1. A Plataforma Web

A maior parte da Web são estáticas. Contém informações e links. Às vezes mexem uma imagem, às vezes têm uma interface gráfica que sugere alguma interatividade. As páginas realmente interativas, que são objeto deste curso, são aquelas que não são simplesmente "páginas", mas que funcionam como *interfaces* para aplicações, sejam simples ou bastante complexas.

A grande vantagem em desenvolver tais interfaces baseadas em Web é a simplicidade. Para criar a interface do usuário não precisa-se mais que saber criar uma página. Pode-se usar apenas HTML ou até uma ferramenta gráfica como o *DreamWeaver* da Macromedia ou o Microsoft *FrontPage*.

Criar a interface do usuário é fácil. Programar a interface da página HTML com uma aplicação no servidor pode não ser. As ferramentas usadas para gerar formulários freqüentemente cuidam do desenvolvimento do código interativo que irá tratar os dados que estes irão receber. O problema é que tais ferramentas resolvem apenas os problemas mais comuns, e quase sempre, usando tecnologias proprietárias. Para fazer uma página oferecer mais que HTML já oferece é quase sempre preciso saber usar uma outra linguagem que possua recursos de programação. Tarefas simples como validação de formulários ou contagem de acessos não são solucionadas usando-se apenas HTML. É preciso recorrer a outras tecnologias.

Existem várias tecnologias que oferecem recursos além do HTTP básico. Nem todas são tecnologias abertas. Às vezes é aceitável usar tecnologias proprietárias. Outras vezes não. Tudo depende de onde ela é aplicável. Elas podem ser divididas em dois grupos: as que são executadas a partir do browser (lado-cliente), e as que são executadas a partir do servidor Web (lado-servidor).

1.1. Soluções lado-cliente

As tecnologias lado-cliente podem começar a executar a partir do momento em que o browser carrega uma página. HTML. O servidor não entende HTML nem tenta ler código misturado com ele, a não ser que sua configuração seja alterada para tal. Mas o browser precisa saber interpretá-lo. Além do HTML, um browser pode ser capacitado a interpretar diversas outras linguagens e extensões.

As soluções lado-cliente, portanto, dependem de suporte pelo browser. Se o browser não suportar a tecnologia, pode desde simplesmente não executar, como provocar perda de

informação. É possível usar soluções lado-cliente que não dependam de forma alguma da comunicação com o servidor Web. Se a conexão com o servidor se perde, depois que toda a página é carregada, grande parte da sua funcionalidade é mantida, pois, em geral, os componentes, instruções e/ou programas estão rodando na máquina do cliente.

As principais tecnologias interativas lado-cliente, além do próprio HTML, são: *scripts* do HTML como JavaScript, JScript e VBScript; objetos ou componentes como applets Java, *plug-ins* Flash, Shockwave e controles ActiveX, outros *plug-ins* e "dynamic HTML" (DHTML). Além das tecnologias interativas, outras tecnologias existem, que podem ou não introduzir elementos interativos. As mais recentes são a tecnologia de *Cascading Style Sheets* (CSS) ou folhas de estilo, e o *Extended Markup Language* (XML) que permite que o provedor de conteúdo desenvolva uma versão personalizada de sua linguagem de formatação de página.

Soluções lado-cliente são mais vulneráveis a incompatibilidades pois não há como o provedor de conteúdo garantir que os browsers dos clientes suportem totalmente as tecnologias utilizadas. Isto é comum até com as tecnologias mais populares, como JavaScript.

JavaScript é uma linguagem de roteiro (script), de propriedade da *Netscape* e VBScript é uma linguagem de roteiro baseada no VB, de propriedade da *Microsoft*. Os códigos-fonte VBScript ou JavaScript são embutidos *diretamente* no HTML de uma página e interpretado linha-por-linha pelo browser.

Para usar componentes Java, plug-ins, activeX, um objeto executável é carregado e exibido na área da página. O código, neste caso, não é visível ou interpretado como no caso dos *scripts*. Componentes não são na verdade *interpretados* pelo browser, mas disparam um módulo de extensão (JVM, interpretadores Flash, Real-audio, etc.) para executá-los.

