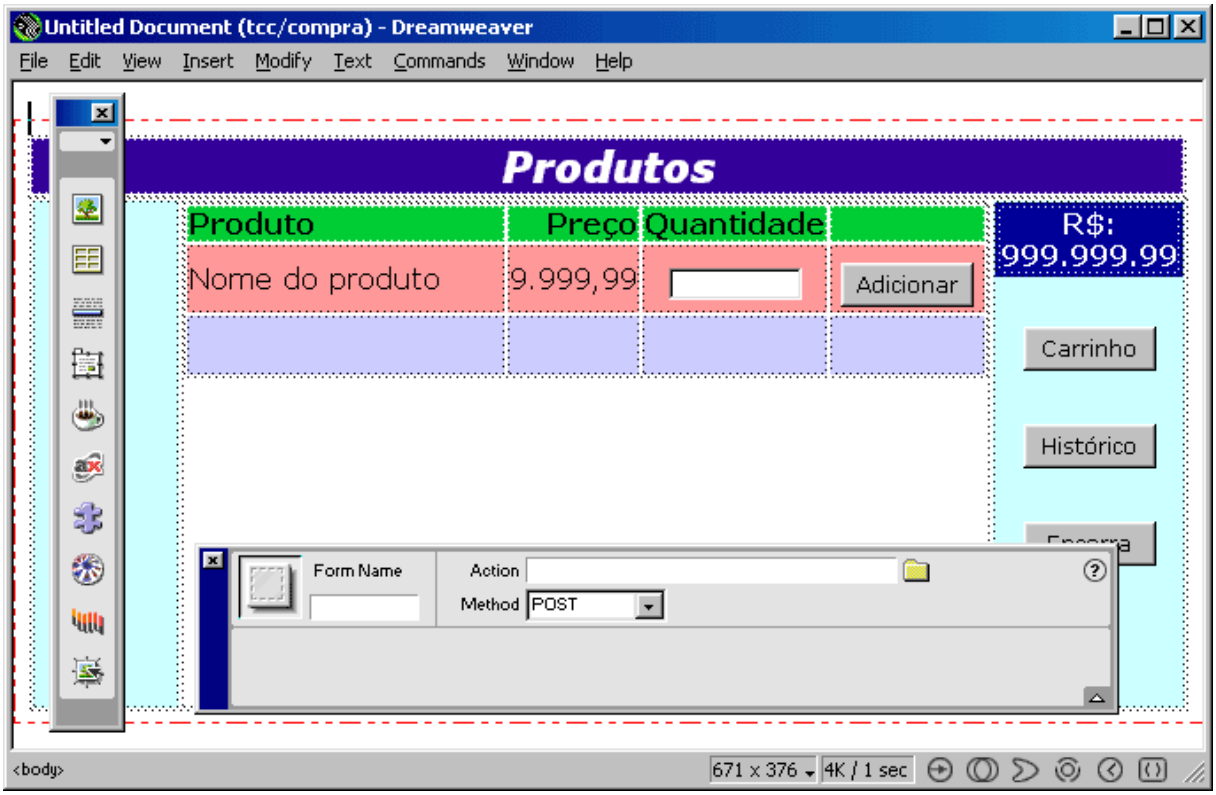
Figura 13 - Caché Studio



3.2.1.3 MACROMEDIA DREAMWEAVER

O Macromedia Dreamweaver (Figura 14) é um editor de HTML que o usuário vê o resultado durante a edição do documento HTML, o resultado gerado por esta ferramenta foi copiada para dentro das classes de interface do usuário do protótipo.

Figura 14 - Macromedia Dreamweaver



3.2.1.4 WEBLINK

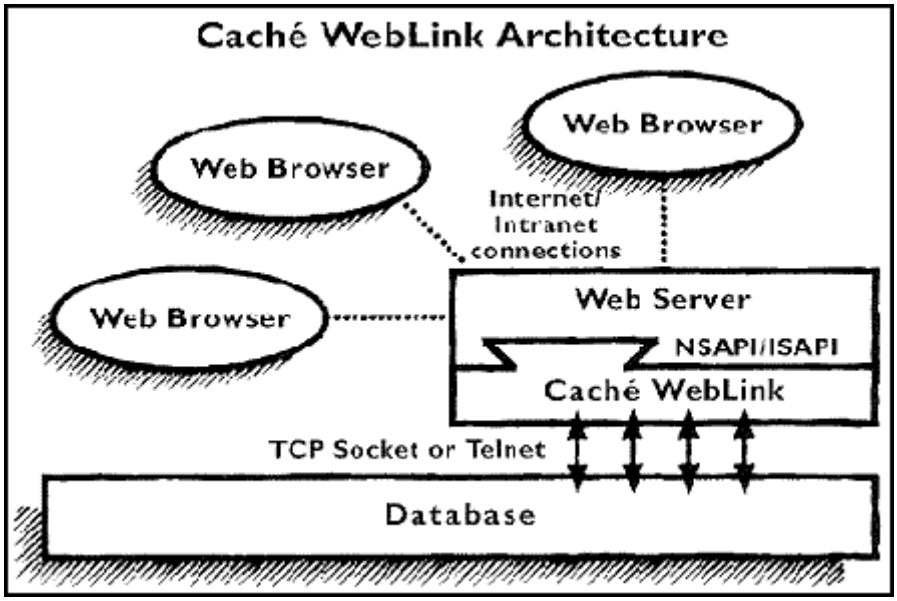
O WebLink é o produto da InterSystems que permite que os programas e objetos do Caché sejam visíveis pela Internet. O WebLink é uma ponte que liga o Caché com o servidor WWW através de *Sockets* ou Telnet conforme figura 15.

