UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

ESTUDO SOBRE AS TÉCNICAS E LINGUAGENS DE PROGRAMAÇÃO HOSPEDEIRAS SUPORTADAS PELA FERRAMENTA HTML

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

RODRIGO RISTOW

BLUMENAU, DEZEMBRO/2000

2000/2-48

ESTUDO SOBRE AS LINGUAGENS DE PROGRAMAÇÃO HOSPEDEIRAS SUPORTADAS PELA FERRAMENTA HTML

RODRIGO RISTOW

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. Wilson Pedro Carli - Orientador da FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Wilson Pedro Carli

_____

Prof. Dalton Solano dos Reis

_____

Prof. Maurício Capobianco Lopes

AGRADECIMENTOS

A todos aqueles que direta ou indiretamente auxiliaram e acreditaram na realização deste, principalmente a meus pais Cleto e Julita e em especial a minha namorada Shirley.

SUMÁRIO

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

ADO - *Active Data Objects*

ASP - *Active Server Page*

CGI - *Common Gateway Interface*

DHTML - *Dynamic Hyper Text Markup Language*

HTML - *Hyper Text Markup Language*

HTTP - *Hyper Text Transport Protocol*

IIS - *Internet Information Server*

ISAPI - *Internet Server Aplication Programming Interface*

NSAPI - *Netscape Server Aplication Programming Interface*

OOP - *Oriented Object Program*

PERL - *Practical Extration and Report Linguage*

SSI - *Server Side Include*

URI - *Uniform Resource Identifiers*

URL - *Uniform Resource Locator*

WWW - *World Wide Web*

GNU - *General Public License*

RESUMO

Este trabalho apresenta um estudo das técnicas de programação que tem por objetivo possibilitar a criação de documentos em formato HTML dinâmicos. Este trabalho pode servir como orientação na escolha de uma linguagem de programação a ser utilizada para desenvolver uma determinada aplicação que utilize páginas dinâmicas no formato HTML.

ABSTRACT

This work presents a study of the programming techniques which have as objective facilitate the creation of documents in dynamic HTML format. This work can be applied to guide the choice of a programming language used to develop a certain application which uses dynamic pages in the HTML format.

1 INTRODUÇÃO

O desenvolvimento rápido da utilização da rede mundial de computadores (Internet), principalmente o aparecimento do comércio eletrônico, fez surgir a necessidade de uma maior interação com o usuário. São necessários o cadastramento de dados do usuário, validação de senhas, impressão de recibos, comprovantes, utilização de recursos compartilhados, acesso a banco de dados, entre outras; tarefas estas que não são suportadas ou não são executadas perfeitamente através da *Hyper Text Markup Language* **(**HTML**)**.

Segundo [RAM1997], a linguagem HTML é especialista na formatação da página e na criação de *hiperlinks*. Ela é totalmente desprovida de recursos para a validação de conteúdos e comandos estruturais que permitam a execução repetitiva de trechos do programa. Quando da criação de um formulário para receber dados do usuário, a linguagem HTML simplesmente mostra o campo e aceita a digitação. Se o usuário digitar dados incompatíveis com o campo, o desenvolvedor não tem como validar através da linguagem HTML pura.

As várias limitações da linguagem HTML fizeram surgir novas técnicas de programação com o objetivo de tornar a linguagem HTML mais dinâmica. Estas técnicas tornam possível avaliar o conteúdo de campos digitados, retornar mensagens de erro ou advertência, consultar base de dados remota, inserir efeitos de computação gráfica, entre outros.

1.1 OBJETIVOS

O objetivo deste trabalho é realizar um estudo das técnicas de programação que tem por objetivo tornar os documentos HTML dinâmicos. São estudadas as linguagens de *script Java Script* e *Visual Basic Script, as Applets* implementadas em linguagem *Java*, solicitação através da *Common Gateway Interface* (CGI) utilizando a linguagem Perl, *Active Server Page* (ASP) e os comandos *Server Side Include* (SSI).

Estas técnicas de programação serão avaliadas de acordo com suas características, tais como, se a linguagem é compilada ou interpretada, se possui suporte ao tratamento de exceções, legibilidade da linguagem, sintaxe, performance em velocidade de processamento, portabilidade, entre outras. Este estudo servirá de suporte na tomada de decisão sobre quais as vantagens e quais as desvantagens de cada técnica utilizada para desenvolver uma aplicação

comercial de venda de produtos utilizando a rede internet, também conhecido como comércio eletrônico ou *e-business*.

1.2 ORGANIZAÇÃO DO TEXTO

No capítulo 2 é apresentada uma visão geral dos documentos HTML e as tecnologias utilizadas para explorar a interatividade nestes documentos.

No capítulo 3 são apresentadas todas as técnicas estudadas para implementar documentos HTML dinâmicos, serão apresentadas as linguagens de *Script Java Script* e *Visual Basic Script*, as *Applets* implementadas em linguagem *Java*, solicitações através da CGI utilizando a linguagem PERL, ASP e Comandos SSI.

No capítulo 4 é especificada a implementação utilizada para demonstrar a utilização das tecnologias estudadas neste trabalho e permitindo fazer uma analogia entre as mesmas.

No capítulo 5 são descritas as conclusões do estudo identificando quais as vantagens e quais as desvantagens de cada tecnologia em relação a legibilidade, performance, acesso a banco de dados entre outros.

2 DOCUMENTOS HTML

Uma página *Hyper Text Markup Linguage* (HTML), também chamada página *World Wide Web* (WWW), é um documento composto basicamente de textos e símbolos especiais chamados *tags*. Quando um *navegador* exibe uma página HTML, ele interpreta esses símbolos especiais que indicam como a informação deve ser exibida. Um *tag* pode informar que um texto deve ser exibido em negrito, itálico ou com um determinado tipo de fonte. Além do aspecto físico do texto, um *tag* pode indicar um *link* (atalho para outro endereço eletrônico), tabelas, figura, aplicações em *Java Applet*, entre outros ([STA1996]).

Os *tags* normalmente são especificados em pares, delimitando uma área que sofrerá algum tipo de formatação, contudo existem vários *tags* individuais. Os *tags* são identificados por serem envolvidos pelos sinais "<" e ">". O formato genérico de um *tag* é <Nome_do_tag> texto </Nome_do_tag>. No quadro 1 é apresentado um exemplo do código fonte de uma página HTML.

Quadro 1 - Exemplo do código fonte de uma página HTML

```
<HTML>
<body>
<b>Este texto está em negrito</b> <br>
enquanto esta <i>palavra</i> está em itálico.<br>
<h3>Este comando gera um cabeçalho título de tamanho 3</h3><br>
</body>
</HTML>
```

Em muitos casos tem-se a necessidade de atribuir mais funcionalidade ou recursos adicionais na página HTML. Existem muitas formas de permitir uma maior interação entre o usuário e as páginas HTML, tornando o documento HTML padrão em um documento dinâmico. A palavra dinâmico tem origem grega *dynamikós* e significa relativo ao movimento e às forças. O conteúdo de um documento HTML dinâmico pode variar de acordo com a solicitação do cliente ao servidor como por exemplo em pesquisas *on-line* de endereços eletrônicos, confirmação do cadastramento de um cliente em um site comercial, entre outros. Uma página também pode responder a interação do usuário como por exemplo exibir mensagens de advertência ao clicar de botões, armazenar informações do usuário, avaliar teclas digitadas, entre outros. Todos estes recursos adicionais existentes em páginas dinâmicas são resultado de um processamento efetuado por alguma técnica de programação em conjunto com os documentos HTML. Estes documentos dinâmicos podem ser processados pela mesma

estação que executa o servidor *internet*, pela mesma estação que executa o navegador cliente ou por ambas as estações utilizando várias técnicas simultaneamente.

As tecnologias que são executadas na estação do Cliente são denominadas *Client Side Include* (CSI) que significa “Incluído no lado do Cliente”, entre estas serão estudadas:

a) Linguagens de *Script*: são linguagens interpretadas que podem ser encapsuladas diretamente no documentos HTML. Podem ser especificadas várias linguagens desde que o navegador tenha capacidade de executar a linguagem de *script* solicitada e exibir o resultado da execução na página HTML. As linguagens *Java Script* e *Visual Basic Script* são apresentadas neste trabalho;
b) *Applet*: é um pequeno programa desenvolvido em linguagem *Java* e incluído no documento HTML através da *tag “applet”*. O código fonte da *Applet* não é encapsulado diretamente no documento HTML. O código é compilado em arquivos binários separados que devem ser carregados e executados pelo navegador.

As tecnologias que são executadas na estação do servidor internet são denominadas *Server Side Include* (SSI) que significa “Incluído no lado do servidor”, entre estas serão estudadas:

a) *Common Gateway Interface* (CGI): é a especificação padrão que define como programas externos são solicitados pelo cliente, como são utilizados no servidor internet e como eles interagem com outras aplicações ([STH1996]). É uma *interface* entre o cliente usuário de um programa navegador e o servidor responsável por executar as solicitações do usuário, gerando uma resposta. Quando o navegador solicita um programa CGI, o servidor executa ou encaminha o código para ser executado por um compilador ou interpretador. Após o processamento são retornadas as informações para o cliente. Estas informações devem obedecer o protocolo *Hyper Text Transport Protocol* (HTTP) para o servidor poder enviar os dados ao cliente e os mesmos serem reconhecidos pelo navegador. Em geral estas informações representam documentos HTML que contém dados solicitados, mas outros tipos de informações também podem ser retornadas como imagem, som ou vídeo, desde que seja respeitado o protocolo HTTP. Uma aplicação CGI pode ser escrita nas mais diversas linguagens como C, *Java* ou *Delphi*. Neste trabalho será

apresentada a linguagem PERL, uma das principais linguagens utilizadas em aplicações CGI;

b) *Active Server Page* (ASP): é uma tecnologia desenvolvida pela empresa Microsoft para o desenvolvimento de documentos HTML dinâmicos. Um documento ASP é um documento HTML normal que inclui *Server Side Scripts*. *Server Side Scripts* são códigos fonte de uma linguagem de programação hospedados em um documento HTML e processados pelo servidor ([MAR1999]). Quando o navegador solicita uma página ASP, o servidor internet analisa o arquivo a procura de códigos de *script*, executa os mesmos e retorna para o cliente apenas o documento HTML padrão;

c) Comando *Server Side Include* (SSI) é uma tecnologia que permite incluir comandos especiais encapsulados em documentos HTML. Embora possua apenas alguns comandos definidos e não possua os recursos de controle de fluxo que as linguagens de programação possuem, é muito utilizada nos documentos HTML. O comando é encapsulado no documento através de uma variação especial da *tag* de comentário do documento HTML. O servidor é responsável por analisar o documento HTML, interpretar o comando e exibir o resultado da execução na página HTML.

3 TECNOLOGIAS DINÂMICAS NA HTML

A seguir serão descritas as várias tecnologias abordadas neste trabalho que possibilitam a criação de documentos HTML dinâmicos.

3.1 AS LINGUAGENS DE *SCRIPT*

Os Scripts são códigos fonte de uma linguagem interpretada encapsuladas no documento HTML. Várias linguagens podem ser utilizadas em Scripts, mas o navegador deve ter a capacidade de interpretar e executar tais linguagens. Um programa de *Script* por ser interpretado, não gera um arquivo executável, por este motivo é possível visualizar o código fonte diretamente no documento HTML.

Para inserir o código fonte no HTML é necessário especificar a *tag script* da linguagem HTML, como pode ser visto no quadro 2.

Quadro 2 - Sintaxe da *tag Script* da linguagem HTML

```
<script language="linguagem_x" src="arquivofonte.ext">
<!--
  código fonte
  //-->
</script>
```

As seguintes especificações de atributos podem ser utilizadas:

a) *language*: o atributo de idioma é opcional, mas recomendado. Pode-se especificar que uma seção de código só pode ser executada por navegadores que suportam uma versão particular da linguagem especificada;
b) *src*: o atributo opcional pode ser usado para incluir um arquivo externo que contenha o código fonte da linguagem.

3.1.1 A LINGUAGEM *JAVA SCRIPT*

A linguagem *Java Script* é uma linguagem interpretada, hospedada nos documentos HTML e muito utilizada para pequenos controles em documentos HTML. É muito semelhante a linguagem *Java*, sendo que implementa, por exemplo, os mesmos operadores condicionais e operadores matemáticos.

3.1.1.1 HISTÓRIA

Inicialmente a linguagem *Java Script* foi criada pela Netscape com o nome de LiveScript, mas por uma questão de marketing e devido a sua associação com a Sun, resolveu-se mudar seu nome para *Java Script*, como é conhecida atualmente, visto que muitas características das linguagens *Java* e *Java Script* são semelhantes, provavelmente por terem usado como modelo a linguagem C ([RIT1998]). O *Java Script* começou a ser utilizado pela empresa Netscape a partir da versão 2.0 do *browser* Netscape Navigator e pela empresa Microsoft a partir da versão 3.0 do *browser Internet Explorer*.

3.1.1.2 VARIÁVEIS

As variáveis em Java *Script* não necessitam ser declaras explicitamente, a simples atribuição de valores a uma variável é responsável por sua criação. A declaração explicita é possível através da utilização da palavra reservada *var*, não sendo necessário identificar o tipo da variável. As variáveis são locais quando declaradas internamente a uma função ou globais quando declaradas no bloco mais externo do *script*. É possível ainda utilizar a palavra reservada *null* para identificar valores nulos nas variáveis. No quadro 3 pode ser observado um exemplo da sintaxe da declaração de variáveis em *Java Script.*

Quadro 3 - Exemplo da sintaxe da declaração de variáveis em *Java Script*

```
<script language="JavaScript">
<!--
  var var_numerica = 12;
  var var_alfa;
  var_alfa = 'teste';
  var_alfa = null;
//-->
</script>
```

3.1.1.3 TRATAMENTO DE EXCEÇÕES

Segundo [EDW1998], uma das maiores deficiências da linguagem *Java Script* é a falta de qualquer suporte ao tratamento de exceções, deficiência esta suprida recentemente pela empresa Microsoft através do seu interpretador JScript 5.0 incorporado ao *browser Internet Explorer* versão 5.

O tratamento de exceções implementado pela empresa Microsoft é muito similar o implementado em C++ e *Java* os quais se utilizam do comando *try*. A sintaxe é a mesma da linguagem *Java*. Como *Java*, a maior diferença é a utilização do comando *Instanceof* para identificar explicitamente o tipo de exceção tratada, enquanto que em *Java* isto é feito automaticamente pela linguagem através da verificação do tipo do objeto *exception*.

Quando uma exceção não é tratada por nenhum método o navegador é responsável pelo tratamento geral das exceções. Em geral são mostradas mensagens de erro ao usuário com a possibilidade de continuar executando os códigos em *Java Script* ou cancelar a execução dos mesmos. No quadro 4 pode ser visto um exemplo do tratamento de erros em *Java Script*.

Quadro 4 - Tratamento de erros em *Java Script*

```
(...)
// Define uma exceção.
Function Exception(Number, Description) {
  this.Number = Number;
  this.Description = Description;}

functction CauseException() {
  // Cria uma instância da exceção.
  exception = new Exception(1, "Erro gerado desenvolvedor");
  // Propaga o erro.
  Throw exception;
  Return = false;}

Function RaiseException() {
  Try {
    CauseException();
  } catch (exception) {
    // Verifica o tipo de exceção.
    if (exception instanceof Exception) {
      window.alert("Falha é nossa! ");
    else {
      // propaga a exceção.
      throw exception;
    }
  }
}
(...)
```

3.1.1.4 ORIENTAÇÃO A OBJETOS

Segundo [RIT1998], a linguagem *Java Script* não é considerada totalmente orientada a objetos porque não oferece muitas das funções básicas de linguagens orientadas a objetos como abstração, herança e encapsulamento. Sua capacidade de usar objetos que foram criados em outras linguagens mais tradicionais de programação orientada a objetos (OOP) como *Java* levou os projetistas *Java Script* a chamá-la de uma linguagem baseada em objetos.

A modelagem de um objeto segue a mesma sintaxe da criação de uma função. As propriedades do objeto são implementadas pelas variáveis locais e os métodos podem ser implementados pelas funções externas. A palavra chave *this* pode ser usada para referenciar o objeto atual. A instanciação de um novo objeto é feita através do comando *new*, como pode ser visto no quadro 5.