Os componentes mais populares são os applets e controles ActiveX. Os plug-ins (extensões 'pesadas') são bastante utilizados também mas com restrições, já que dependem da plataforma onde o browser está instalado. Muita coisa que se fazia com *plug-ins* já se faz com componentes. Vários *plug-ins* populares já foram incorporados definitivamente nos principais browsers (suporte a som, vídeo, VRML, etc.). Um dos mais populares é o *Flash* da Macromedia.

As aplicações *lado-cliente* geralmente são usadas quando a tarefa a ser executada não depende de comunicação com o servidor Web (embora freqüentemente sejam usadas juntamente com soluções lado-servidor). Exemplos são a validação de campos de formulários antes do envio, a realização de operações aritméticas simples, a realização de funções simples baseadas na ação do usuário, etc. Em resumo: se não é necessário enviar ou recuperar algo do servidor, pode-se usar as soluções lado-cliente.

Mas fazer tudo no browser nem sempre é suficiente. Para contar o número de acessos a uma determinada página, por exemplo, pouco servem as funções matemáticas do lado do cliente se estas não puderem guardar em um disco no servidor um registro com os acessos anteriores. Para isto, usamos soluções lado-servidor.

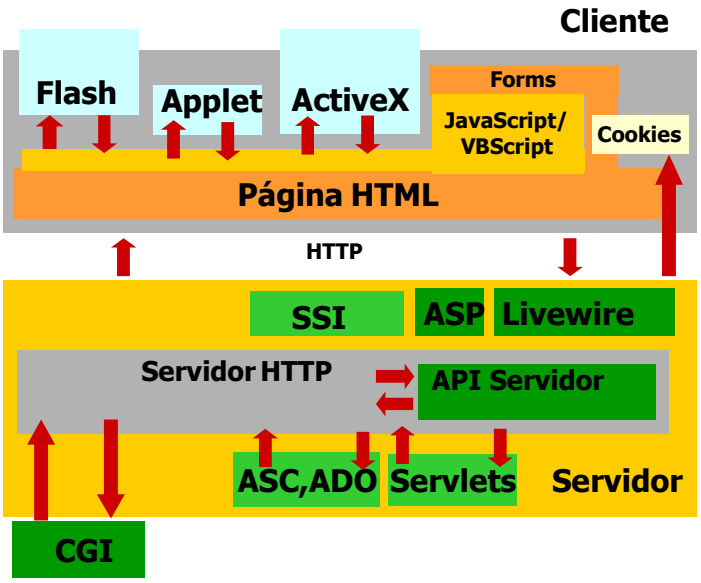
1.2. Soluções lado-servidor ou cliente-servidor

Estas soluções dependem da comunicação entre o cliente (browser) e servidor. Os dados ou ações realizadas geralmente resultam da execução de um programa ou rotina no servidor.

A comunicação cliente-servidor sempre ocorre via protocolo HTTP. É o que garante que qualquer browser vai conseguir se comunicar com qualquer servidor Web. Portanto, qualquer transformação dos dados deve ocorrer antes do envio dos mesmos às portas de comunicação, ou depois, no caso das soluções lado-cliente. A transformação dos dados pode ser feita em vários níveis e depende da interferência nas tarefas comuns do servidor Web. Vários servidores possuem tecnologias proprietárias para este fim. O nível mais baixo da programação do servidor Web ocorre através da interface CGI, *Common Gateway Interface*. A programação CGI é de *baixo nível* porque lida diretamente com a manipulação do formato dos dados enviados pelo servidor, como a criação de cabeçalhos, definição de tipos de dados, etc.

CGI já foi a única forma de interatividade via Web. Hoje, a cada dia, vem sendo substituída por soluções mais eficientes, muitas delas proprietárias. As tecnologias mais comuns, que diferentemente do CGI, dependem do tipo e plataforma do servidor são o ISAPI, da *Microsoft*, e o NSAPI, da *Netscape*. Estas tecnologias são módulos ou "plug-ins" que permitem que um programador desenvolva extensões destinadas ao tratamento de dados e comunicação pelo servidor, podendo substituir totalmente o CGI com ganhos de desempenho, porém com um razoável acréscimo de complexidade e perda de portabilidade.