Como o WebLink funciona como uma ponte entre o servidor WWW e o banco de dados, não é necessário que estes estejam na mesma máquina. Desta maneira existe uma proteção ao servidor de banco de dados, que através de um *firewall*, pode estar protegido, enquanto o servidor WWW pode estar visível para a Internet.

O WebLink se conecta ao banco de dados e cria uma sessão onde executa uma série de procedimentos para fazer as chamadas aos programas que tratam uma requisição WWW. Estes procedimentos consistem em criar as variáveis de ambiente necessárias para o controle da conexão e as variáveis com os parâmetros e informações passados para o servidor pelo formulário HTML. Então são chamadas as rotinas que devem tratar estas requisições e retornar mais código HTML para o *browser* do usuário.

A arquitetura do WebLink permite que se faça dois tipos de conexão com o servidor Caché, a primeira é a *State Less Session* onde cada conexão do *browser* com o WebLink é feita por uma conexão com o banco e a segunda é a *State Aware Session* onde cada conexão com o WebLink abre uma nova sessão com o banco de dados. A vantagem da primeira é a utilização de apenas uma conexão com o banco, e a vantagem da segunda é a facilidade de controle do fluxo de dados pelo fato de que cada usuário tem sua própria sessão no banco de dados. As variáveis criadas durante o tratamento de uma requisição WWW são mantidas e podem ser utilizadas nas próximas requisições. É sobre este tipo de conexão que trata o protótipo deste trabalho.

Figura 15 - Arquitetura WebLink



Fonte: [INT2000]

O WebLink pode conectar-se a diversos Servidores Caché, bastando para isto criar “configurações” distintas para cada servidor.

3.2.2 APRESENTAÇÃO DO PROTÓTIPO

O protótipo é um sistema para uma loja virtual onde o cliente pode fazer pedidos e consultar o seu histórico de compras e situação dos pedidos. Depois que o usuário passou da tela de entrada ele tem acesso a uma lista de produtos onde podem ser escolhidos quais produtos devam ser incluídos no pedido. A partir desta tela é possível consultar o conteúdo do

pedido, o histórico e fazer o encerramento do pedido. O cliente deve ter um nome de usuário e uma senha cadastrada no sistema para consultar seu histórico.

A seguir mostram-se as telas do protótipo e uma breve explicação sobre elas.

Figura 16 - Entrada na loja virtual

Na tela de Entrada (figura 16) o usuário deve clicar no botão Entrar para iniciar uma seção no sistema da loja virtual.

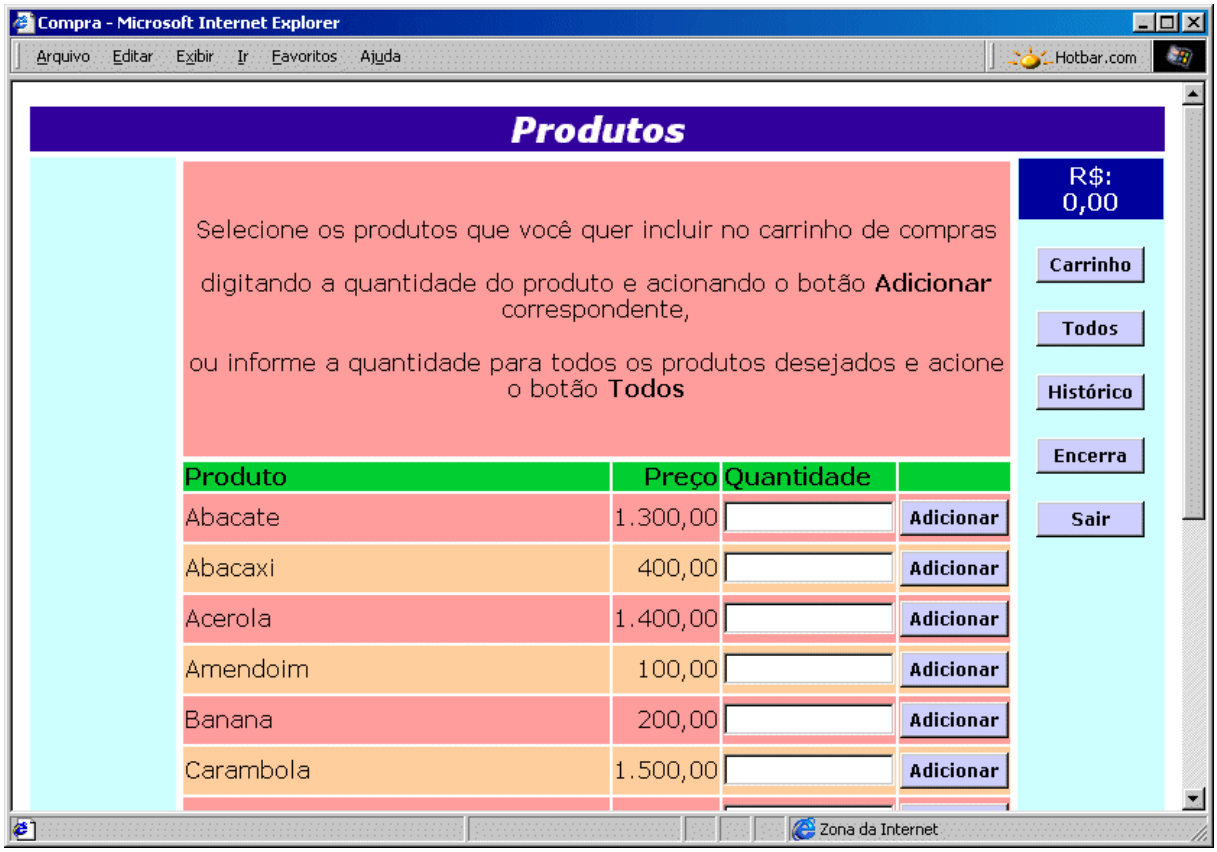
Esta página HTML possui o código abaixo, que é responsável pela conexão com o Caché e de chamar o método **post** da classe **HTMLCompra**.

```
<form method="post" action="/scripts/mgwms32.dll" name="login">
    <input type="hidden" name="MGWLPN" value="TCC">
    <input type="hidden" name="MGWCHD" value="p">
    <input type="hidden" name="OBJETO" value="HTMLCompra">
  <p align="center">Clique no Bot&atilde;o Abaixo para Comprar</p>
  <div align="center">
          <p><input class="botao" type="submit" name="entrar"
value="Entrar">
        </p></div>
</form>
```