Quadro 5 - Exemplo dos eventos na linguagem *Java Script*

```
(...)
<script language="JavaScript">
function campovital(x) {
  if (x.value=="") {
    alert("O campo "+ x.name +" não pode ser deixado em branco")
  }
}

function MeuObjeto(Param1) {
  this.prop1 = Param1
  this.metodo1 = Func_Metodo1
}

function Func_Metodo1() {
  alert(“Metodo do objeto MeuObjeto executado”)
}

</script>
(...)
<form >
<h2>Testando alterações em um campo com onChange</h2>
<br>Código....:<input type="text" name="código" value="" size=5
maxlength=5 onBlur="campovital(this)"
</form>

NovoObjeto = new meuObjeto(“valor_qualquer”)
NovoObjeto.Metodo1()
(...)
```

Java Script possibilita a utilização de todos os objetos disponibilizados pelo próprio ambiente e que estão definidos no documento HTML, como por exemplo formulários, tabelas, imagens, entre outros. Os principais objetos disponibilizados pelo ambiente HTML são:

a) objeto *window*: é o objeto de maior hierarquia dentro de um documento. Contém informações sobre as janelas da página, métodos para emitir mensagens de alerta, abrir ou fechar janelas do navegador, entre outros;
b) objeto *document*: este objeto representa a página ou documento HTML como um todo, incluindo dados sobre os elementos de formulários, *links* e âncoras, além de uma série de funções que permitem mudar as características da página;
c) objeto *form*: este objeto representa os formulários implementados na página HTML, tais como o seu método, URL e dados sobre seus elementos ou campos;

d) objeto *history*: este objeto mantém uma lista de todos os sites visitados na sessão de uso atual do navegador;

e) objeto *location*: este objeto possui informações sobre o local da página e informações relacionadas, tais como o protocolo utilizado e seu domínio.

3.1.1.5 CONTROLE DE FLUXO

Os comandos condicionais ou de controle de fluxo no *Java Script* tem sintaxe muito semelhantes aos comandos em *Java Applet* e sua funcionalidade é a mesma. A única exceção é o comando *switch* que não está disponível na linguagem *Java Script*. Um exemplo da sintaxe destes comandos pode ser visto no quadro 6.

Quadro 6 - Exemplo da sintaxe dos comandos condicionais na linguagem *Java Script*

```
(...)
// Comando For
For (var Contador = 1; Contador < 11; Contador++)
{ document.write(Contador); }

// Comando If..Else
If anos >= 16
{ Return ('Já pode votar!') }
else
{ Return (' Ainda é muito cedo para votar...') }

// Comando While
Var Contador=1;
While ( Contador < 11 )
{ document.write(Contador++) ;}
(...)
```

As funções em *Java Script* são a base para a criação de estruturas capazes de manipular os eventos do documento, permitindo que uma página na internet implementada com *Java Script* reaja aos eventos provocados pelo usuário sem se basear em consultas ao servidor e chamadas de rede. É interessante declarar as funções no interior dos marcadores <head> e </head> porque esta parte é interpretada antes do restante da página, evitando a chamada a alguma função não carregada em memória. O comando *return* (retornar) é utilizado para retornar os valores de resposta da função. A chamada de uma função pode ocorrer de forma explícita pelo nome da função ou através dos eventos invocados por algum objeto do documento HTML. Os principais eventos que podem ser tratados são:

a) onBlur: ocorre sempre que um campo perde o foco;

b) onChange: ocorre sempre que o conteúdo de um campo é alterado;

c) onClick: ocorre sempre que um objeto do formulário recebe um clique;
d) onFocus: ocorre sempre que um campo recebe o foco;
e) onLoad: ocorre sempre que uma janela é totalmente carregada;
f) onMouseOver: ocorre sempre que o *mouse* passa por cima de um objeto;
g) onSelect: ocorre sempre que um texto de um campo do tipo *text* ou *textarea* é selecionado;
h) onSubmit: ocorre sempre que o usuário envia um formulário para o servidor;
i) onUnload: ocorre sempre que um documento é abandonado.

No quadro 5 pode ser visto um exemplo da utilização dos eventos em *Java Script*. No evento “onBlur” do campo de texto denominado “código” é setada a função “campovital”, que faz a consistência do campo.

3.1.2 A LINGUAGEM *VISUAL BASIC SCRIPT*

O *Visual Basic Script* (VBS) é uma linguagem interpretada que descende da linguagem *Visual Basic* padrão. Foi criada pela empresa Microsoft e por este motivo é executada de forma nativa pelo *browser Internet Explorer* a partir da versão 3 e suportada de forma nativa pelo servidor *Internet Information Server* (IIS).

3.1.2.1 HISTÓRIA

A linguagem Basic foi criada em 1963 por John Kemeny e Thomas Kurtz no Dartmouth College. Eles criaram o Basic com o propósito de ensinar conceitos de programação, enfatizando a clareza, em prejuízo da velocidade e eficiência ([MAR1986]). O progresso do Basic seguiu de perto a revolução do computador pessoal e em meados dos anos 70 a empresa Microsoft utilizou esta linguagem como base para criar um sistema residente em circuitos *Read Only Memory* (ROM), para os primeiros computadores pessoais baseados em microprocessadores ([CRA1994]).

Em 1982, a linguagem QuickBasic, descendente da linguagem Basic, se tornou uma linguagem confiável para o ambiente MS-DOS. Esta linguagem evoluiu muito ao longo destas últimas décadas, resultando atualmente as linguagens *Visual Basic* e *Visual Basic Script* respectivamente.

3.1.2.2 VARIÁVEIS

Enquanto o *Visual Basic* suporta muitos tipos de dados em variáveis, o *Visual Basic Script* apenas apresenta o tipo *variant*. Este tipo é único porque suporta os outros tipos de dados *integer, double, string, date* e *currency* utilizados na linguagem. O tipo *variant* assume o tipo de variável dependendo da atribuição feita a ele. Por padrão a declaração das variáveis não é obrigatória. Pode-se apenas fazer referências a elas, sem declará-las explicitamente.

Escopos podem ser definidos na declaração das variáveis. O termo escopo refere-se ao espaço de tempo no qual uma variável pode ser referenciada ([GHE1992]). Em VBS podem ser definidos os seguintes escopos para variáveis:

a) *dim*: ao declarar uma variável como *dim*, a mesma é referida como uma variável local ao âmbito em que foi declarada;
b) *public*: variáveis declaradas como *public* estarão disponíveis em qualquer contexto da aplicação;
c) *static*: as variáveis do tipo *static* preservam o seu valor mesmo saindo do contexto em que estão declaradas.

Um exemplo da declaração das variáveis em VBS pode ser visto no quadro 7.

Quadro 7 - Sintaxe da declaração de variáveis em VBS

```
Dim teste
If (teste = empty) then
  Teste = “Uma variavel”
Teste = 25
Teste = Null
```

A palavra chave *Empty* é usada para indicar o valor de uma variável não inicializada e a palavra chave Null significa que uma variável não contém dados válidos.

A linguagem VBS é pouco tipificada. Isso significa que não existe um tratamento rígido em relação aos tipos de variáveis, o interpretador automaticamente determina as conversões que precisam ser feitas e como elas serão feitas.

Embora a declaração das variáveis não seja necessária na linguagem VBS, é possível forçar a declaração explícita através da diretiva de compilação chamada “Option Explicit”. A

colocação de *Option Explicit* no início de uma seção <*SCRIPT*> faz o interpretador gerar exceções sempre que o código tentar utilizar alguma variável não declarada.

A linguagem VBS utiliza-se do conceito de variáveis booleanas para *True* (Verdadeiro, valor interno igual a "-1") e *False* (Falso, valor interno igual a "0").

É possível a criação de *arrays* de variáveis através da sintaxe "Dim nome_array(Tamanho)". Um *array* é uma coleção (conjunto) de variáveis do mesmo tipo. Para arrays com mais de uma dimensão é necessário utilizar a sintaxe nome_array (dimensão1.dimensão2). Os *arrays* são iniciados em zero (base zero).

É possível redimensionar um *array* em tempo de execução. Neste caso o *array* deve ser declarado sem fazer referência ao tamanho através da sintaxe "Dim nome_array()" e depois redimensionado o número de vezes que for necessário através da sintaxe "ReDim nome_array (Tamanho)".

3.1.2.3 TRATAMENTO DE EXCEÇÕES

A instrução *on error* e o objeto *err* trabalham em conjunto com o objetivo de proporcionar o controle de erros nos procedimentos do VBS. Para iniciar o tratamento de erros é necessário chamar a instrução "*on error resume next*". No VBS este tipo de instrução "*on error*" pode ser criada para informa ao interpretador para ignorar qualquer erro que possa ser gerado, prosseguindo a execução do programa na linha seguinte ao erro. O objeto *err* contém as seguintes propriedades que permitem avaliar o erro:

a) *number*: número do erro;
b) *description*: descrição do erro;
c) *source*: recurso ao qual se refere o erro.

O objeto *err* contém também dois métodos:

a) *clear*: reinicializa os valores padrões para o objeto *err*;
b) *raise*: propaga a exceção para o próximo tratamento.

Um exemplo do tratamento de exceções em VBS pode ser visto no quadro 8.

Quadro 8 - Exemplo do tratamento de exceções em VBS

```
(...)
‘ inicia o tratamento de erros
on error resume next
response.write "--- Executando divisão: 23 / 0 <p>"
‘quando ocorre um erro a execução continua na prox. instrução
set i = 23 / 0
if Err.number > 0 then
response.write "    Erro numero = "& err.number&"<p>"
  response.write "   Descricao = "&err.description&"<p>"
  Err.Clear
end if
(...)
```

3.1.2.4 ORIENTAÇÃO A OBJETOS

A linguagem VBS é uma linguagem procedural que não possui os recursos de orientação a objetos como classes ou heranças, mas que possibilita o acesso aos objetos definidos no documento HTML, tais como botões, tabelas, figuras, texto, entre outros. Para fazer referência a um objeto definido no documento HTML pode-se especificar Nome_do_Formulario.Nome_do_objeto.Propriedade_do_objeto. O comando *set* é utilizado - para relacionar uma variável a um objeto do HTML, através da sintaxe “*Set* Nome_Variavel = Objeto”. No quadro 9 pode ser visto um exemplo onde é definido um formulário denominado “RodarCli”. Depois é executado um código em VBS no evento “*On_Click*” (clique do *mouse* no botão) que altera a propriedade *value* (Descrição) do objeto “Botão_1”.

Quadro 9 - Exemplo em VBS

```
(...)
<form method="POST" name="RodarCli">
    <p><input type="button" name="Botao1"
    value="Antes do script"></p>
</form>
<p><script language="VBScript">
SUB BOTAO1_ONCLICK()
  RodarCli.BOTAO1.Value = "Depois do Visual Basic Script"
END SUB</script></p>
(...)
```

3.1.2.5 FUNÇÕES E PROCEDIMENTOS

A palavra chave *sub* inicia uma sub-rotina. Uma sub-rotina é um conjunto de instruções de programa inserido entre a palavra chave “*Sub*” e a instrução “*end sub*”, que assinala o final da sub-rotina.

Uma função é iniciada pela instrução *function* e finalizada pela instrução *end Function*. Ela tem a mesma estrutura geral de uma sub-rotina, porém tem a capacidade de retornar valor através da atribuição do valor ao nome da função.

3.2 APPLET JAVA

Um programa *Java* com a capacidade de ser executado no navegador do cliente encapsulado no documento HTML é chamada *Java Applet* ou simplesmente *Applet*. A linguagem *Java*, utilizada para desenvolver as *applets*, é uma linguagem de alto nível, similar a C, C++, Pascal e Modula-3. *Java* foi desenvolvida pela empresa Sun Microsystems e destaca-se pela portabilidade entre plataformas. A Netscape foi a primeira empresa a licenciar a linguagem para utilizá-la em seu *browser* Netscape Navigator. Isto quer dizer que as *Applets* vão rodar tão bem no *browser* Netscape Navigator a partir da versão 2, em sistema operacional Windows 95, quanto rodam no *browser* HotJava em sistema operacional Solaris da Sun Microsystems ([HOF1996]).

3.2.1 HISTÓRIA

Segundo [HOF1996], a linguagem de programação *Java* foi projetada e implementada em dezembro de 1990 por uma pequena equipe de pessoas coordenadas por James Gosling na empresa Sun Microsystems, localizada em Mountain View, na Califórnia.

A primeira utilização da linguagem *Java* foi no projeto *Green*. Este projeto tinha o propósito de estudar um novo tipo de *interface* com o usuário para controlar aparelhos domésticos ([PER1996]). Foi construído um computador experimental de mão, que recebeu o nome código de “*7” (Star Seven). Este computador permitia manipular os aparelhos domésticos através de toques na tela. O plano original era desenvolver o sistema operacional do Star Seven em C++, mas o grande número de problemas originados das características da linguagem levaram James Gosling a desenvolver uma linguagem de programação que era melhor para os propósitos do projeto *Green* que a linguagem C++. Ele chamou a nova linguagem de Oak, que significa carvalho em inglês, em homenagem a um grande carvalho que havia na frente da janela de seu escritório na Sun Microsystems. Posteriormente este nome foi alterado para *Java*, pois descobriu-se que Oak era o nome de uma outra linguagem de programação anterior à linguagem da Sun.

O anúncio oficial da tecnologia *Java* foi feito em maio de 1995 na conferência SunWorld em San Francisco (EUA).

3.2.2 CARACTERÍSTICAS

As seguintes características identificam a linguagem *Java*:

c) Client-Side: são executadas pelo navegador na estação cliente;

d) compilada: a *Applet* é compilada para o chamado "*byte-code*", que é próximo às instruções de máquina, mas não de uma máquina específica. O "*byte-code*" é um código de uma máquina virtual idealizada pelos criadores da linguagem. Esta característica leva o código *Java* a ser mais rápido do que se fosse simplesmente interpretada;

e) portável: a linguagem *Java* foi criada para ser portável. O "*byte-code*" gerado pelo compilador para a aplicação específica pode ser transportado entre plataformas distintas que suportam *Java*. Não é necessário recompilar um programa para que ele rode numa máquina e sistema diferente;

f) erros: a presença de coleta automática de lixo (*garbage collector*) evita erros comuns que os programadores cometem quando são obrigados a gerenciar diretamente a memória. A eliminação do uso de ponteiros, em favor do uso de vetores, objetos e outros, traz benefícios em termos de segurança. O programador é proibido de obter acesso a memória que não pertence ao seu programa, além de não ter chances de cometer erros de endereçamento de memória. Possui mecanismos de tratamento de exceções, tornando as aplicações mais robustas, não permitindo que elas abortem, mesmo quando rodando sob condições anormais;

g) orientada a objeto: *Java* dá suporte a uma hierarquia de classes de herança simples, onde cada classe só pode herdar de uma outra classe por vez. Na raiz da hierarquia de classes está sempre a classe *Object* (objeto);

h) segurança: para evitar que a execução de uma *Applet* possa implicar em riscos para o usuário que o executa, os programas em *byte-code* possuem informações de tipo adicionais, além de referenciar métodos e variáveis pelo nome, diferente de outras linguagens que se utilizam de endereçamento numérico. Desta forma, é possível executar a verificação dos *byte-codes* pelo navegador para verificar se as restrições da linguagem *Java* estão sendo respeitadas. A *Applet* também é executado em um

ambiente restrito e necessita de autorizações para executar determinadas funções consideradas perigosas. Em geral, há várias restrições sobre o que uma *Applet* está habilitado a realizar durante seu processamento, tais como:

- não podem manipular o disco local;
- não podem criar conexões de rede, exceto com o local de onde se originaram;
- têm acesso restrito para interagir com o sistema operacional.

3.2.3 HOSPEDAGEM EM HTML

A hospedagem da *Applet* no documento HTML é possível através da *tag Applet*. No Quadro 10 é apresentado um exemplo da *tag Applet* especificando o arquivo "AloMundo.class", alguns atributos de formatação (alinhamento, tamanho da área e grossura da borda), a passagem do valor "1" para o parâmetro "ParamExe_1" e um comentário da *tag Applet*. O comentário é exibido no lugar da *Applet* no caso do navegador reconhecer a *tag Applet*, mas não estar habilitado ou capacitado a executar a mesma ([VEN1996]).