As variáveis “MGWLPN” e “MGWCHD” são variáveis de controle do WebLink, que são respectivamente o nome da configuração que será utilizada, e o tipo de conexão, “p” para *state-aware* e “” (nulo) para conexões *state-less*, a variavel “OBJETO” tem como objetivo

fazer com que o WebLink invoque o método **post** da classe contida nesta variavel, estas tres variáveis existem em todas as páginas geradas pelo sistema.

Figura 17 - Seleção dos produtos



Na tela de seleção de produtos (figura 17) o usuário deve informar uma quantidade para o produto que deseja comprar e acionar o botão **Adicionar**, ou pode colocar quantidades para todos os produtos de interesse e acionar o botão **Todos** para adicionar todos ao mesmo tempo. O botão **Carrinho** apresenta a tela dos produtos já selecionados, o botão **Histórico** leva a uma tela contendo todos os pedidos que o usuário já incluiu e o botão **Encerra** fecha o pedido.

Esta página é gerada pela classe **HTMLCompra**, o trecho de código a seguir é o método **getHTMLLista** desta classe.

```
New Result,i
Set Result=##class(%ResultSet).%New("Produto.allProdutos")
Do Result.Execute()
For i=1:1 Quit:'Result.Next()  &HTML<
<tr bgcolor="<%=$S(i#2=1:"#FF9999",1:"#FFCC99")%>">
<td width="59%" height="33">
<%=Result.GetDataByName("descricao")%></td>
```

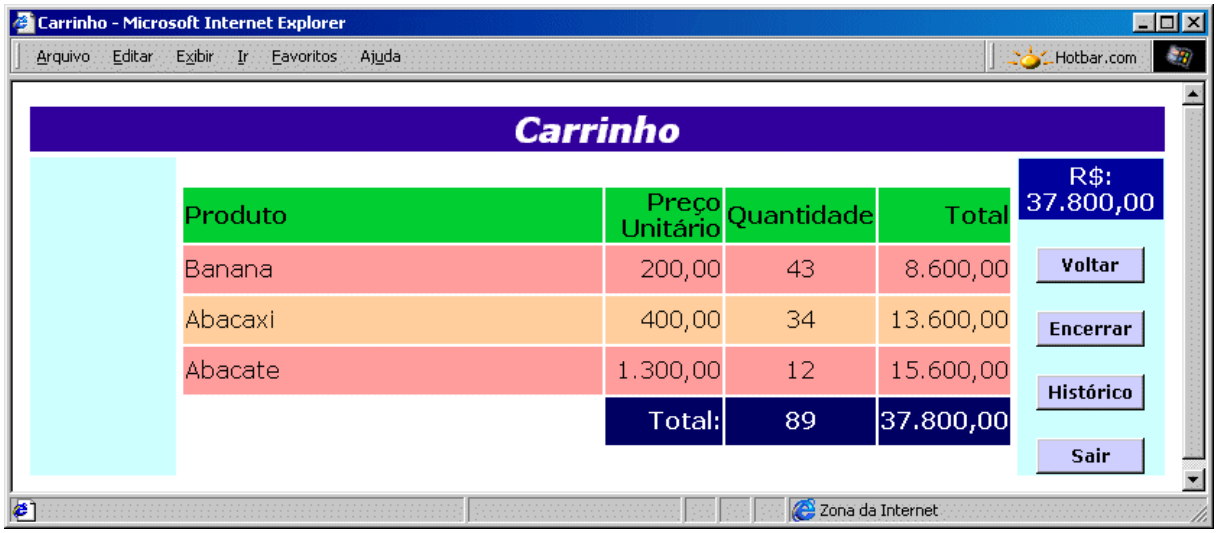
```
<td width="15%" height="33">
<div align="right">
<%=$FN(Result.GetDataByName("precoUnitario"),".",2)%></div>
</td>
<td width="14%" height="33">
<div align="center">
<input type="text" name="quantidade.<%=Result.GetDataByName("ID")%>"
size="15" maxlength="5" onBlur="validate(this)">
</div>
</td>
<td width="12%" height="33">
<div align="center">
<input type="submit" class="botao"
name="adicionar.<%=Result.GetDataByName("ID")%>" value="Adicionar" >
</div>
</td>
</tr>
>
Do Result.Close()
Do Result.%Close()
Quit
```

Este método faz uso da *Query* “allProdutos” da classe **Protutos**, esta é uma *Query* SQL, a seguir esta o comando SQL que busca os produtos da classe.

```
SELECT  ID, descricao, precoUnitario
FROM    Produto
ORDER   BY descricao
```

Para a inclusão dos produtos no banco de dados, foi utilizado o recurso da Projeção SQL e a ferramenta SQL Explorer que aconpanha o pacote do Borland Delphi 4.

Figura 18 - Carrinho



Na tela Carrinho (figura 18), o usuário pode verificar os produtos que já estão no "carrinho de compras" ou seja os produtos que já foram incluídos no pedido.

Esta página é gerada pela classe **HTMLCarrinho** a seguir o código do método **post** que chama o método **getHTML** da classe correspondente ao botão clicado.

```
new x
;
; sair
;
if $data(%KEY("sair")) do  quit
. Set x=##class(HTMLSair).%New()
. Do x.getHTML(),x.%Close()
;
; encerrar
;
if $data(%KEY("encerrar")) do  quit
. Set Origem="HTMLCarrinho",Destino="HTMLEncerrar"
. if $Data(cliente(%KEY("MGWCHD"))) Do
.. Set x=##class(HTMLEncerrar).%New()
. if '$Data(cliente(%KEY("MGWCHD"))) do
.. Set x=##class(HTMLTipoCliente).%New()
. Do x.getHTML(),x.%Close()
;
; voltar
;
if $data(%KEY("voltar")) do  quit
. Set x=##class(HTMLCompra).%New()
. Do x.getHTML(),x.%Close()
;
; consultar historico
;
if $data(%KEY("consultar")) do  quit
. Set Origem="HTMLCarrinho",Destino="HTMLConsultar"
. if $Data(cliente(%KEY("MGWCHD"))) Do
```

```
.. Set x=##class(HTMLConsultar).%New()
. if '$Data(cliente(%KEY("MGWCHD"))) do
.. Set x=##class(HTMLLogin).%New()
. Do x.getHTML(),x.%Close()
quit
```

Figura 19 - *Login* de consulta de histórico

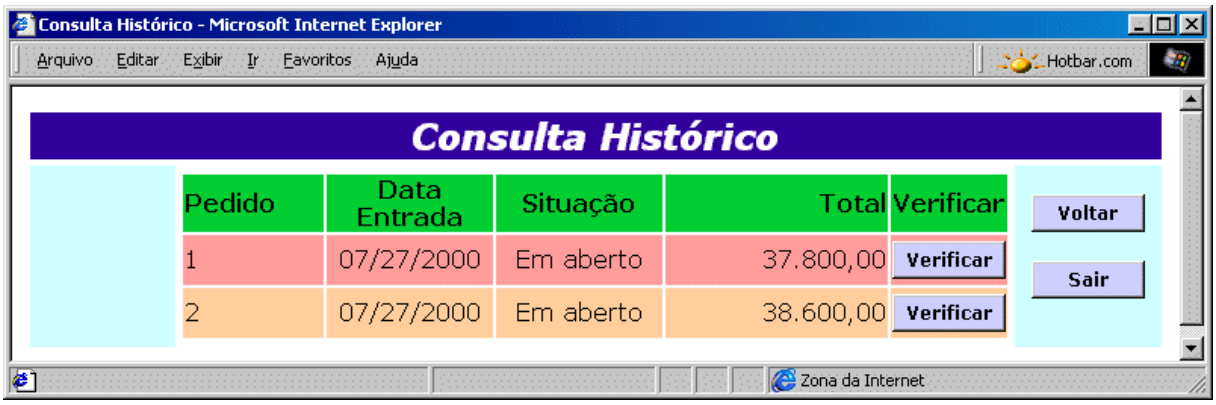
Na tela *Login* de consulta de histórico (figura 19) ,o usuário deve informar o nome e senha para consultar o seu histórico de compras. Esta opção só tem validade para clientes que já compraram pelo menos uma vez. Esta tela só é solicitada uma vez a cada seção.

A seguir encontra-se o código do método **post** que faz a validação do usuário, este método utiliza-se da *Query* "clientePorLogin" da classe **Cliente** para validacao do usuário.

```
;
; entrar
;
If $Data(%KEY("entrar")) Do  Quit
. New Result,flag
. Set Result=##class(%ResultSet).%New("Cliente.clientePorLogin")
. Do Result.Execute(%KEY("usuario"))
. Set flag=Result.Next()
. If 'flag Do  Quit
.. Set x=##class(HTMLErroUsuario).%New()
.. Do Result.Close(),Result.%Close()
.. Do x.getHTML("HTMLLogin"),x.%Close()
. If Result.GetDataByName("senha")'=$G(%KEY("senha")) Do  Quit
.. Set x=##class(HTMLErroSenha).%New()
.. Do Result.Close(),Result.%Close()
.. Do x.getHTML("HTMLLogin"),x.%Close()
. Set cliente(%KEY("MGWCHD"))=##class(Cliente).%OpenId(Result.Get↵
```

```
DataByName("ID"))
        . Set x=$$InvokeMethod^%apiOBJ(Destino,,"%New")
        . Do Result.Close(),Result.%Close(),x.getHTML(),x.%Close()
        ;
```