Quadro 10 - Exemplo da sintaxe da *tag Applet.*

```
<p>
<Applet code="AloMundo.class" align="top"
hspace="2" vspace="2" width="150" height="100">
<param name="ParamExe_1" value="1">
Seu Browser não suporta Java Applet
</Applet>
</p>
```

As seguintes especificações de atributos podem ser utilizadas na *tag Applet*:

a) *codebase*: este atributo opcional especifica o *Uniform Resource Identifiers* (URI) que contém o código compilado. Se este atributo não é especificado, então é considerado o URI do documento HTML atual;
b) *code*: este atributo obrigatório especifica o nome do arquivo que contém o código compilado. Este arquivo é relativo ao URI base da *Applet*;
c) *name*: este atributo opcional especifica um nome para a instância da *Applet*, tornando possível a comunicação entre *Applets*;
d) *width* / *height*: estes atributos obrigatórios especificam a largura e altura iniciais (em pixels) da área de exibição da *Applet*;

e) *align*: este atributo obrigatório especifica o alinhamento da *Applet*. Os valores possíveis para este atributo são: *left, right, top, texttop, middle, absmiddle, baseline, bottom e absbottom;*

f) *vspace* / *hspace*: estes atributos opcionais especificam o número de pixels de borda vertical e horizontal, respectivamente, da *Applet*.

A passagem de parâmetros para a *Applet* é feita através da *tag* "param", que possui os seguintes atributos:

a) *name*: este atributo obrigatório especifica o nome do parâmetro;

b) *value*: este atributo obrigatório especifica o valor do parâmetro;

c) *value type*: este atributo opcional especifica o tipo de valor do atributo *value* e pode representar:

- *data*: este é o valor padrão, indica que o valor é do tipo *string;*
- *ref*: indica que o valor é uma URI;
- *object*: indica que o valor é um objeto.

3.2.4 OPERADORES E CONTROLES ESPECIAIS

Com a linguagem *Java* podem ser utilizados os operadores condicionais listados na tabela 1.

Tabela 1 - Operadores condicionais da linguagem *Java*

Operador	Exemplo	Significado
&&	x1 && x2	verdadeiro se x1 e x2 são ambos verdadeiros, senão falso
||	x1 || x2	verdadeiro se x1 ou x2 são verdadeiros, senão falso
!	!x1	negação de x1. Verdadeiro se x1 é falso, senão falso.

Os operadores relacionais da linguagem *Java* podem ser vistos na tabela 2.

Tabela 2 - Operadores relacionais da linguagem *Java*

Operador	Exemplo	Significado
>	x1 > x2	verdadeiro se x1 maior que x2, senão falso
>=	x1 >= x2	verdadeiro se x1 maior ou igual a x2, senão falso
<	x1 < x2	verdadeiro se x1 menor que x2, senão falso
<=	x1 <= x2	verdadeiro se x1 menor ou igual a x2, senão falso
= =	x1 = = x2	verdadeiro se x1 igual a x2, senão falso
!=	x1 != x2	verdadeiro se x1 diferente de x2, senão falso

Os operadores matemáticos da linguagem *Java* podem ser vistos na tabela 3.

Tabela 3 - Operadores matemáticos da linguagem *Java*

Operador	Exemplo	Significado
+	x1 + x2	retorna a soma de x1 por x2
-	x1 - x2	retorna a subtração de x1 por x2
*	x1 * x2	retorna a multiplicação de x1 por x2
/	x1 / x2	retorna a divisão de x1 por x2
%	x1 % x2	retorna o resto da divisão de x1 por x2
++	x1++	incrementa x1 em uma unidade depois do processamento atual
++	++x1	incrementa x1 em uma unidade antes do processamento atual
--	x1--	decrementa x1 em uma unidade depois do processamento atual
--	--x1	decrementa x1 em uma unidade antes do processamento atual

Os controles especiais da linguagem *Java* podem ser vistos na tabela 4.

Tabela 4 - Controles especiais da linguagem *Java*

Representação visual	Função ou significado
\n	pula linha, *linefeed*
\r	retorno de carro
\b	*backspace*
\t	tabulação
\f	*formfeed*
\’	apóstrofe
\”	aspas
\\	barra inversa
\u223d	caractere unicode
\gfa	octal
\fff	hexadecimal

3.2.5 CONTROLE DE FLUXO

Os comandos condicionais são utilizados para controlar a execução de um grupo específico de instruções, formando estruturas para tomada de decisão ou estruturas de repetição na linguagem.

O comando *if* é utilizado para tomada de decisão de acordo com algum critério. Caso a condição seja verdadeira será executado o bloco de comandos especificado logo após a cláusula *then*. Caso a condição seja falsa será executado o bloco de comandos especificado logo após a cláusula *else*. Se a cláusula *else* não estiver presente - por ser opcional - segue a execução normal do programa após o comando. A sintaxe do comando *if* está descrita no quadro 11.

Quadro 11 - Sintaxe do comando *if* da linguagem *Java*

```
If (condição) then
  < bloco_1 >
[else
  <bloco_2>]
```

O comando *switch* é utilizado para tomada de decisão de acordo com valor. O comando desvia a execução para a cláusula *case* que tiver o mesmo valor que o definido após a cláusula *switch*. Cada valor da cláusula *case* deve ser único e o tipo de dado deve ser o mesmo que o definido na cláusula *switch*. Depois de cada caso é necessário utilizar a instrução *break*, caso contrário todas as cláusulas posteriores também serão executadas. A cláusula *default* é opcional e será executada quando nenhuma outra cláusula do comando *switch* for correspondida. A sintaxe do comando *switch* está descrita no quadro 12.

Quadro 12 - Sintaxe do comando *switch* ... *case* da linguagem *Java*

```
(...)
switch (x1) {
  Case 1: <Bloco Instruções> break;
  Case 5: <Bloco Instruções> break;
 default: <Bloco Instruções> break;
}
(...)
```

O comando while é utilizado como uma estrutura de repetição. Os comandos do bloco de instruções definidos após a cláusula *do* serão executados até que a condição seja falsa. A avaliação lógica do comando é feita no final do bloco de instruções, o que significa que os comandos do bloco sempre são executados no mínimo uma vez. O comando *while* apresenta uma variação onde a avaliação da condição é executada antes do bloco de instruções sem definir a cláusula *do*. A sintaxe do comando *while*, em suas duas formas, está descrita no quadro 13.

Quadro 13 - Sintaxe do comando *while* da linguagem *Java*

```
Do
  <Bloco Instruções>
while (condição)


While (condição)
<Bloco Instruções>
```

O comando *for* é utilizado como uma estrutura de repetição. Ele é composto de três parâmetros:

a) parâmetro 1: inicializa a variável de controle do *loop*;
b) parâmetro 2: condição de avaliação para finalizar o *loop*. Quando retorna falso o *loop* é finalizado;
c) parâmetro 3: indica o valor de incremento/decremento da variável de controle do *loop*.

A avaliação lógica do comando é feita no início do bloco, sendo possível que os comandos especificados no bloco de instruções não sejam executados nenhuma vez durante o processamento. A sintaxe do comando *for* está descrita no quadro 14.

Quadro 14 - Sintaxe do comando *for* da linguagem *Java*

```
(...)
// criação e alocação do array
a[] = new int[10];
int i;
// irá zerar os elementos do array (de 0 a 9)
For (i=0; i < 10; i++)
  a[i] = 0;
(...)
```

3.2.6 VARIÁVEIS

As variáveis em *Java* podem ser declaradas em qualquer ponto do código especificando o tipo e o nome da variável. Num esforço para tornar o sistema *Java* mais simples os tipos de dados *int*, *char*, *byte* e *boolean* não são classes, entretanto todos esses tipos simples podem ser especificados como objetos. As variáveis em *Java Applet* podem ser inicializadas no momento de sua declaração ([STH1996]). O *array* é um tipo fundamental na linguagem *Java* e guarda vários objetos que podem ser referenciados por indexação que sempre inicia em zero como pode ser visto no quadro 14. A alocação de um vetor sempre usa o operador *new* para criar o vetor e atribuí-lo a uma variável ([RIT1998]). Na tabela 5 são mostrados os tipos de variáveis em *Java*.

Tabela 5 - Tipos de variáveis da linguagem *Java*

Tipo	Tamanho	Faixa
Byte	8-bit	-128 até 127
Short	16-bit,	-32768 até 32767
Int	32-bit,	-2147483648 até 2147483647
Long	64-bit,	-9223372036854775808 até 9223372036854775807
Boolean	1 - bit	valor true ou false
Char	16-bit	utiliza o padrão UNICODE para representar os caracteres
Float	32-bit	valor com ponto flutuante, norma IEEE754, precisão simples
Double	64-bit	valor com ponto flutuante, norma IEEE754, precisão dupla

3.2.7 O PACOTE APPLET

Além das classes que permitem que se agrupem campos e métodos, *Java* tem uma construção chamada *package* (pacote) que permite agrupar classes relacionadas. Classes selecionadas deste pacote podem ser usadas por outras classes que não estejam no pacote através do comando *import*, ou pode-se também, utilizar as classes colocando explicitamente o nome completo do pacote antes da classe. O uso da declaração *import* é um atalho que torna opcional a prefixação do nome do pacote na frente de cada uma das classes do pacote ([RIT1998]). Os pacotes padrões da linguagem *Java* são: Java.lang, Java.io, Java.util, Java.net, Java.awt e Java.Applet.

Ao contrário das aplicações tradicionais na linguagem *Java*, uma *Applet* escrita em *Java* não define explicitamente um método inicial de execução do programa. As *Applets* executam especificamente a partir dos métodos e eventos definidos na classe *Applet*. No quadro 15 pode ser visto um exemplo simples de um programa *Java Applet*.

Quadro 15 - Código fonte de um programa simples da linguagem *Java*

```
// Applet exemplo AloMundo
import Java.awt.Graphics;

public class AloMundo extends Java.Applet.Applet{
     public void paint(Graphics g) {
          g.drawString("Alô Mundo !!",1,1);
     }
}
```

3.2.7.1 PRINCIPAIS EVENTOS

Os principais eventos disponíveis, implementados na classe *Applet* são:

a) *init*(): este método é invocado quando a *Applet* é carregado pela primeira vez pelo navegador;

b) *destroy*(): este método é invocado quando a *Applet* está para ser eliminado do navegador;

c) *start*(): este método é invocado quando a *Applet* inicia, tornando-o ativo;

d) *stop*(): este método é invocado quando a *Applet* fica em estado estático.

Os principais eventos implementados na classe *Applet* pode ser visto no quadro 16.

Quadro 16 - Exemplo das chamadas dos eventos da linguagem *Java*

```
Import Java.Applet.Applet;
Import Java.awt.Graphics;

Public class Simples extends Applet {
 StringBuffer buffer = new StringBuffer();

 Public void init() {
  Acrescenta("inicializando... ");  }

 Public void start() {
  Acrescenta("começando... ");  }

 Public void stop() {
  Acrescenta("parando... ");  }

 Public void destroy() {
  Acrescenta("preparando para descarregar..."); }

 Void acrescenta(String palavra) {
  System.out.println(palavra);
      buffer.append(palavra);
       repaint();
     }

 public void paint(Graphics g) {
  g.drawRect(0, 0, size().width - 1, size().height - 1);
      g.drawString(buffer.toString(), 5, 15);
    }
}
```

3.2.7.2 PRINCIPAIS MÉTODOS

Os principais métodos implementados na classe *Applet* que permitem interagir com o navegador e com o documento HTML são:

a) getdocumentbase(): retorna o *Universal Resource Locator* (URL) do documento que contém a *Applet*. A classe URL retornada está no pacote Java.net e tem diversos métodos que permitem manipular e ler o conteúdo do URL;
b) getcodebase(): retorna o URL da *Applet* em si, que pode ser distinto do URL do documento onde ela está contida;
c) getparameter(): retorna o valor de um parâmetro nomeado no descritor HTML através da *tag* <PARAM>;
d) showstatus(): exibe uma mensagem na barra de *status* dos navegadores que executam a *Applet*.

3.2.8 TRATAMENTO DE EXCEÇÕES

Segundo [RIT1998], uma exceção em *Java* será invocada no momento em que for verificada uma condição anormal de funcionamento do programa, como um acesso a um índice inválido de um vetor, tentativa de uso de um objeto não inicializado, falha na transferência de uma informação, entre outros. Então o método que está sendo executado é imediatamente interrompido e o controle passa para o método que o chamou, onde pode ocorrer o tratamento da exceção. Caso não seja feito o tratamento da exceção o controle será propagado por todos os métodos invocados, respectivamente, até o método de mais alta ordem onde se não tratado ocorrerá o término anormal do programa. O comando *try* (tentativa) precede um bloco de texto onde serão verificadas as exceções. A cláusula *catch* (capturar) inicia o bloco de tratamento da exceção gerada. É possível implementar várias cláusulas *catch* para tratar exceções diferentes que possam ocorrer. Se existir algum código que deva ser executado mesmo que uma exceção ocorra ou não, deverá ser utilizado a cláusula *finally* (finalização). O comando *throw* é utilizado para propagar uma nova exceção. No quadro 17 pode ser visto um código em *Java* que gera uma exceção de divisão por zero tratada pelo comando *try*, a implementação de um bloco protegido pela cláusula *finally* e a propagação de uma nova exceção através do comando *throw*.

Quadro 17 - Exemplo do tratamento de exceções em *Java*

```
(...)
class exceptionTest {
  public static void main(String args[]) {
    try {
      int i = 0;
      i = i / 0;
      //outros códigos aqui não serão executados

    } catch (ArithmeticException e) {
      //tratamento da divisão por zero
      System.out.println('Tentativa de dividir por zero');

    } catch (Exception e) {
      //Caso ocorra alguma outra exceção será tratada aqui
      System.out.println('Erro deconhecido');
    }
  }
}
try {
  i = i / 0;
} finally {
  //será executado ocorrendo exceção ou não
  i = 0 }
// Gera uma nova exceção no sistema (não tratada)
throw new newException();
(...)
```

3.2.9 MULTIPROCESSAMENTO

A maior parte dos sistemas operacionais modernos é multitarefa, ou seja, compartilham o uso do processador em mais de uma tarefa ao mesmo tempo. *Java* tem o suporte a multitarefa embutido na linguagem. Um programa em *Java* pode ter mais de uma *thread* (que significa linha de execução em inglês) executando ao mesmo tempo se o sistema operacional der suporte a essa característica. Segundo [HOF1996], linha de execução é o conjunto de todos os estados necessários para um processamento. O processador executa em pequenos intervalos de tempo cada uma das linhas de execução, resultando na sensação de que estas linhas estejam sendo executas simultaneamente ou em paralelo.

Existem casos em que é muito interessante implementar o uso de uma *thread*, por exemplo, alguns navegadores não continuam a exibição da página enquanto a *Applet* não acabar sua inicialização. Caso a inicialização da *Applet* consuma muito tempo, poderia ser criada uma *thread* para este processamento.

Para uma classe ter suporte ao multiprocessamento é necessário implementar a *interface Runnable*. *Interface* é um conjunto de métodos que não contém código de

implementação. A *Runnable* é uma *interface* encontrada no pacote Java.lang. Qualquer classe que implemente a *Runnable* deve definir um método *run*. No quadro 18 pode ser visto um exemplo da implementação de uma *thread* em *Java Applet*.