Figura 20 - Consulta do histórico

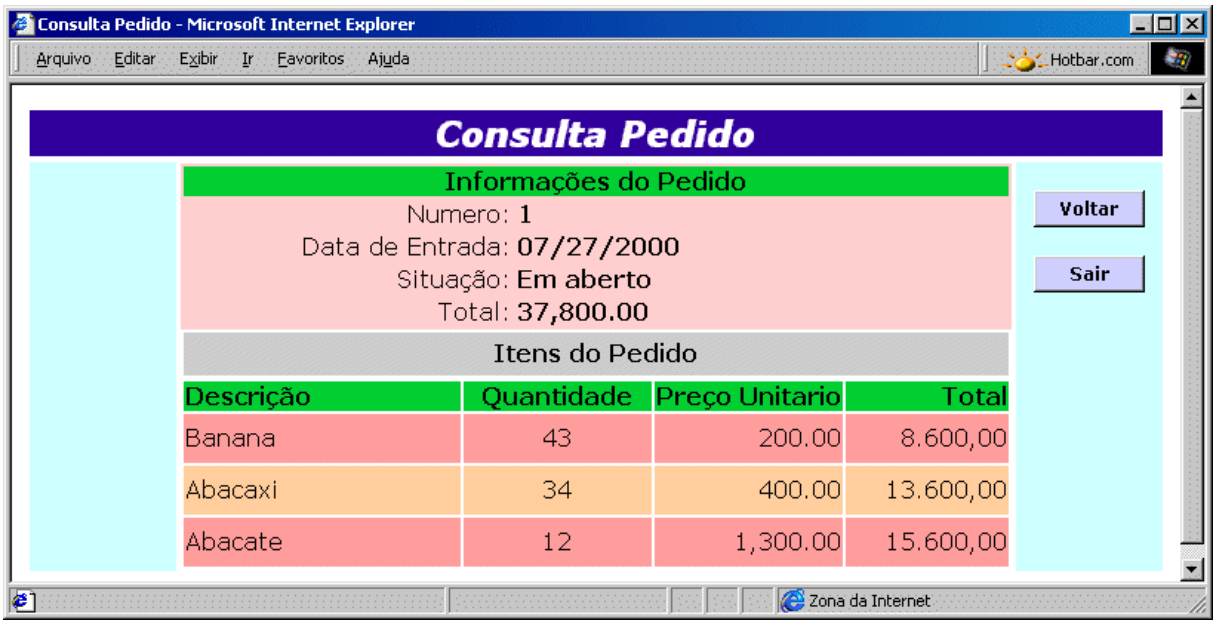


Na tela Consulta de histórico (figura 20), o usuário pode verificar todos os pedidos que já foram cadastrados por ele com a situação e opção de verificar informações detalhadas do pedido.

Esta tela é gerada pela classe **HTMLConsultar**, que utiliza a *Query* "allPedidosCliente" da classe **Pedido**, a seguir a definição desta *Query* em CDL.

```
query allPedidosCliente((cliente:%Integer))
{
        type = %SQLQuery(CONTAINID=1,ROWSPEC="ID,numero:%Integer,
        dataEntrada:%Date,situacao:%String,total:%Currency");
        sqlquery =
        {
                :SELECT ID, numero,
                : %EXTERNAL(dataEntrada),
                : %EXTERNAL(situacao), total
                :FROM Pedido
                :WHERE (cliente->ID = :cliente)
                :ORDER BY dataEntrada
        }
        sqlproc = 0;
        sqlview = 0;
}
```

Figura 21 - Consulta de pedido

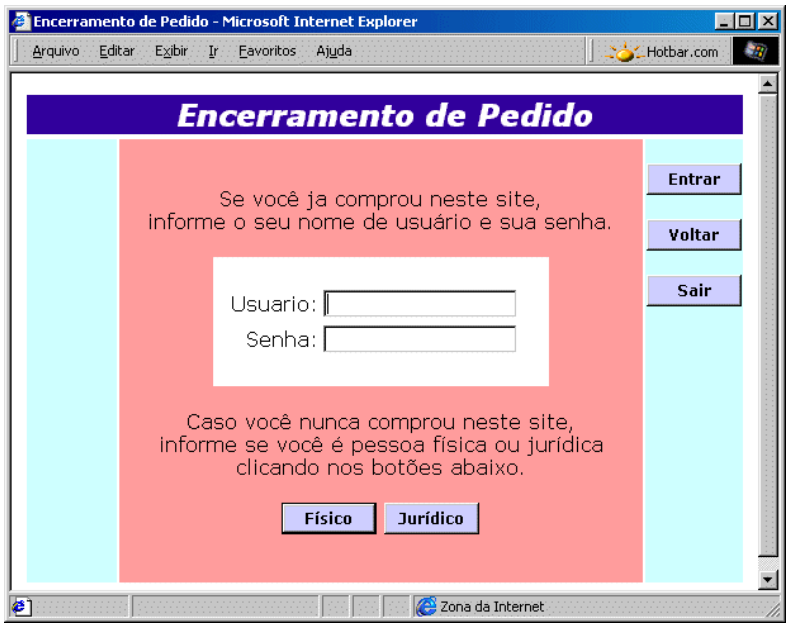


Na tela Consulta de pedido (figura 21), o cliente pode ver todas as informações a respeito do pedido e de seus itens.