Quadro 18 - Exemplo de *Applet* que implementa *Thread*

```
// --------- Classe que implementra a thread ---------
public class SorteioThread extends Thread {
  int num_esc;
  int num_sorteado;

  // método contrutor
  public SorteioThread(int parnumero) {
    num_esc = parnumero;
  }

  // O metodo que vai ser executado no thread
  public void run() {
    // Sorteia um número aleatório até ser igual ao parâmetro (num_esc)
    while (num_esc != num_sorteado) {
      System.out.println("Numero escolhido: "+num_esc+", Numero
sorteado:"+num_sorteado);
      num_sorteado =  (int) Math.round(Math.random()*10);
      // suspende a thread (intervalo)
      sleep(50);
    }
    System.out.println("*** Acertou numero:"+num_esc);
  }
}

// --------- Classe principal. ---------
Class SorteioPrinc
{
   public static void main (String args[])
   {
    // declara duas threads
    SorteioThread a,b;
    // chamada do construtor da thred passando um inteiro como parâmetro
    a=new SorteioThread(2);
    // inicia execução da thread
    a.start();
    b=new SorteioThread(7);
    b.start();
    // aguarda  o fim da execução da thread
    try {a.join(); } catch (InterruptedException ignorada) {}
    try {b.join(); } catch (InterruptedException ignorada) {}
    System.out.println("fim do sorteio");
   }
}
```

Entre os métodos da classe *Thread* pode-se citar:

a) sleep(long milisegundos): suspende a execução da *thread* em milisegundos;

b) start(): inicia a execução da *thread*, chamando o método run();

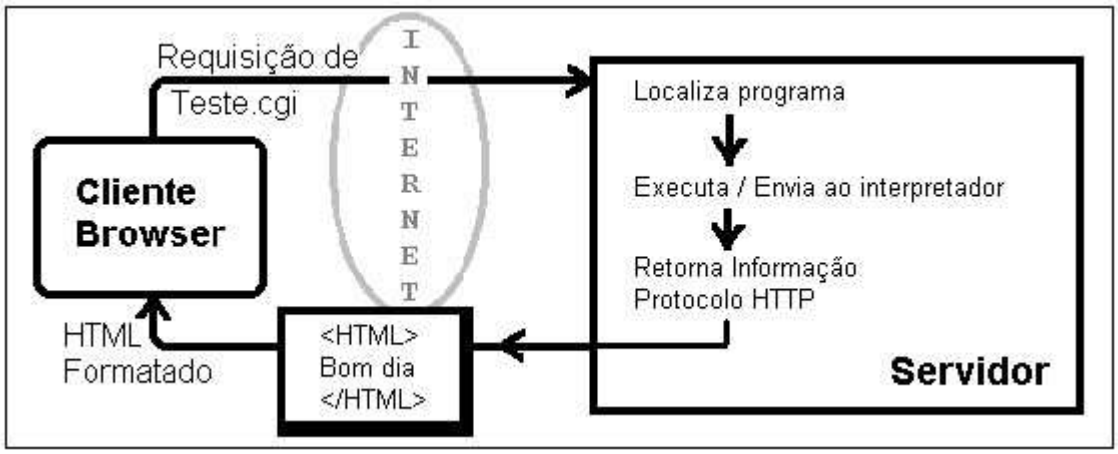
c) run(): bloco principal da *thread* que deverá conter as rotinas que devem ser executadas quando a *thread* estiver ativa;

d) stop(): para a execução da *thread*.

3.3 COMMON GATEWAY INTERFACE

Segundo [BUT1997], A *Common Gateway Interface* (CGI) foi especificada no protocolo de comunicações HTTP a fim de designar um meio para que a programação do lado do servidor atuasse como interface com o servidor *Web*. Assim como a HTML no lado do cliente, a CGI pretende ser independente do software servidor e do sistema operacional do lado do servidor. Estes programas podem ser escritos nas mais diversas linguagens suportadas pelo servidor, como por exemplo Perl, *Visual Basic Script*, C++, entre outras. Um programa CGI será responsável por retornar as informações para o cliente, respeitando o protocolo HTTP. Segundo [HER1997] a programação CGI constitui a criação de aplicações que atuam como programas de interface ou *gateway* (transição) entre o navegador cliente, o servidor da *Web* e uma aplicação tradicional de programação. Essas aplicações são conhecidas como *Scripts* de CGI porque, historicamente, muitas destas eram escritas em linguagens de *script* como Perl. Na figura 1, pode ser visto o esquema geral de uma solicitação em CGI. Inicialmente o cliente faz a requisição dos dados, o servidor localiza o programa e o executa ou envia para um compilador executar. Deste processamento devem ser gerados cabeçalhos de resposta válidos conforme protocolo HTTP que será enviado ao cliente como resposta ao pedido.

Figura 1 - Esquema geral das solicitações CGI



Fonte: [UNI2000].

3.3.1 CARACTERÍSTICAS

Uma aplicação CGI possui as seguintes características:

a) pode ser escrita em mais de uma linguagem de programação, dando ao desenvolvedor a flexibilidade para escolher a linguagem que melhor lhe convém;
b) o desenvolvedor de uma aplicação CGI tem que ter uma preocupação a mais com relação aos dados retornados ao cliente estarem respeitando o protocolo HTTP;
c) CGI habilita o cliente, sem se preocupar com sua plataforma, para passar e receber informações do *script* de CGI, pois o processamento ocorre no servidor e o cliente apenas recebe os dados como se fossem uma página estática da internet;
d) abstrai sua complexidade do usuário, pois todo código e processamento é realizado no servidor e o cliente somente recebe uma página HTML padrão.

3.3.2 MÉTODOS DE SOLICITAÇÃO DA CGI

Um programa CGI é solicitado de forma semelhante a uma página estática da internet, com a diferença de que podem ser passados parâmetros para o programa CGI através do *Uniform Resource Identifiers* (URI). A URI, também chamada de *Universal Resource Locator* (URL) ou *Universal Resource Name* (URN) é um endereço eletrônico da internet que é composto basicamente pelos campos:

a) protocolo: o campo protocolo identifica o protocolo da internet que será utilizado na transferência de dados entre o servidor e o cliente. Há uma série de tipos de protocolo válidos para a internet como *FTP, WAIS, GOPHER, TELNET, HTTP* entre outros;
b) nome do domínio: logo após o protocolo e os indicadores “://” encontra-se o nome do domínio. O nome do domínio é o endereço de seu servidor na Internet. Este nome ou endereço fica localizado entre os indicadores “://” e a barra “/” seguinte. Após o nome de domínio e antes da barra finalizadora é opcional especificar o número da porta, por padrão igual a 80;
c) o diretório, o arquivo ou o programa CGI: após a primeira barra (/) encontra-se o caminho e o nome do arquivo destino no servidor internet. Se o último elemento for um arquivo, esse será retornado ao cliente. Se for um programa CGI, tal programa será executado e os dados por ele gerados serão retornados ao cliente;

d) path_info: logo após a identificação do nome do programa CGI é opcional a existência de uma informação adicional de caminho. Esta informação será interpretada pelo servidor internet e estará disponível numa variável de ambiente chamada Path_Info e que poderá ser utilizada pelo programa CGI;
e) query_string - a informação de query_string está separada das demais informações do URI pelo caracter especial “?” (interrogação). As informações da query_string estão organizadas em pares na forma nome_campo=valor_campo e são separadas umas das outras pelo caracter especial “&” (*ampersand*). Estas informações estarão disponíveis numa variável de ambiente chamada query_string e podem ser consideradas como parâmetros para a execução do programa CGI.

No quadro 19 pode ser observado um exemplo do URI de solicitação, informando o protocolo “HTTP”, o endereço “192.168.0.217”, o diretório destino “/TCC/CGI/”, o programa “Hora_Perl.cgi”, onde a informação “/inf_adicional/” corresponde ao Path_info e a informação “nome=++2espacos&interrogacao=%3F” corresponde ao Query_String.

Quadro 19 - Exemplo de URI de solicitação de um programa CGI

```
Http://192.168.0.217/TCC/CGI/Hora_Perl.cgi/inf_adicional/?nome=++2espacos&
interrogacao=%3F
```

Existem alguns caracteres especiais que necessitam ser codificados no URI para poderem ser transmitidos como informação. O sinal “%” (porcentagem) é utilizado para codificar estes caracteres em seu correspondente hexadecimal. Os dois caracteres após o símbolo % representam o número hexadecimal, conforme o padrão ASCII, do caracter especial. Um exemplo pode ser visto no quadro 19, onde os caracteres %3F serão convertidos pelo servidor como o caracter especial “?”. Os espaços em branco nos dados de valores representam um caso especial e são substituídos pelo caracter “+” (adição). A Tabela 6 lista os caracteres ASCII que devem ser codificados com o URI e o seu respectivo valor em hexadecimal.

Tabela 6 - Caracteres especiais do URI

Caracter	Valor Hex.	Caracter	Valor Hex.	Caracter	Valor Hex.
Tabulação	09	**:**	3A	**|**	7C
?	3F	<	3C	**~**	7E
"	22	>	3E	**/**	2F
(	40	**@**	40	**#**	23
)	29	**[**	5B	**}**	7D
,	2C	\	5C	**&**	26
.	2E	**]**	5D	**=**	3D
;	3B	^	5E	**%**	25
`	60	**{**	7B		

Fonte: [STH1996]

3.3.2.1 UTILIZAÇÃO DE FORMULÁRIOS

Segundo [RAM1997], um dos recursos interessantes da linguagem HTML é a possibilidade de criar formulários eletrônicos. Por meio de um formulário eletrônico o usuário pode interagir com o servidor enviando dados que serão processados no servidor e eventualmente devolvidos ao cliente. O *tag* Form é um dos principais responsáveis pela viabilização de troca de informações entre o cliente e o servidor. Este comando tem duas funções importantes: especificar o local do programa que controlará o formulário e especificar a forma ou método como os dados serão enviados. Um exemplo da *tag* Form pode ser visto no quadro 20.

Quadro 20 - Sintaxe da *tag* Form

```
<html><body>
<form method="post" action="Http://192.168.0.217/TCC/CGI/Hora_Perl.cgi">
informe seu nome:<input type=text size=10 name="ednome:"><br><br>
<input type="submit" value="Botão Consultar">
</form>
</body></html>
```

Os principais parâmetros da *tag form* são:

a) *action*: especifica o local (URL) do servidor e do programa CGI que vai processar o *form*;
b) *method*: indica o método de troca de dados usado pelo servidor para receber o *form*. As opções disponíveis para este são:
   - *get*: o método *get* envia os dados codificados nos formulários anexados ao *string* do URI;
   - *post*: o método *post* também codifica os dados do formulário. Entretanto, os dados serão enviados após a transmissão ao servidor de todos os cabeçalhos de solicitação.

Através do método *Get* as informações dos campos do formulário estarão visíveis no URI, ou seja, irão aparecer no campo endereço do navegador, enquanto que o método *Post* irá incluir os dados após os cabeçalhos de solicitação do protocolo HTTP informando juntamente o tamanho do buffer de dados ([MCF1998]). Através do método *Post* tem-se as vantagens de não limitar a quantidade de dados enviadas ao servidor ao *buffer* de entrada de dados e não aparecer os dados no campo endereço do navegador.

3.3.3 O PROTOCOLO HTTP

O protocolo do HTTP é formado por instruções enviadas entre cliente e servidor. Estas instruções são denominadas de cabeçalhos do HTTP e formam o protocolo utilizado entre o servidor Internet e o navegador para que eles possam comunicar-se. Os cabeçalhos de HTTP são enviados tanto pelos clientes como pelos servidores, podendo ser cabeçalhos de solicitação ou de resposta. Há apenas alguns poucos cabeçalhos utilizados exclusivamente pelo servidor, os quais são sempre cabeçalhos de resposta.

3.3.3.1 O CABEÇALHO DE SOLICITAÇÃO E RESPOSTA DE MÉTODOS

O cabeçalho de solicitação completa do método é o primeiro cabeçalho de solicitação enviado juntamente com qualquer solicitação de cliente. A linha de solicitação completa de método é composta de três partes separadas por espaços: o tipo do método, o URI solicitado e

número de versão do HTTP. No quadro 21 pode ser visto a sintaxe e um exemplo do cabeçalho.

Quadro 21 - Exemplo e Sintaxe do cabeçalho de solicitação do HTTP

```
Sintaxe:
Request_Method URI HTTP_Protocol_Version \n

Exemplo:
GET http://accn.com/inde4x.HTML HTTP/1.0
```

Os seguintes parâmetros são válidos para o cabeçalho de solicitação:

a) request_method: pode ser qualquer um dos seguintes tipos de métodos: *Get, Post, Head, Put, Delete, Link* ou *Unlink;*
b) URI: representa o endereço do arquivo, programa ou diretório que se pretende acessar;
c) HTTP_Protocol_Version: representa o número de versão do protocolo HTTP que possa ser acessado pelo cliente/navegador.

Os cabeçalhos de resposta representam a resposta do servidor à solicitação de URI feita pelo cliente. Sua sintaxe e exemplo podem ser vistos no quadro 22.

Quadro 22 - Exemplo e Sintaxe do cabeçalho de resposta do HTTP

```
Sintaxe:
PROTOCOL/Version_Number Status_Code Status_Description

Exemplo:
HTTP/1.0 200 Ok
```

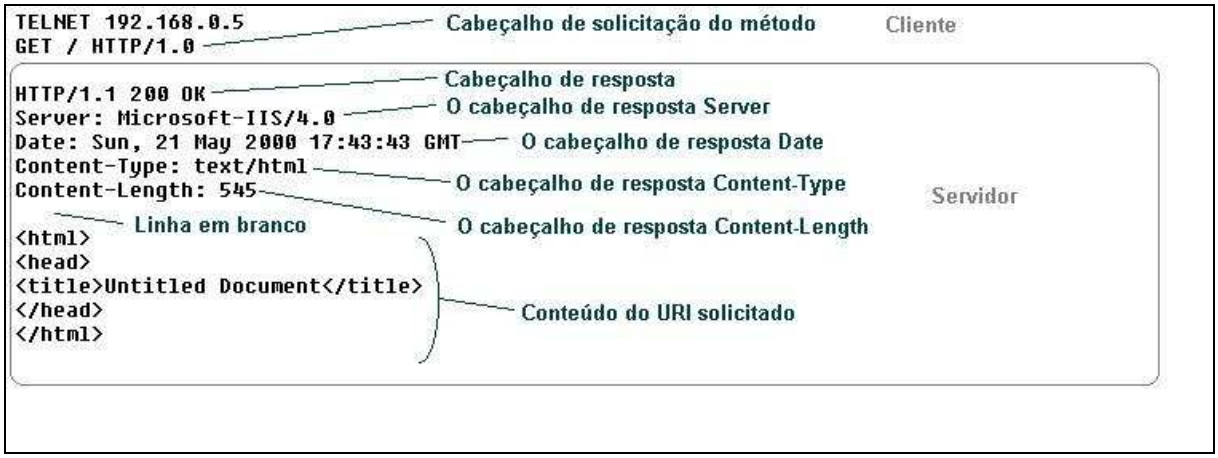
Os códigos de status existentes nos cabeçalhos de resposta informam ao cliente o grau de sucesso obtido quanto a solicitação do URI. Os principais códigos de status são:

a) código de *status* de informação estão na faixa dos números 100 e são de caráter informativos;
b) códigos de *status* de sucesso estão na faixa dos números 200 e são indicadores de sucesso;
c) códigos de *status* de redirecionamento estão na faixa dos números 300 e indicam a alteração de local do *Uniform Resource Identifiers*;

d) códigos de erro do cliente estão na faixa do número 400 e indicam que a solicitação não obteve sucesso devido a um problema relativo ao cliente;
e) códigos de erro do servidor estão na faixa do número 500 e indicam que a solicitação não obteve sucesso devido a um problema relativo ao servidor.

No figura 2 é exibida uma listagem dos comandos do protocolo HTTP enviados e recebidos em uma conexão.

Figura 2 - Exemplo dos comandos do HTTP



Os seguintes cabeçalhos de resposta podem ser identificados na figura 2:

a) o cabeçalho de solicitação completa do método;
b) o cabeçalho de resposta do método;
c) o cabeçalho *Server*: identifica o software de servidor internet utilizado pelo servidor;
d) o cabeçalho Date: identifica a data/hora no padrão *Greenwich Mean Time* (GMT) no momento de início da transferência de dados;
e) o cabeçalho Content-Type: identifica o tipo de dados em transferência;
f) o cabeçalho Content-Lenght: identifica o tamanho da transferência de dados em *bytes* decimais;
g) conteúdo do URI: as informações solicitadas, geralmente um documento HTML.

Antes de iniciar a execução de um programa CGI, o servidor da Internet cria um ambiente especial de processamento para a operação do programa CGI. O ambiente inclui a tradução de todos os cabeçalhos de solicitação do protocolo HTTP para variáveis de

ambiente, as quais podem ser utilizadas pelo programa CGI. Além de informações de sistema (como a data atual), o ambiente inclui informações sobre quem está chamando o CGI, de onde o programa está sendo chamado e possivelmente até mesmo informações de estado (o que foi executado pelo programa na última vez que ele foi chamado) que ajudem a controlar as ações de cada visitante da *Web*.

O programa CGI, a HTML e o HTTP precisam operar de forma coordenada a fim de que a aplicação *on-line* para a internet funcione de forma apropriada. O código em HTML define a forma com que o usuário visualiza a interface do programa, sendo também responsável pela coleta de dados inseridos pelo usuário. Isto é freqüentemente denominado "código da interface humana com o computador"; trata-se da janela onde o programa e o usuário interagem mutuamente. O HTTP representa o mecanismo de transporte para o envio de dados entre o programa CGI e o usuário. Ele é o diretor oculto que traduz e envia as informações entre o cliente da *Web* e o programa CGI. O programa CGI será responsável por entender as instruções do HTTP e as solicitações do usuário. O programa CGI recebe as solicitações feitas pelos usuários e retorna respostas válidas e úteis ao cliente da *Web* que está dando cliques na sua página HTML da *Web* ([HER1997]).