A classe que gera esta página HTML é a HTMLVerificaPedido, a seguir um trecho de código do método **getHTML** que utiliza métodos do *Data Type* ***%Date***.

```
<tr>
<td width="40%">
<div align="right">Data de Entrada:</div>
</td>
<td width="60%"><b>
<%=pedido.dataEntrada.LogicalToDisplay(pedido.dataEntrada)%>
</b></td>
</tr>
```

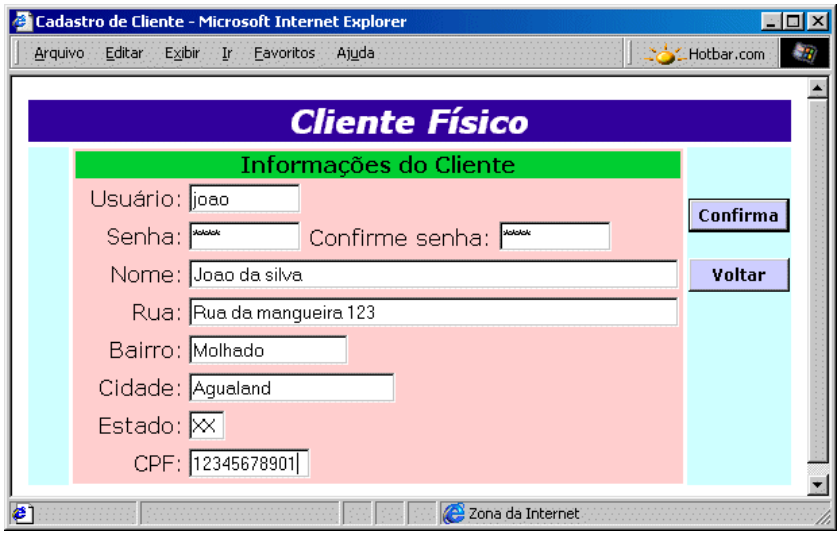
Figura 22 - *Logon* para encerramento



Quando o usuário clicar no botão de encerramento e ainda não ter registrado dados de usuário e senha para consultar o histórico, é solicitado o nome e senha para o encerramento (figura 22) e se o cliente ainda não fez compras neste site ele deve se cadastrar clicando no botão correspondente à sua categoria.

Esta página é gerada pela classe HTMLTipoCliente, ela é identica à classe HTMLLogin, mas com os dois botões do tipo de cliente onde o usuário deve clicar se for um cliente novo.

Figura 23 - Cadastro de cliente



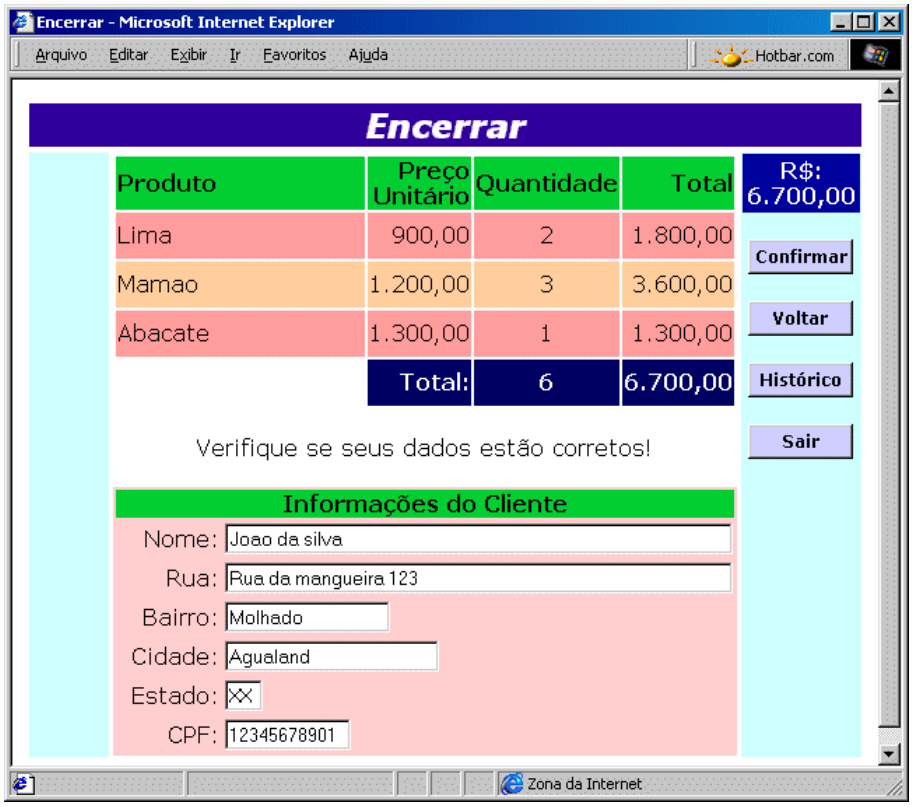
Na tela Cadastro de cliente (figura 23), o cliente que esta comprando pela primeira vez no site deve dar todas as informações solicitadas para continuar o encerramento do pedido.