3.4 A LINGUAGEM PERL

O nome PERL é a abreviatura de *Practical Extration and Report Language* (Linguagem Prática de Extração e Relatório). Perl é uma linguagem de programação de alto nível, que combina elementos característicos do C e do shell do Unix - de onde foi originada - para criar uma ferramenta poderosa de processamento de textos e relatórios genéricos. Pelo fato de muitos dos processamentos em aplicações CGI necessitarem a manipulação de texto, Perl é uma linguagem muito utilizada para o processamento destes *Scripts* ([STH1996]).

3.4.1 HISTÓRIA

Perl nasceu em 1987, mas apenas recentemente houve um interesse em sua documentação e especificação. Em 18 de outubro de 1987 a primeira versão de Perl foi liberada para o grupo de discussão comp.sources. Em janeiro de 1992 a primeira versão de Perl para a arquitetura Macintosh, denominada MacPerl 4.0.2 é liberada por Matthias Neeracher. Em 1989 é distribuida por Larry Wall pela primeira vez segundo os termos da

General Public License (GNU) que é uma licença de livre distribuição de software ([ASH2000]).

3.4.2 CARACTERÍSTICAS

As seguintes características identificam a linguagem PERL:

a) possui recursos que permitem manipular mais facilmente documentos texto, gerar relatórios e manipular arquivos;
b) *strings* em Perl não tem limitação de tamanho. O conteúdo inteiro de um arquivo pode ser utilizado como uma simples *string*;
c) a sintaxe em Perl é similar a da linguagem C. Muitas construções são utilizadas de maneira idêntica a linguagem C;
d) a sintaxe da linguagem é sensível ao caso, ou seja, um caracter maiúsculo é diferente do mesmo em minúsculo.

3.4.3 VARIÁVEIS

Segundo [HER1997] um dos recursos poderosos da linguagem Perl é a capacidade de distinguir os diversos nomes de variáveis com base no caracter inicial da variável. Todas as variáveis em PERL iniciam com um cifrão ($), um sinal de arroba (@) ou um sinal de porcentagem (%). Pode-se utilizar também o *umpersand* (&) para iniciar chamadas a sub_rotinas. O asterisco (*) é um coringa e se refere a qualquer variável. Os identificadores de tipos de variáveis podem ser vistos na tabela 7:

Tabela 7 - Identificadores de tipos de variáveis em PERL

Caracter	Tipo de dado
$	refere-se a *string* ou números. A própria linguagem Perl se encarrega do tratamento, dependendo do contexto, da variável como número ou como *string*
@	se refere a *arrays* indexados por números
%	se refere a *arrays* indexados por *strings*. Um *array* indexado por *string* tem as mesmas características de um *array* normal, mas cada referência a um elemento do *array* é feita por uma *string* e não por um número. Perl denomina tais elementos como *arrays* associativos.

O caracter sustenido “#” indica que as palavras da sua direita até o final da linha são comentários e devem ser desconsideradas, mas o conjunto de caracteres “#!” - quando iniciados na primeira coluna - representam uma diretiva especial destinada ao pré-processador. Essa diretiva pode indicar, por exemplo, o caminho onde o interpretador da linguagem Perl está localizado.

Existem algumas variáveis especiais na Perl que são chamadas de *handles* (apelido), Segundo [FOG1997] um *handle* de arquivo pode ser considerado como um ponteiro para um arquivo que a Perl deve ler, ou no qual deve gravar. A idéia básica é associar um *handle* a um arquivo ou dispositivo e, depois se referir ao *handle* sempre que for preciso executar uma operação de leitura ou gravação. A Perl possui alguns *handles* predefinidos:

a) STDIN: significa *standard in* (entrada padrão), normalmente o teclado;
b) STDOUT: significa *standard out* (saída padrão), normalmente a tela;
c) STDERR: significa *standard error* (erro padrão), dispositivo onde as mensagens de erro devem ser gravadas, normalmente em tela.

3.4.4 CONTROLE DE FLUXO

As expressões condicionais em Perl tem duas formas básicas de sintaxe, conforme a quantidade de instruções que validam. Quando validam apenas uma instrução os comandos são inseridos após o comando sem nenhum separador explícito. Esta construção pode ser empregada pelos seguintes comandos:

a) *If*: a declaração é executada apenas se a expressão lógica é verdadeira;
b) *unless*: a declaração é executada apenas se a expressão lógica é falsa;
c) *while*: a declaração é executada repetidamente até que a expressão lógica seja falsa;
d) *until*: a declaração é executada repetidamente até que a expressão lógica seja verdadeira.

No exemplo do quadro 23 pode-se observar um exemplo de utilização das estruturas condicionais em PERL.

Quadro 23 - Exemplo de expressões condicionais para uma instrução em Perl

```
(...)

#   Verifica se a variável “num”
#   é menor que 100
#   e envia uma mensagem de erro.
$num = 1;
print STDERR "Erro. Numero < 100. \n" if $num < 100;


#   Se não for possível abrir o arquivo “NomArq”
#   será enviada uma mensagem de alerta
print STDERR " Arquivo Inexistente. \n" unless open(ARQ,"NomArq.txt");


#   Decrementa a variavel “total” de acordo com decrement
#   até que seja menor que o próprio decrement.
$total = 100;
$decrement = 3;
$total -= $decrement while $total > $decrement;
print "$total decrementando em $decrement. \n" ;


#   Cria um array com três elementos e lista os
#   elementos do array até encontrar o elemento
#   com valor igual a “Terceiro”
@um_array = ("Primeiro" ,"Segundo", "Terceiro");
$Ind = 0;
print "$um_array[$Ind] $Ind \n" until $um_array[$Ind++] eq "Terceiro";
print "Ate o terceiro temos $Ind. \n" ;

(...)
```

A sintaxe altera quando deseja-se a execução de várias instruções. O comando é iniciado pelo identificador (*if, while* ou *until*), seguido pela expressão lógica entre parênteses e pelas instruções entre chaves. As expressões condicionais na linguagem Perl são semelhantes as expressões da linguagem *Java*, exceto os parênteses ao redor da expressão lógica e as chaves ao redor do bloco de declarações que são obrigatórios.

O *foreach* é um operador especial. Ele faz iterações sobre o conteúdo de um *array* e executa as declarações em um bloco de declarações para cada elemento do *array*.

No quadro 24 pode-se observar um exemplo semelhante ao do quadro 23, mas utilizando a sintaxe para múltiplas instruções.

Quadro 24 - Exemplo de expressões condicionais para múltiplas instruções em Perl

```
(...)
#   Verifica se a variável “num” é menor que 100
#   e envia uma mensagem de erro.
$num = 1;
if ($num < 100) {
  print STDERR "Erro. Numero < 100. \n"; }
else {
  print "Numero maior que 100";}

#   Decrementa a variável “total” de acordo com decrement
#   até que seja menor que o próprio decrement.
$total = 100;
$decrement = 3;
while ($total > $decrement) {
  $total -= $decrement }
print "$total decrementando em $decrement. \n" ;

#   Cria um array com três elementos e lista todos os elementos
@um_array = ("Primeiro", "Segundo", "Terceiro");
foreach $elemento_atu(@um_array) {
  print "$elemento_atu \n" };
(...)
```

A linguagem Perl implementa o conceito de subrotinas. Para implementar uma subrotina em Perl é necessário utilizar o comando Sub. Podem ser passados parâmetros para a subrotina de forma idêntica às funções e procedimentos implementados em diversas linguagens, mas não é necessário definir explicitamente quais ou quantos parâmetros podem ser passados para a subrotina. Todos os parâmetros estarão disponíveis numa variável especial de *array* denominada “@_”. O retorno de valor pela subrotina se dá pela simples definição de um valor sem nenhum comando de atribuição, como pode ser visto no quadro 25.

Quadro 25 - Exemplo de subrotinas em Perl

```
(...)
#definição da subrotina
sub maximum {
  if ($_[0] > $_[1]) {
    $_[0]; # retorna o valor do primeiro parâmetro
  } else {
    $_[1]; # retorna o valor do segundo parâmetro
  }
}

# Executa a função passando dois valores como parâmetro
$biggest = &maximum(37, 24);
# o valor de giggest é 37
(...)
```

3.5 ACTIVE SERVER PAGES

Active Server Pages (ASP) é uma linguagem de programação encapsulada em documentos HTML que incorporam conteúdo dinâmico. Estes arquivos em geral possuem a extensão “ASP” e são formados por uma combinação de *tags* HTML normais e *Server-Side Script*s que são códigos de programa executados pelo servidor Internet. Desta forma o servidor *Web* retorna apenas informação em formato HTML padrão, tornando as aplicações em ASP acessíveis por qualquer navegador existente no mercado. Uma aplicação feita em ASP pode ainda conter linhas *Client-Side Script*, que são códigos de programa semelhantes aos *Server-Side Script*, mas que são executados na estação cliente ([MAR1999]).

3.5.1 CARACTERÍSTICAS

As principais características em ASP são:

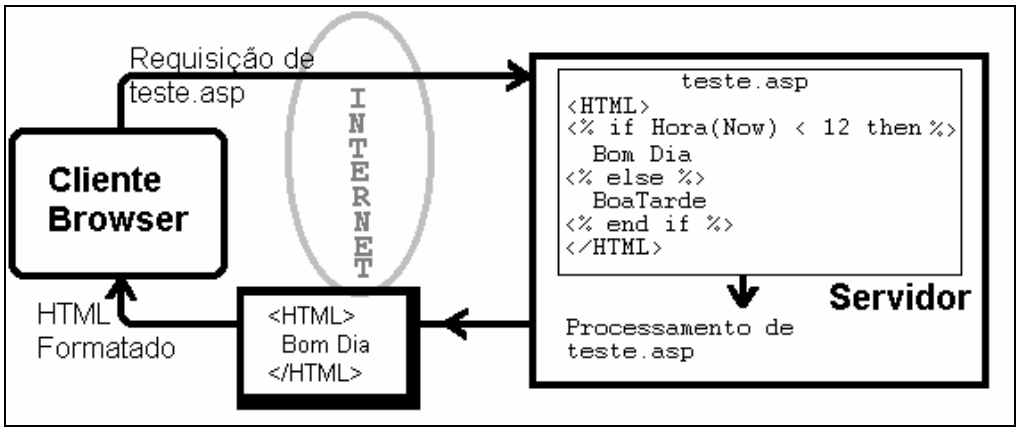
a) independência do navegador: os *Script*s rodam no servidor e somente os resultados são enviados ao usuário, sendo que qualquer navegador pode visualizar as páginas ASP;
b) segurança do código fonte: o cliente visualiza somente o resultado do processamento no formato HTML.

3.5.2 HOSPEDAGEM EM HTML

É possível identificar os arquivos que contém códigos ASP pela extensão “asp”. Ele contém basicamente, códigos HTML e *Script*s que irão rodar no servidor. A linguagem *Visual Basic Script* é a linguagem padrão utilizada pelo servidor Internet da empresa Microsoft, mas pode-se utilizar outras linguagens. Em síntese, a tecnologia ASP é muito semelhante a tecnologia CGI. A principal diferença é que na programação CGI tem-se um aplicativo que será executado e retornará informações de acordo com o protocolo HTTP, enquanto que uma aplicação ASP é um documento HTML que possui *tags* especiais interpretadas pelo servidor Internet, que também é responsável por enviar o documento no formato HTML padrão ao cliente. Na figura 3 pode-se observar o esquema geral de uma solicitação feita por um cliente para uma aplicação em ASP. Inicialmente o cliente faz a requisição dos dados, o servidor localiza o documento HTML e percorre todo o seu conteúdo a procura dos códigos que devem ser interpretados e substituídos pelo resultado do processamento. Após a finalização desta

análise o documento HTML padrão será enviado ao cliente como resposta ao pedido ([MAR1999]).

Figura 3 - Esquema geral das solicitações em ASP



Fonte: [UNI2000]

Os seguintes processos ocorrem na execução de uma aplicação em ASP:

a) o usuário faz a requisição de uma aplicação em ASP através do navegador;
b) a requisição é enviada ao servidor usando o protocolo HTTP;
c) o servidor verifica o tipo de arquivo. Encontrando um arquivo ASP (extensão .ASP) o servidor analisa o código *Script* contido no arquivo e executa-o;
d) o resultado do processamento é devolvido ao navegador no formato HTML padrão que interpreta o código HTML e mostra a página ao usuário.

3.5.3 SINTAXE

Os códigos de programa que devem ser interpretados pelo servidor são identificados pelas *tags* "<%" e "%>" que são uma variação da *tag* padrão da linguagem HTML. A sintaxe para os comandos ASP podem ser vistos no quadro 26.

Quadro 26 - Sintaxe dos comandos ASP

```
<html><body>
Página HTML
<%
response.write "Contendo conteúdo dinâmico <br>"
response.write "Endereço IP do cliente: " &
request.Servervariables("remote_Addr")&"<BR>"
response.write "Servidor: " & request.Servervariables("server_name")
%>
</body></html>
```

3.5.4 ORIENTAÇÃO A OBJETO

Para acessar as informações de solicitação dos navegadores o ambiente ASP implementa o objeto *request*. Para tratar as diversas informações que as solicitações disponibilizam o objeto *request* se utiliza de coleções ([MAR1999]). As coleções que podem ser acessadas através do objeto *request* são:

a) *Query String*: permitem acesso às informações adicionais do HTTP passadas através do método *get*;
b) *Form*: permitem acesso às informações adicionais do HTTP passadas através do método *Post*;
c) *ServerVariables* - retornam informações das varáveis de ambiente do servidor;
d) *ClientCertificates* - permitem acesso as informações de certificados quando do acesso a um site seguro como meio de identificação;
e) *Cookies* - permitem tratar os valores dos *cookies* enviados em uma requisição HTTP.

Para enviar informações ao cliente (navegador) o ambiente ASP implementa o objeto *response* que possui apenas a coleção *Cookies.* Através desta coleção é possível criar e manipular os valores dos *cookies* que são enviados ao cliente.

O objeto *application* tem como objetivo armazenar e compartilhar valores, estados e propriedades a nível de aplicação, ou seja, informações comuns a todos os usuários que solicitem uma mesma página HTML. Este objeto é inicializado quando for feita a primeira solicitação a página e somente finaliza quando o servidor também for finalizado.

O objeto *session* tem como objetivo armazenar e compartilhar valores, estados e propriedades a nível de seção, ou seja, informações individuais a cada usuário que solicita uma determinada página HTML. Este objeto é inicializado a cada solicitação a página e finaliza quando a conexão com o cliente for finalizada.

3.6 COMANDOS SERVER SIDE INCLUDE

Segundo [HER1997], os comandos *Server Side Include* (SSI) foram criados para satisfazer o desejo de incluir um arquivo comum no interior de um grupo de arquivos diferentes. A utilização mais comum para os SSI ocorre na inclusão de assinaturas ou

logotipos de empresas em todos os documentos criados. O arquivo a incluir fica residente no servidor, sendo incluído sempre que um arquivo da HTML que contenha um comando *include* for solicitado, sendo essa a origem do termo *Server Side Include.*

3.6.1 SINTAXE

Os comandos SSI são uma extensão do comando de comentário da HTML. Desta forma, se o documento contendo o comando for interpretado por um servidor que não suporte os comandos SSI eles serão simplesmente ignorados. A sintaxe para os comandos SSI podem ser vistos no quadro 27.

Quadro 27 - Sintaxe dos comandos SSI

```
<html>
<body>

Data de modificação do documento:
<!-- #flastmod file("teste.shtml") --> <br>

Nome do Documento: <!-- #echo var="DOCUMENT_Uri"--><br>

<!-- #config timefmt= "%H:%M - %A - %d  de %B de %Y"  -->
Data Hora Atual: <!-- #echo var="DATE_Local"--><br>

</body>
</html>
```

Para retornar corretamente um documento que contenha comandos SSI o servidor necessita ler cada linha do arquivo HTML e pesquisar por comandos a serem executados (*parsing)*, esta análise coloca uma carga extra sobre o servidor. Isto significa que documentos que contenham comandos SSI irão demorar mais tempo para retornar a estação cliente se comparados com documentos comuns da HTML.

Os comandos SSI podem ser considerados como uma porta de entrada para o servidor, principalmente quando da utilização do comando *exec* (executar) que permite a execução de comandos do sistema operacional, por este motivo muitos servidor *Web* implementam um tratamento especial para a execução dos comandos SSI, possibilitando desabilitar este ou todos os comando SSI.

3.6.2 PRINCIPAIS COMANDOS SSI

Os principais comandos SSI que podem ser implementados em documentos HTML são:

a) #flastmod - retorna a data de última modificação do arquivo especificado no argumento do comando;
b) #config: modifica a saída gerada por outros comandos SSI como mensagens de erro, formatos de datas e hora, tamanho do arquivo, entre outros;
c) #include: permite a inclusão do conteúdo de outros arquivos no documento atual;
d) #echo: pode exibir as informações: data e hora atuais, nome do documento atual e a data e hora de modificação do documento atual;
e) #exec - Permite a execução de comandos do sistema operacional e programas no servidor da *Web*.

4 PROTÓTIPO

Para a demonstração prática do trabalho foi implementada uma ferramenta de auxílio para o processo de compra utilizando a infra-estrutura da internet, também conhecido como *eBusiness*, para uma loja de produtos artesanais.

Um dos objetivos desta implementação é fazer uma analogia das técnicas empregadas destacando suas vantagens e desvantagens.

4.1 LEVANTAMENTO DE INFORMAÇÕES E ESPECIFICAÇÃO

Observou-se o interesse da loja em informatizar seus processo de venda de produtos. As vendas podem ser efetuadas no balcão ou através dos pedidos de compra que são cadastrados pelos vendedores e enviados a loja. Os pedidos cadastrados serão previamente analisados quanto a disponibilidade do produto, quantidade mínima, prazo de entrega, entre outros. A confirmação é feita por telefone ou e-mail ao vendedor. Muitas informações a respeito dos clientes geralmente estão desatualizadas ou distribuídas entre os vendedores.

A entrega é feita através do serviços dos correios ou transportadoras e o pagamento por depósito bancário, cartão de crédito ou diretamente na loja. Para demonstração dos produtos a loja mantém um catálogo atualizado dos produtos e preços.

Alguns problemas observados são a dificuldade em obter a informação sobre os pedidos que estão sendo processados, cumprir a data de entrega, a dificuldade dos clientes em localizar a loja e a demora em receber os pedidos dos vendedores.

O protótipo desenvolvido propõe a implementação de uma ferramenta para o auxílio das vendas e o controle das informações de clientes, produtos, pedidos de compra, catálogo de produtos para compra e relatórios. Serão utilizadas as várias técnicas estudadas com o objetivo de fazer uma análise comparativa entre as mesmas.

4.1.1 AMBIENTE

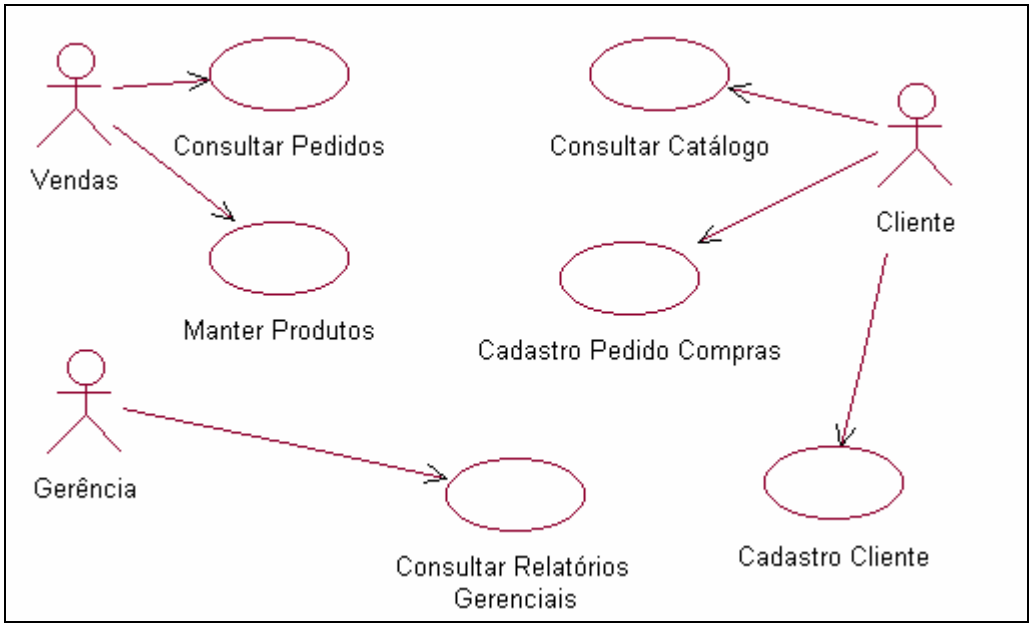
São utilizados diversos ambientes para o desenvolvimento das aplicações, dentre os quais:

a) editores de texto para o desenvolvimento do código fonte nas linguagens de *script* que serão interpretadas pelo servidor internet ou pelo navegador;
b) na execução dos aplicativos em ASP é utilizado o servidor Internet *Personal Web Server* da empresa Microsoft;
c) na execução dos *Scripts* CGI em Perl é utilizado o servidor *Web* Apache 1.3.11, com o interpretador Perl 5.005;
d) navegador com suporte a *Java Script*, *Visual Basic Script* e *Java Applets*;
e) na execução das *Applets* em *Java* é utilizado o compilador do pacote *Java* Developers Kit versão 1.0.2, da empresa Sun Microsystems;
f) banco de dados *Microsoft Access*.

4.1.2 DIAGRAMAS DE CASOS E USOS

Com a utilização da ferramenta de modelagem Rational Rose, foi desenvolvido o diagrama de casos de uso conforme a figura 4, onde são modelados os principais casos de uso do sistema.

Figura 4 - Diagrama de casos de uso



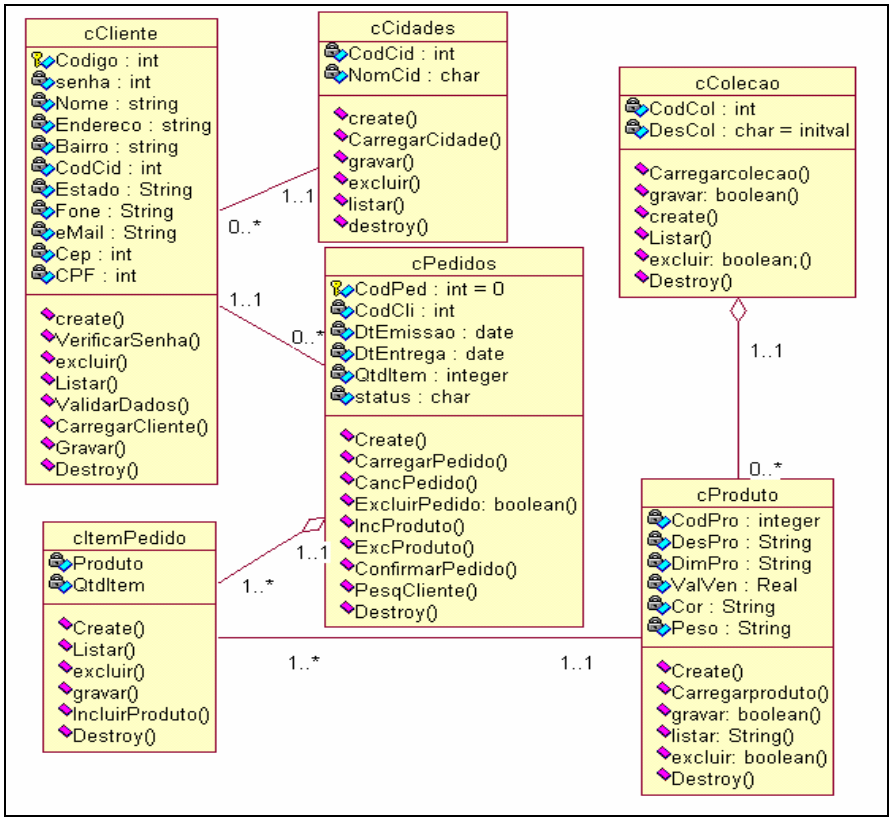
Os casos de uso definidos para o sistema são:

a) consultar pedidos: possibilita consultar as informações dos pedidos cadastrados;
b) manter produtos: inclusão, alteração e exclusão das informações de produtos;
c) consultar relatórios gerenciais: relatórios com as informações necessárias para o melhor gerenciamento do negócio;
d) cadastro de pedido de compra: inclusão de um pedido de compras com a relação dos produtos desejados;
e) cadastro cliente: inclusão e alteração das informações do cliente;
f) consultar catálogo: apresentação dos produtos para o cliente com o objetivo de efetuar a compra.

4.1.3 DIAGRAMAS DE CLASSES

A figura 5 está representado o diagrama de classes na fase de análise com as principais classes do sistema, seus atributos e relacionamentos.

Figura 5 - Diagrama de classes



As principais classes do sistema são:

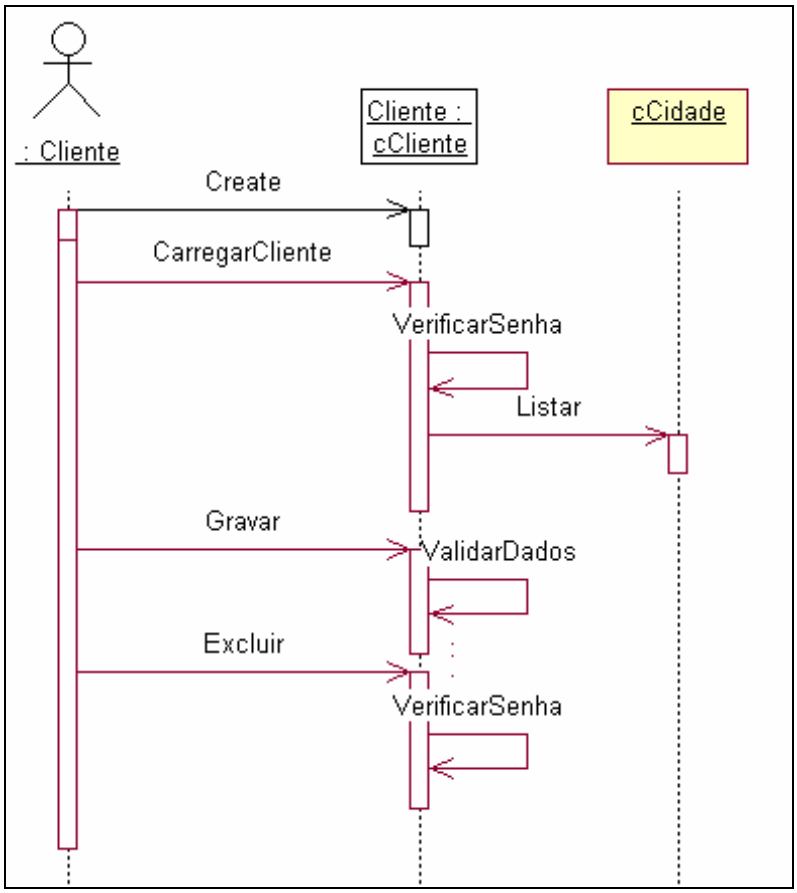
a) cClientes: classe que modela o cliente. Permite a inclusão, alteração e exclusão das informações dos clientes além dos métodos para validar a senha, listar as informações dos clientes e validar os dados informados para gravação;
b) cPedidos: classe que modela o pedido de compras efetuado pelo cliente. Permite a inclusão e cancelamento (exclusão) das informações do pedido além dos métodos para incluir e excluir produtos ao pedido;
c) cItemPedido: classe que modela os produtos incluídos no pedido de compras. Permite a inclusão, exclusão e listagem das informações dos itens do pedido;
d) cProduto: classe que modelo os produtos. Permite a inclusão, alteração e exclusão das informações dos produtos;
e) cCidades: classe que permite acessar as cidades que podem ser utilizadas no cadastro do cliente;
f) cColecao: classe que modela as coleções. As coleções são agrupamento de produtos do mesmo tipo.

4.1.4 DIAGRAMAS DE SEQUÊNCIA

Nos diagramas de seqüência pode-se observar o tempo de execução de cada ação e verificar as mensagens trocadas entre os objetos. As principais seqüências encontradas no sistema são:

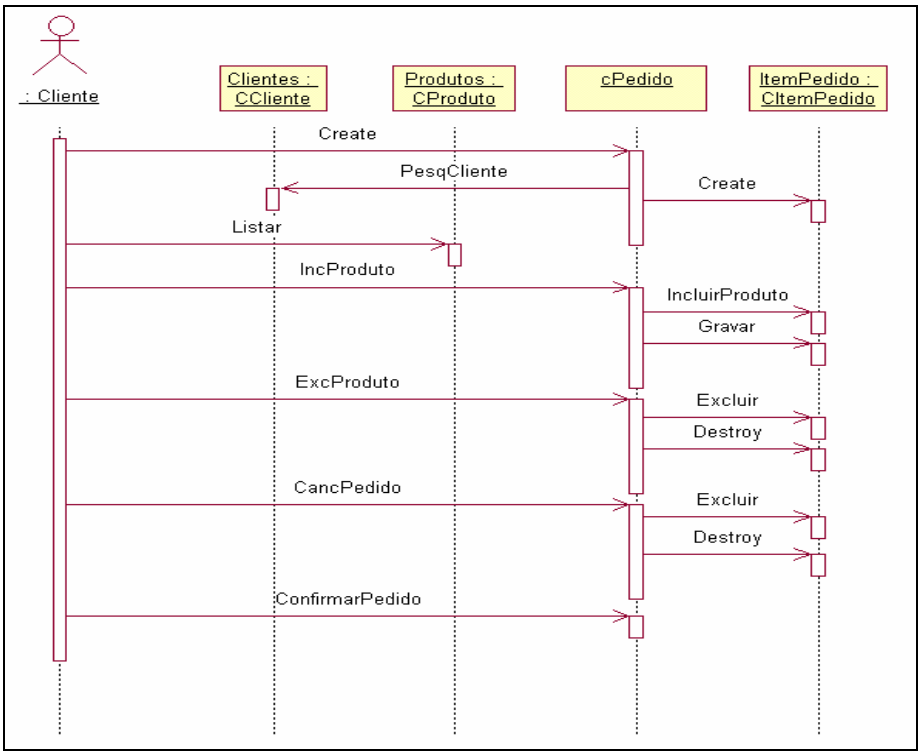
a) manter cliente (figura 6): para a utilização dos serviços de compra na página é necessário o cliente estar cadastrado. Caso o cliente não tenha se cadastrado, um novo formulário deverá ser preenchido com suas informações. Caso o cliente já esteja cadastrado o método CarregarCliente e VerificarSenha serão disparados para apresentarem suas informações. O método Listar da interface cCidade é utilizado para retornar uma listagem das cidades que o cliente poderá escolher. Para o cliente gravar seus dados deverão ser validadas as informações digitadas. O cliente também tem a opção de excluir suas informações desde que seja confirmada sua senha;

Figura 6 - Manutenção dos clientes



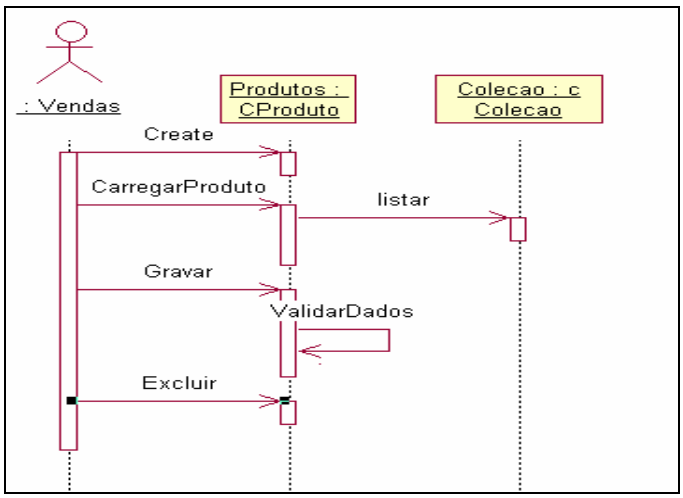
b) cadastro de pedido de compra (figura 7): sempre que acessada esta página, o pedido atual é exibido ou um novo pedido é criado automaticamente através do método *create* da interface cPedido. As informações sobre o cliente são retornadas através da interface Clientes. Para incluir um novo item de produto no pedido, uma listagem dos produtos é apresentada através do método listar da interface pedidos. Quando incluído um novo produto o método Incproduto é invocado na interface Pedidos. Esta por sua vez, acessa a interface ItemPedido para incluir e gravar o produto e a quantidade pedida. O método ExcProduto exclui o produto do pedido de compras e o método CancPedido exclui todos os itens de produto e cancela o pedido. Quando o pedido está corretamente cadastrado, o método ConfirmaPedido é acessado para confirmar as informações digitadas e enviar um e-mail de aviso ao responsável pelas vendas.