Esta tela é gerada pela classe **HTMLCliente**, a seguir o código do método **salvarCliente** que é chamado pelo método **post** quando é clicado o botão “Confirma”.

```
if %KEY("senha")'=%KEY("senha2") Do  quit $$$ERROR()
. Do ..getHTML(tipo,"Senhas Diferentes")
new cliente,sc
set:tipo="fisico" cliente=##class(ClienteFisica).%New()
set:tipo'="fisico" cliente=##class(ClienteJuridica).%New()
set cliente.login=%KEY("usuario")
set cliente.senha=%KEY("senha")
set cliente.nome=%KEY("nome")
set cliente.endereco.rua=%KEY("rua")
set cliente.endereco.cidade=%KEY("cidade")
set cliente.endereco.bairro=%KEY("bairro")
set cliente.endereco.estado=%KEY("estado")
set:tipo="fisico" cliente.cpf=%KEY("cpf")
Do:tipo'="fisico"
. set cliente.cnpj=%KEY("cnpj")
. set cliente.pessoaContato=%KEY("contato")
set sc=cliente.%Save()
if $$$ISERR(sc) Do  quit $$$ERROR()
. Do ..getHTML(tipo,$$DisplayError^%apiOBJ(sc))
Do cliente.%Close()
Quit $$$OK
```

Este método testa se o cliente informou a mesma senha nos campos correspondentes do formulário, depois os campos do formulário, que estão na variável indexada **%KEY**, são inseridos na nova instância de **Cliente** e o objeto é salvo no banco de dados.

Figura 24 - Encerramento do pedido



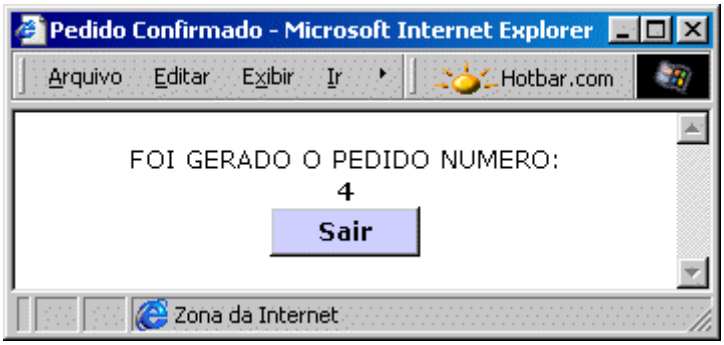
Na tela Encerramento de pedido (figura 24), o cliente confere se o pedido e suas informações cadastrais estão corretas e finaliza a transação confirmando o pedido, o sistema então devolve o número do pedido que foi gerado (figura 25).

Esta página é gerada pela classe **HTMLEncerra**, esta classe possui um método chamado “gerarPedido” que faz a geracao do pedido e gravação dos dados do cliente, que é invocado quando o cliente clica o botão “Confirmar”. Abaixo o trecho responsavel da gravação do pedido e de seus itens.

```
;
set varped=##class(Pedido).%New()
set varped.cliente=varcli
set varped.situacao=1
set varped.total=Itens(%KEY("MGWCHD"))
set varped.dataEntrada=+$h
&sql(SELECT MAX(NUMERO) INTO :numero FROM PEDIDO)
set varped.numero=numero+1
set sc=varped.%Save()
if $$$ISERR(sc) D  quit sc
. set x=##class(HTMLFinaliza).%New()
. do x.getHTML($$DisplayError^%apiOBJ(sc)),x.%Close()
set codpro=""
;
```

```
        for  set codpro=$o(Itens(%KEY("MGWCHD"),1,codpro)) quit:codpro="" ↵
do
        . set varpro=##class(Produto).%OpenId(codpro)
        . set varitem=##class(PedidoItem).%New()
        . set varitem.produto=varpro
        . set varitem.pedido=varped
        . set varitem.precoUnitario=varpro.precoUnitario
        . set varitem.quantidade=Itens(%KEY("MGWCHD"),1,codpro)
        . do varitem.%Save()
        . do varitem.%Close()
        . do varpro.%Close()
        ;
        set x=##class(HTMLFinaliza).%New()
        do x.getHTML(varped.numero),x.%Close()
        ;
        do varped.%Close()
        ;
```

Figura 25 - Pedido confirmado



Esta página é gerada pela classe HTMLFinaliza, onde o método **getHTML** recebe como parâmetro o número do pedido gerado. A seguir o trecho da definição deste método.

```
        method getHTML(numero:%Integer)
        {
                classmethod = 0;
                not final;
                public;
                sqlproc = 0;
                code =
                {
                        &html<
                        <HTML><HEAD><TITLE>Pedido Confirmado</TITLE>
                        </HEAD><BODY class="corpo">>
                        d ..getForm("finaliza","HTMLFinaliza")
                        &html<
                        <p align="center">
                        Foi gerado o pedido numero:<BR>
                        <B><%=numero%></B><BR>
                        <input class="botao" type="submit" value="Sair">
                        </p>
                        </form></body></html>>

                }
        }
```

4 CONCLUSÕES

Este capítulo apresenta as considerações finais sobre o trabalho e sugestões para aperfeiçoamento do protótipo.