Figura 7 - Cadastro de Pedido de Compra



c) manter produtos (figura 8): A manutenção dos produtos estará disponível somente pelo responsável pelas vendas. Caso seja informado o código do produto, o método carregar produto será acessado para exibir as informações atuais do produto para alteração senão um novo produto é cadastrado. Quando solicitado o método gravar, o método ValidarDados será executado para verificar os dados informados. O método excluir também poderá ser acessado para excluir o produto.

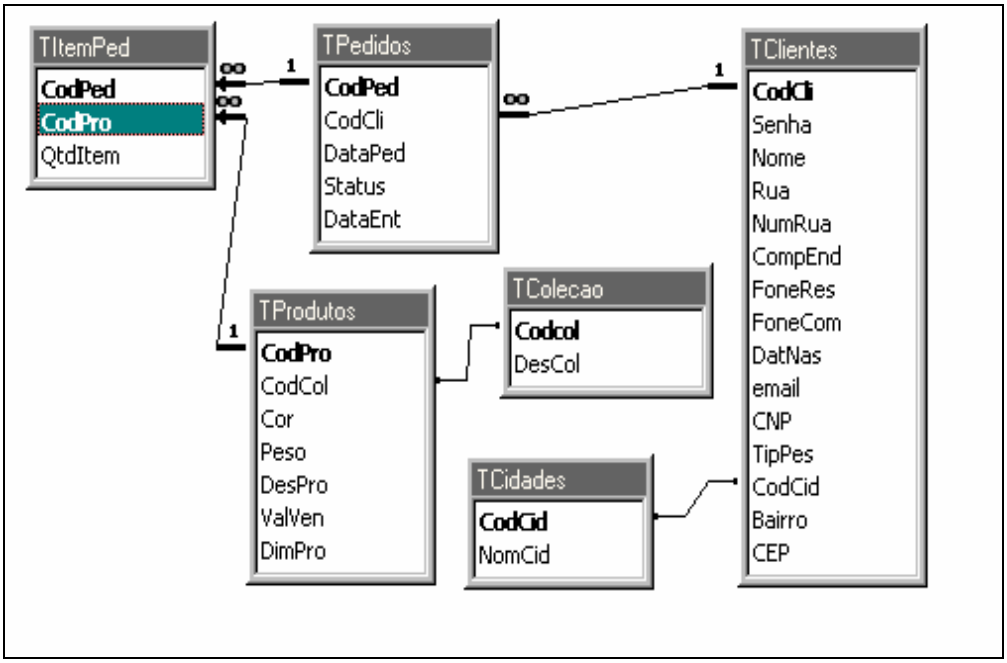
Figura 8 - Manter produtos



4.2 IMPLEMENTAÇÃO

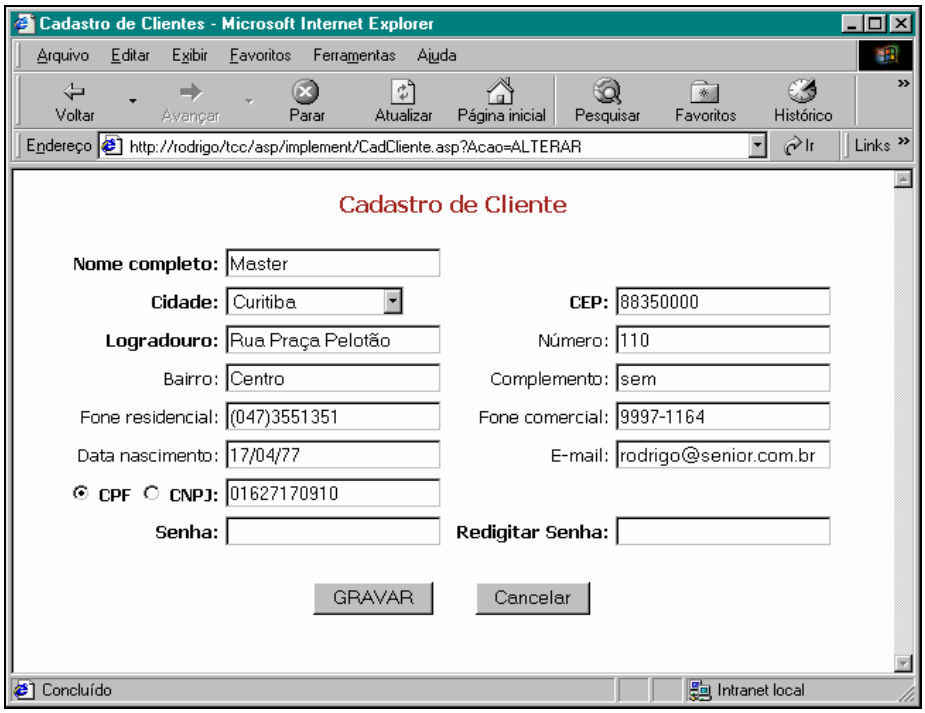
A seguir são apresentados alguns detalhes da implementação. Na figura 9 pode ser observado o dicionário de dados utilizando o Banco de Dados *Microsoft Access* utilizado na aplicação.

Figura 9 - Modelagem de Dados



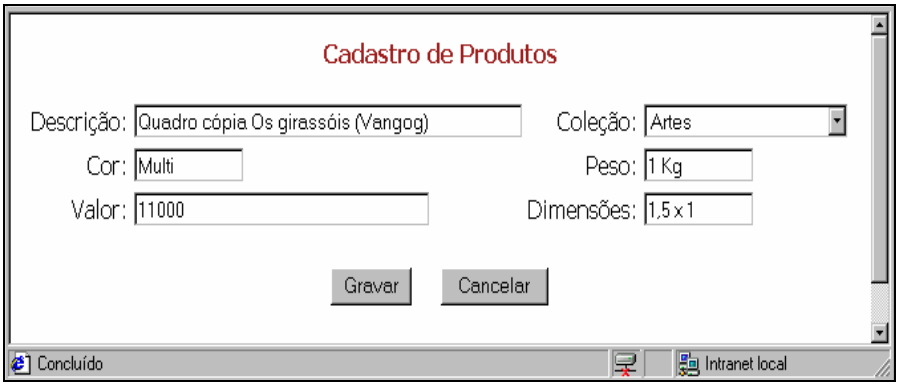
A figura 10 mostra a tela de cadastro dos clientes. É utilizado o navegador *Internet Explorer* versão 4 para interface com o usuário, pois as telas apresentadas são documentos utilizando formulários de entrada de dados da HTML. Todos os dados obrigatórios estão em negrito. Caso o cliente já esteja cadastrado suas informações serão carregadas possibilitando a alteração dos dados. A senha sempre deve ser reinformada. Quando é selecionada a opção gravar, os dados serão consistidos. Caso os dados não estejam corretamente cadastrados o formulário é novamente carregado com todas as informações previamente digitadas e uma mensagem de erro.

Figura 10 - Tela Cadastro de Clientes



Na figura 11 é apresentada a tela de cadastro dos produtos com o funcionamento semelhante a tela de cadastro de clientes.

Figura 11 - Tela Cadastro de Produtos



Na figura 12 é apresentada a tela de Pedidos de Compra. Esta tela é composta por três *frames* (Molduras). No *frame* de cima são exibidas as informações do cliente e do pedido de compra, no *frame* da direita são relacionadas as ações disponíveis na página e no *frame* de baixo são relacionados os itens de produtos. As seguintes ações estão disponíveis:

a) Atualizar: Recarrega as informações da página atualizando os dados;

b) Produtos: um *link* para o tela de catálogo dos produtos;

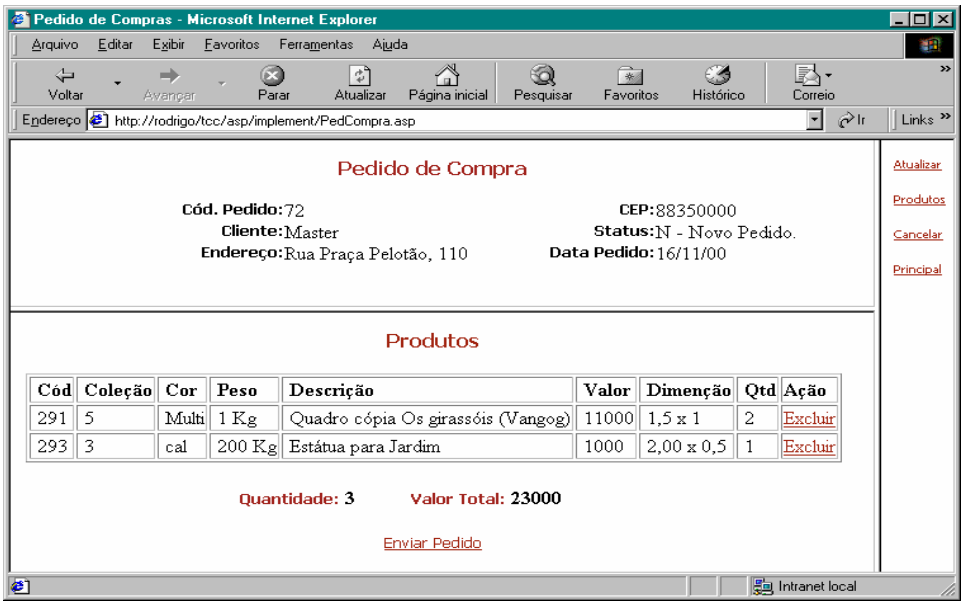
c) Cancelar: Exclui todos os itens de Produtos e o Pedido da base de dados;

d) Principal: link para o menu principal da aplicação;

e) Excluir: Exclui o item de produto selecionado da base de dados;

f) Enviar Pedido: Confirma todas as informações do pedido.

Figura 12 - Tela Pedido de Compras



4.2.1 IMPLEMENTAÇÃO COM LINGUAGENS DE *SCRIPT*

Devido às restrições de segurança impostas pela navegador e o fato de as mesmas serem executadas no lado do cliente dificultando o acesso ao banco de dados no servidor Internet, a implementação com linguagens de *script* se limitou a efetuar apenas pequenos controles como responder aos eventos da página, validar os dados antes de serem enviados ao servidor, apresentar mensagens e textos de ajuda ao usuário, entre outros. A aplicação em ASP foi utilizada como base para a implementação destes tratamentos utilizando a linguagem *Java Script* de modo a formar uma implementação mais completa e eficiente. No quadro 28 pode ser vista a página HTML contendo os fontes do script.

Quadro 28 – Consistências no Cadastro de Clientes implementas em *JavaScript*

```
<html><body>
<script language="JavaScript" >
// ---------- Limita o Tamanho máximo campo ----------
function tammax(campopar,tammaxpar){
  if (campopar.value.length > tammaxpar) {
    campopar.value = campopar.value.substr( 0, tammaxpar);
    statusmens("Este é o tamanho máximo permitido!");
  }
}
function statusmens(msgpar){
  window.status=msgpar;
  idTimer=window.setTimeout("window.status='';",1500);  }
// ---------- Valida a informação como uma data (DD/MM/YYYY) ----------
function validadata(campopar) {
  texto = campopar.value;
  dia = texto.substr(0, texto.indexOf("/"));
  texto = texto.substr(texto.indexOf("/")+1,7);
  mes = texto.substr(0, texto.indexOf("/"));
  ano = texto.substr(texto.indexOf("/")+1,4);
  datanova = new Date(ano, mes, dia);
  if (datanova == 'NaN') {
       alert("Data Invalida");
       campopar.value = "";
  } else {
       campopar.value =
datanova.getDate()+"/"+datanova.getMonth()+"/"+datanova.getYear();
  }
}
// ---------- Valida a informação como um e-mail (XXX@XXX) ----------
function validamail(campopar) {
  texto = campopar.value;
  if (texto.indexOf("@") <= 0) {
       alert("E-mail inválido");
       campopar.value = "";
  }
}
</script>
Nome:<input type='text' name='Nome' onkeyup='tammax(this,40)'><br>
Nacto:<input type='text' name='DatNas' onkeyup='tammax(this,10)'
onblur='validadata(this)'><br>
Mail:<input type='text' name='email' onkeyup='tammax(this,40)'
onblur='validamail(this)'><br>
</body> </html>
```

4.2.1.1 VANTAGENS

A principal utilização das linguagens de *script* está no tratamento dos eventos da página HTML. Através destas linguagens é possível executar rotinas na entrada ou saída da ativação da página HTML, validar os dados no momento da digitação pelo cliente em um campo texto, executar tarefas quando pressionados botões, entre outros. Esta capacidade de manipular os objetos e eventos da página HTML não é possível de ser executada pelas outras técnicas de programação estudadas.

4.2.1.2 DESVANTAGENS

Além das restrições de acesso ao banco de dados no servidor, um documento HTML contendo determinados comandos de uma linguagem de *script* pode funcionar somente em alguns navegadores ou somente em determinadas versões dos navegadores. Além disto, o código fonte do *script* sempre será carregado para estação cliente, não sendo possível proteger ou esconder o mesmo.

4.2.2 IMPLEMENTAÇÃO COM APPLET JAVA

Para o desenvolvimento com *Applets* foi utilizado o compilador do pacote *Java Developers Kit* (JDK) versão 1.0.2, da empresa Sun Microsystems. Para executar as *applets* foi utilizado o navegador *Internet Explorer* versão 5. Para o acesso aos dados foi utilizado o driver *Java Data Base Connection* (JDBC) que acessa um banco de dados local em Access, como pode ser visto no quadro 29.

Quadro 29 - Rotina de acesso ao banco de dados em *Applet*

```
Public class ConBanco
{
  public Connection con;
  public Statement stmt;
// ---Conectar no banco ---
  public void inibd()
  {
    String url= "jdbc:odbc:tcc";
    try
    {
      System.out.print("Iniciando a conexão com o BD...");
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      con = DriverManager.getConnection(url,"","");
      stmt = ConBanco.con.createStatement();
    } catch(SQLException sql) {
      System.out.println("Houve uma SQLException: "+sql);
    } catch(ClassNotFoundException fnf) {
      System.out.println("Houve uma ClassNotFoundException: "+fnf);
    }
  }
  // ---Desconectar do banco ---
  public void fimbd()
  {
      try {
      System.out.print("fechando a conexão com o BD...");
      Stmt.close();
      Com.close();
      System.out.println("pronto!");
      } catch(SQLException sql) {
        System.out.println("Houve uma SQLException: "+sql);
      }
  }
}
```

Para a entrada de dados no sistema é utilizada a interface implementada pela própria *Applet* uma vez que a mesma é executada no navegador do cliente e possui os componentes gráficos necessários. No quadro 30 pode ser observada a rotina para inclusão dos dados do cliente no banco de dados.

Quadro 30 - Rotina de gravação dos dados em *Applet*

```
// ------------- Alterar dados cliente  ----------------------
  public void gravar(int codclipar)  {
    // cria um novo objeto para
    // conexão com o banco de dados
    if (this.validardados()) {
      ConBanco conexao;
      conexao = new ConBanco();


      // inicia banco dados
      status = "iniciando conexao com o banco de dados...";
      conexao.inibd();
      String query = "Update TClientes set "
                    +"nome = '"
                    +nome+"', "
                    +"rua = '"
                    +rua+"', "
                    +"numrua = '"
                    +numrua+"', "
                    +"compend = '"
                    +compend+"', "
                    +"foneres = '"
                    +foneres+"', "
                    +"fonecom = '"
                    +fonecom+"', "
                    +"email = '"
                    +email+"', "
                    +"cnp = '"
                    +cnp+"', "
                    +"codcid = "
                    +codcid+", "
                    +"bairro = '"
                    +bairro+"', "
                    +"cep = '"
                    +cep+"', "
                    +"senha = '"
                    +senha+"' "
                    +" where codcli = "+String.valueOf(codclipar);

      try {
        codcli = codclipar;
        // Executa a query, criando um objeto ResultSet
        ResultSet rs = conexao.stmt.executeQuery(query);
      }catch (Exception erro) { }
      // fecha banco dados
      conexao.fimbd();
    } else { status = "dados nao foram corretamente validados..."; }
  }
```

4.2.2.1 VANTAGENS

As aplicações em *Applet* apresentam uma facilidade de implementação pois são totalmente carregadas e executadas no navegador de forma semelhante as aplicações tradicionais. A aplicação em *Applet* não necessita da utilização dos formulários ou objetos da HTML para a entrada de dados pois é possível implementar suas próprias telas e componentes gráficos para interface com o usuário.

4.2.2.2 DESVANTAGENS

A principal desvantagem na utilização das *Applets* está no tempo necessário para carregamento de todo o código binário para o navegador do cliente. O fato da aplicação ser executada na estação cliente também pode gerar dificuldades para o acesso a informações ou recursos do servidor *Internet*.

4.2.3 IMPLEMENTAÇÃO EM CGI UTILIZANDO PERL

Para o desenvolvimento do aplicativo em CGI foi utilizada a linguagem PERL com o interpretador Perl 5.005. Os dados são acessados através da interface ODBC em banco de dados Access. A implementação com PERL é semelhante a implementação em ASP, utilizando os formulários da HTML para a entrada de dados. A principal diferença é que o código em Perl é executado pelo interpretador Perl instalado e não pelo servidor *internet*. No quadro 31 pode ser visto uma parte do código fonte utilizado para gravar as informações do cliente.

Quadro 31 - Gravação dos dados do cliente em PERL

```
use Win32::ODBC;
require("rrlib.pl");
&ReadParse(*field);
# ---- Passa os valores para as variáveis ----
$codcli = &retglobal("codcli");
$confsenha = $field{'confsenha'} ;
(..)
# 'Verifica se necessita senha
if (($acao eq "ALTERAR") || ($acao eq "EXCLUIR")) {
  #Se não tem usuário ativo
  if (($codcli eq 0) || ($codcli eq "")) {
    print "Location: IdentCli.pl?URLDest=CadCli.pl?acao=$acao \n\n";
    goto FIM;
  }
}
# imrimi o inicio da página de resposta
print "Content-type: text/html\n\n";
print <<FPRINT;
<html>
<body>
Cadastro de Cliente</p>
FPRINT

If ($acao eq "GRAVAR") {
  if (&validadados) {
    if (($codcli ne 0) && ($codcli ne "")) {
      # altera dados
      $db0 = new Win32::ODBC( "tcc" ) || die "Erro criando objeto\n";
      $db0->SetMaxBufSize(2048);

      if ( $db0->Sql("update TProdutos set nome = '$nome' "
      ."and cidade = '$cidade' "
      (..)
      ."and senha = '$senha' where codcli = $codcli ")) {
        print "ERRO: Código do Cliente não encontrado";
      } else {
         print " Cadastro atualizado com sucesso ";
      }
    }
    $db0->Close();
    goto FIM;
  } else {
    print "Os Dados não foram corretamente preenchidos!!!";
  }
}
FIM:
```

4.2.3.1 VANTAGENS

A principal vantagem na utilização em Perl, está na capacidade em manipular grandes quantidades de texto, pois possui muitas funções para comparação de padrões e equivalências entre *strings*. Por ser uma linguagem interpretada também oferece facilidade de manutenção dispensando o processo de compilação.

4.2.3.2 DESVANTAGENS

Existe uma certa dificuldade na implementação e manutenção do código HTML em PERL, pois é necessário implementar todos as *tags* da HTML através dos comandos de saída da linguagem PERL sem o auxilio de um editor específico de HTML. A sintaxe da linguagem PERL inicialmente apresenta uma maior complexidade e exige uma maior atenção durante o desenvolvimento em relação as outras linguagens estudadas.

4.2.4 IMPLEMENTAÇÃO EM ASP

Para a execução do aplicativo em ASP foi utilizado o servidor *Personal Internet Server*. O acesso ao banco de dados Access é possível através da implementação de um objeto *Activex Data Object* (ADO) derivado da classe *adodb.connection* disponibilizado no ambiente ASP. Para manipulação dos registros de dados foi utilizado um objeto derivado da classe *adodb.Recordset*. O código utilizado para conexão com o banco de dados pode ser visto no quadro 32.

Quadro 32 - Conexão com a base de dados em ASP

```
Function AbreBaseDados
  'Cria um objeto de conexão com a base de dados
  set Conntemp = server.createObject("adodb.connection")
  Conntemp.ConnectionTimeOut = 15
  Conntemp.Open "DBQ=C:\Inetpub\bd\BDTCC.mdb;DRIVER={Microsoft Access
Driver (*.mdb)}"

  'Cria um objeto de manipulação de registros da base de dados
  set RS = server.createObject("adodb.Recordset")
end Function
```

Para a entrada de dados no sistema é utilizado o formulário disponível pela HTML, estes dados são enviados para as rotinas em ASP através dos métodos *get* e *post* e ficam disponíveis para a aplicação através do objeto *request*. No quadro 33 pode ser observada a rotina para inclusão dos dados do cliente no banco de dados.

Quadro 33 - Gravação dos dados do cliente em ASP

```
If UCase(request.form("Acao")) = "GRAVAR" then
  IF ValidaDados Then
    call AbreBaseDados

    if session("CodCli")<>0 and session("CodCli")<>"" then
      RS.open "Select * from TClientes where CodCli = "&session("CodCli"),
Conntemp, adOpenKeySet, adLockOptimistic
      if RS.eof then
        response.write "ERRO: Código do Cliente não encontrado"
        response.end
      end if
    '--- Se cliente não identificado inclui novo registro
    else
      RS.open "Select * from TClientes", Conntemp, adOpenKeySet,
adLockOptimistic
      'adiciona um novo registro ao Banco Dados
      RS.AddNew
    End if

    RS("Nome")    = request.form("Nome")
    RS("Rua")     = request.form("Rua")
    RS("NumRua")  = request.form("NumRua")
    RS("CompEnd") = request.form("CompEnd")
    RS("FoneRes") = request.form("FoneRes")
    RS("FoneCom") = request.form("FoneCom")
    auxdata       = CDate(request.form("DatNas"))
    if err.number = 0 then
      RS("DatNas")= auxdata
    end if
    err.clear
    ' (..)
    RS("Senha")   = UCase(request.form("Senha"))
    RS.UpDate

    if err.number <> 0 then
      response.write "ERRO: Não foi possível cadastrar cliente.<br>"
      response.write "Descrição do erro: "&err.description&"<br>"
      response.write "Erro Nativo: "&err.nativeerror&"<br>"
    else
      'confirmação dos dados
      response.write "Cadastro efetuado com sucesso!<br>"
      session("CodCli")   = RS("CodCli")
    end if
    call FechaBaseDados
  end if
end if
```

4.2.4.1 VANTAGENS

A principal vantagem observada na utilização um ASP é a facilidade de desenvolvimento e manutenção do código HTML. O código ASP fica delimitado entre *tags* o que possibilita alterar a página através de um editor HTML comum, sem se preocupar com o código encapsulado. Outra vantagem observada é a possibilidade de utilizar variáveis a nível de seção para controlar as informações sobre cada cliente. As informações gravadas nestas

variáveis são mantidas para cada usuário, enquanto a seção estiver ativa, ou seja, enquanto o usuário estiver interagindo com as páginas HTML.

4.2.4.2 DESVANTAGENS

Uma desvantagem observada está na fase de depuração e testes da aplicação que é executada no servidor *internet.* Os servidores *internet* não possuem todos os recursos de depuração passo a passo ou visualização de variáveis como a maioria dos aplicativos compiladores ou interpretadores, dificultando a correção de erros. Para a validação dos dados informados pelo cliente é sempre necessário fazer uma solicitação ao servidor e carregar a página inteira novamente, pois não é possível responder a eventos da página ou fazer consistência durante a digitação do cliente.

4.2.5 IMPLEMENTAÇÃO COM COMANDOS SSI

Pelo fato de não possuir os recursos de controle de fluxo, acesso a banco de dados ou acesso a objetos distribuidos, não foi possível a implementação do programa utilizando apenas os comandos SSI.

4.2.6 RESULTADOS

Para o auxílio na tomada de decisão sobre quais as vantagens de cada técnica, é apresentada uma relação com algumas premissas sobre o desenvolvimento de aplicações dinâmicas na *Internet*:

a) proteção: quando deseja-se a proteção do acesso ao código fonte as linguagens de *script* hospedadas em documentos HTML não são indicadas, pois disponibilizam seu acesso juntamente com o código da página HTML no *navegador*;
b) desempenho: quando utilizadas as linguagens de *script* tem-se a vantagem da carga do programa ser em geral mais rápida em comparação com o código binário da *Applet*, mas durante a execução, um programa em *Applet* passa a ter melhor performance, principalmente devido a característica de ser compilada e não interpretada. Quando executam-se tecnologias como CGI ou ASP o processamento depende do servidor e da velocidade da rede, a performance ainda poderá ser comprometida no caso de ocorrerem vários acessos simultâneos ao servidor;

c) banco dados: para o acesso a banco de dados a técnica mais indicada é ASP ou alguma linguagem utilizando a interface CGI, pois por serem executadas no servidor tem facilidade em acessar os recursos desta máquina. As linguagens *Java Script* e *Visual Basic Script* tem sérias restrições de segurança para o acesso de qualquer recurso na estação cliente;

d) portabilidade: as aplicações em *Java Script*, *Visual Basic Script* e *Applet* são denominadas *Client-Side*, ou seja, são executadas no cliente. O programa navegador deve estar habilitado a executar as mesmas e o desenvolvedor deve ter uma preocupação sobre qual navegador que o cliente poderá estar utilizando. Para a aplicação CGI utilizando PERL e ASP apenas o servidor *Internet* deve estar capacitado a executar as solicitações e qualquer programa navegador poderá ser utilizado;

e) é possível a utilização de linguagens de *script*, *Java Applet* e ASP em um mesmo documento HTML simultaneamente, mas não é possível utilizar CGI e ASP para uma mesma requisição pois o servidor deve identificar uma única forma de processar e retornar o documento HTML.

Na tabela 8 podem ser observadas comparações das características nas diversas técnicas estudadas.

Tabela 8 - Comparações das técnicas em relação a suas características

Caracte-rística	***VB Script***	***Java Script***	***Applets Java***	**CGI utilizando Perl**	**ASP**
Suporte a objetos	Baseada em objetos	Baseada em objetos	Orientada a Objetos	Orientada a objetos	Utiliza os objetos disponíveis no ambiente e suporte a COM
Trata-mento de erros	Não Possui	Implementado em apenas alguns navegadores.	Possui	Possui	Tratamento simples que permite identificar o erro e continuar a execução do programa.
Código Executá-vel	Interpretada	Interpretada	Compilada em *byte Code*	Interpretada	Interpretada

Portabili-dade	Portável para todos os navegadores que implementam o interpretador *VB Script*	Portável para todos os navegadores que implementam o interpretador *Java Script*	Portável para todas as navegadores que implementam a máquina virtual *Java*	Portável para todas arquiteturas que tenham o interpretador PERL	Portável para todos os servidores *internet* que implementam o interpretador ASP.
Acesso Banco Dados	Não contém recursos para acesso a banco de dados	Não contém recursos para acesso a banco de dados	Permite acesso através da JDBC	Contém bibliotecas para acesso a banco de dados	Permite o acesso a objetos COM para acesso a banco de dados
Tipificação	Fracamente tipificada. Não é obrigatório declarar variáveis explicitamente	Fracamente tipificada. Não é obrigatório declarar variáveis explicitamente	Fortemente tipificada. declarações de variáveis explicitas	o primeiro caracter identifica o tipo de variável sem declaração explicita	Fracamente tipificada
Execução	Podem ser executadas pelo navegador ou servidor.	Podem ser executadas pelo navegador ou servidor.	*Applets* são Executadas pelo navegador	Executado pelo interpretador Perl no servidor.	Executada pelo servidor.
Proprieda-de	Microsoft Corporation	Implementada por diversos programas navegadores	Sun Microsystems	Distribuída segundo os termos do *General Public License* (GNU).	Microsoft Corporation

5 CONCLUSÕES

Através do estudo realizado e da experiência adquirida com o desenvolvimento da aplicação pode se concluir que a tecnologia ASP se apresentou como a mais adequada para a aplicação especificada neste trabalho. A facilidade da edição do código fonte do HTML através da utilização de um programa editor de HTML, a facilidade e rapidez no aprendizado da sintaxe da linguagem, a possiblidade da utilização de objetos distribuidos e a interpretação nativa do código fonte pelo servidor Personal Web Server são os fatores que determinaram a escolha desta técnica. Foi observado também a necessidade da utilização em conjunto das linguagens de script a fim de tratar os eventos da página e validar os dados informados sem a necessidade de efetuar solicitações ao servidor e tornando a aplicação mais consistente.

5.1 EXTENSÕES

Este trabalho mostrou que a área de programação voltada a *Internet* é muito ampla, possibilitando a implementação de várias técnicas no protótipo desenvolvido. A título de sugestão, pode-se citar o aprofundamento do estudo em cada uma das técnicas abordadas neste trabalho ou ampliar o estudo para outras tecnologias, tais como:

a) *Dynamic Hyper Text Markup Language* (DHTML): extensão da Linguagem HTML que possibilita criar documentos HTML dinâmicos com mais facilidade, utilizando os recursos da HTML;
b) *Cold Fusion Markup Language* (CFML): linguagem baseada em *tags*, de forma semelhante ao HTML, mas que permite integração entre elementos para acesso a banco de dados, aplicações de e-mail, entre outros;
c) PHP: linguagem de programação interpretada de livre distribuição (GNU), pré-processada pelo servidor *Web* e disponível para diversas plataformas incluindo Linux, Windows NT, Unix, entre outras.

5.2 LIMITAÇÕES

Neste trabalho foi apresentado um estudo de diversas tecnologias o que não permitiu um maior aprofundamento em uma tecnologia específica. Cada técnica poderia ser estudada em um trabalho distinto.

REFERÊNCIAS BIBLIOGRÁFICAS

[ASH2000] ASHTON, Elaine. ***Perl History.*** Endereço Eletrônico: http://www.perl.org/press/history.html. Data da Consulta: 01/10/2000.

[BUT1997] BUTTERICK, Szeto, SPENCER, McKirchy. **Interatividade na *Web*.** São Paulo : Berkeley, 1997.

[CRA1994] CRAIG, John Clark Craig. **Microsoft Visual Basic.** São Paulo : Makron Books, 1994.

[EDW1998] EDWARDS, Michael. **Microsoft JScript Version 5.0 Adds Exception Handling** 1998. Endereço Eletrônico: http://msdn.microsoft.com/workshop/languages/jscript/handling.asp. Data da consulta: 20/05/2000.

[FOG1997] FOGHLÚ, Mícheál Ó. **Perl 5 Guia de referência rápida**. Trad. Elisa M. Ferreira. Rio de Janeiro : Campus, 1997

[GER1999] GERVARD, Michel. **Usando ActiveX** 1999. Endereço Eletrônico: http://www.astentech.com/tutorials/ActiveX.html#What. Data da consulta: 20/05/2000.

[GHE1992] GHEZZI, Carlo. **Conceitos de linguagens de programação.** Rio de Janeiro : Campus, 1992.

[HER1997] HERRMANN, Eric. **Programação CGI com PERL 5**. Rio de Janeiro : Campus, 1997.

[HOF1996] HOFF, Arthur Van, SHAIO, Sami, STARBUCK, Orca. **Ligado em *Java***. São Paulo : Makron Books, 1996.

[MAR1986] MARSHALL, Garry J. **Linguagens de programação para micros**. Rio de Janeiro : Campus, 1986.

[MAR1999] MARCORATTI, José Carlos. **ASP, ADO e banco de dados na Internet.** Florianópolis : Visual Books, 1999.

[MCF1998] MCFEDRIES, Paul. **Guia incrível para criação de páginas *Web* com HTML**. trad. Elaine Pezzoli. São Paulo : Makron Books, 1998.

[PER1996] PERRY, Paul J. ***Creating Cool Web Applets with Java***. Foster : IDG Books, 1996.

[RAM1997] RAMALHO, José Antônio. **HTML Avançado.** São Paulo : Makron Books, 1997.

[RIT1998] RITCHEY, Tim. ***Java* e *Java Script* para Netscape.** São Paulo : Quark, 1998.

[STA1996] STAUFFER, Todd. **Dominando o essencial HTML 3.2**. Rio de Janeiro : Erica, 1996.

[STH1996] STHANEK, William Robert. **HTML CGI SGML VRML *JAVA WEB Publishing*.** Indianapolis : Sams Net, 1996.

[UNI2000] UNICAMP, Centro de Computação da. **Desenvolvimento de aplicações para internet.** 2000. http://www.ccuec.unicamp.br

[VEN1996] VENETIANER, Tomas. **HTML desmistificando a linguagem da Internet.** São Paulo : Makron Books, 1996.