4.1 CONSIDERAÇÕES FINAIS

Depois de concluído o trabalho, a pesquisa das teorias e a aplicação destas teorias no protótipo, percebe-se que as tecnologias orientadas a objeto, cumprem o papel que elas sugerem, de facilitar o desenvolvimento e aperfeiçoamento de sistemas com a sua característica de herança.

Os recursos que o Caché oferece ao desenvolvedor são de grande utilidade, estes recursos como, por exemplo, os objetos embutidos, *Data Types.* e herança. Estes recursos permitem que o desenvolvedor consiga criar sistemas com as características da orientação a objetos, como a modularidade e a reusabilidade de objetos.

Com a atual popularidade da Internet, os sistemas baseados em web estão em evidência no mercado de software. Como este trabalho faz uso intenso de HTML, o HTML embutido facilitou a inclusão de código HTML nos métodos das classes, que juntamente com o WebLink, que faz a integração entre o servidor WWW e o banco de dados Caché, permitiram a criação deste protótipo com facilidade.

Como muitas das aplicações web são utilizadas através de um *browser*, estas aplicações fazem extenso uso do HTML. A vantagem de utilizar o HTML para desenvolvimento de aplicações web é pelo fato de que são relativamente simples de serem construídas, podem ter ótima apresentação, e são de fácil aprendizado pelo usuário. E com a utilização de outras tecnologias, como JavaScript, agregadas ao HTML, é possível a criação de aplicações baseadas em web, que são tão iterativas com o usuário, quanto aplicações comums utilizando interfaces gráficas como o Windows, por exemplo.

Outro recurso interessante do Caché é a projeção SQL, que permitiu que o banco de dados de objetos fosse acessível às ferramentas para tratamento de dados, como, geradores de relatórios e ambientes de desenvolvimento como Delphi e VisualBasic. Neste trabalho foi utilizada a ferramenta SQL Explorer que acompanha o Borland Delphi 4.

O recurso de poder incluir o SQL dentro de métodos e programas do Caché é um grande aliado para o desenvolvimento, pois permite criar métodos de consultas flexíveis, e mais simples.

Foi constatado um problema durante a inclusão dos produtos que se refere a coluna ID da projeção SQL, quando o mecanismo de acesso ao banco de dados do Delphi faz uma inclusão numa tabela, no comando SQL é incluido o ID, mas o Caché não aceita que se atribua um valor para o ID, e isto gera um erro, para conseguir incluir os produtos, foi necessário a digitação dos comandos SQL sem se referenciar ao ID.

4.2 SUGESTÕES PARA FUTUROS TRABALHOS

Como sugestões para futuros trabalhos, existe a possibilidade de:

o) Pesquisar e utilizar os outros recursos do Caché, como campos *Array* ou Lista, campos calculados e *BLOB's* para dados multimídia;

p) Criar uma aplicação para o fornecedor consultar os pedidos gerados utilizando o recurso da projeção SQL dos objetos;

q) Estudar os meios de otimização que o Caché permite, para a criação de bases de dados extremamente rápidas;

r) Melhorar o aspecto segurança estudando protocolos seguros de comunicação, como SSL;

s) Facilitar a manutenção das telas HTML, já que elas estão incorporadas aos métodos dos objetos;

REFERÊNCIAS BIBLIOGRÁFICAS

[BAR2000] BARROS, Pablo **UML - Linguagem de modelagem unificada - em português.** Endereço Eletrônico: http://cc.usu.edu/~slqz9/uml/index.html, 2000.

[DAT1990] DATE, C.J. **Introdução a sistemas de banco de dados, 4ª ed.** Rio de Janeiro : Campus, 1990.

[FUR1998] FURLAN, José Davi **Modelagem de objetos através da UML**. São Paulo : Makron Books, 1998.

[INT1997a] INTERSYSTEMS Corporation. **Caché programming guide.** Estados Unidos : InterSystems Corporation, 1997.

[INT1997b] INTERSYSTEMS Corporation. **Object script language reference.** Estados Unidos : InterSystems Corporation, 1997.

[INT1998a] INTERSYSTEMS Corporation. **Caché Development Guide.** Estados Unidos : InterSystems Corporation, 1998.

[INT1998b] INTERSYSTEMS Corporation. **User's guide.** Estados Unidos : InterSystems Corporation, 1998.

[INT2000] INTERSYSTEMS Corporation. **Caché.** Endereço Eletrônico: http://www.intersystems.com , 2000

[KER1994] KERN, Vinícius. **Banco de dados relacionais.** São Paulo : Érica, 1994.

[KHO1994] KHOSHAFIAN, Setrag. **Banco de dados orientado a objeto.** Rio de Janeiro : Infobook, 1994.

[KOR1997] KORTH, Henry F., SILBERSCHATZ, Abraham. **Sistema de banco de dados, 2ª ed.** São Paulo : Makron Books, 1997.

[MAC1995] MACIASZEK, Leszek A. **Relational versus object databases: contention or coexistence?** Endereço Eletrônico : http://www.comp.mq.edu.au/courses/comp866/oovsrel.html, 1999.

[MCC1997] MCCOMB, Gordom. **JavaScript Sourcebook.** São Paulo : Makron Books 1997.

[RIT1996] RITCHEY, Tim. **Programando Java & JavaScript para Netscape 2.0**. :Editora Quark, 1996.

[SAV1997] SAVOLA, Tom. **Usando HTML o guia de referência mais completo.** Rio de Janeiro : Campus, 1997.