Entre as SAPIs e o CGI, existem os *componentes* para servidor. São programas em Java (*servlets*) ou objetos ActiveX que fazem o mesmo que as SAPIs, porém mantêm uma portabilidade maior. Mais simples ainda são os *scripts* ou roteiros de código embutidos em páginas Web, que servidores devidamente configurados usam para realizar tarefas de transformação de dados e comunicação com outros programas (ASP, JSP, LiveWire, PHP, Cold Fusion). Com essas tecnologias, o conteúdo da página pode ser alterado no próprio servidor no momento do envio, através da interpretação de *scripts*, que também servem de *gateway* com aplicações no servidor, como bancos de dados. Todas essas tecnologias substituem completamente o CGI e são geralmente mais eficientes. Todas, também, operam sobre o servidor HTTP, da mesma forma que CGI. Conhecer CGI, portanto, é bastante útil para o domínio de qualquer uma dessas tecnologias.



A figura ao lado ilustra algumas das principais tecnologias utilizadas no desenvolvimento de páginas interativas e aplicações intranet.

1.3. A plataforma Web

A World Wide Web é um serviço TCP/IP baseado no *protocolo de nível de aplicação* HTTP (*HyperText Transfer Protocol*) – *Protocolo de Transferência de Hipertexto*. A plataforma Web é o meio virtual formado pelos servidores HTTP (servidores Web que mantém sites), clientes HTTP (browsers) e protocolo HTTP (a língua comum entre o cliente e o servidor).

Hipertexto

Hipertexto é uma forma não linear de publicação de informações onde palavras que aparecem no decorrer do texto podem levar a outras seções de um documento, outros documentos ou até outros sistemas de informação, fugindo da estrutura linear original de um texto simples. O hipertexto baseia-se em ligações entre dois pontos chamados de *âncoras*. As ligações entre as âncoras são chamadas de *vínculos* (*links*). Vínculos de hipertexto são implementados em textos publicados na Web usando uma linguagem declarativa chamada *HTML - HyperText Markup Language*.

HTML

HTML é usada para *marcar* um arquivo de *texto simples* (texto simples é texto sem formatação alguma, visualizável em *qualquer* editor de textos). Se um arquivo de texto simples receber uma extensão de nome de arquivo ".html" ou ".htm", um navegador como o Internet Explorer irá tentar interpretá-lo como HTML. Dentro do texto, pode-se definir descritores (ou comandos HTML) entre os símbolos "<" e ">":

```
<h1>Arquivo de texto</h1>
<p>Este é o <i>primeiro</i> parágrafo.</p>
```

Os descritores só serão visíveis quando o arquivo for visualizado em um editor de textos (como o Bloco de Notas do Windows). Ao ser visualizado em um programa capaz de entender HTML, apenas o texto aparece, com uma aparência determinada pelos descritores:

Arquivo de texto

Este é o *primeiro* parágrafo.

O texto com marcadores é chamado *código-fonte HTML*. O código-fonte é usado para produzir a página visualizada o browser que é chamada de *página HTML* ou *página Web*.

O browser, por ser capaz de exibir diversos tipos de informação, depende totalmente da *extensão do arquivo* para saber o que fazer com ele. Se a extensão ".htm" ou ".html" não estiver presente ou se o arquivo tiver a extensão ".txt", o browser exibirá o código-fonte.

Além da formatação da página, o HTML é responsável também pela inclusão de imagens e definição dos links que permitem a navegação em hipertexto.

Servidor HTTP

O serviço HTTP funciona de forma semelhante ao serviço *FTP - File Transfer Protocol* (protocolo de comunicação usado na Web para operações de transferência de arquivos). Ambos oferecem aos seus clientes um *sistema de arquivos virtual* onde podem localizar *recursos* (arquivos, programas, etc.) e transferi-los de um computador para outro. O sistema virtual pode ter uma *hierarquia* própria e totalmente diferente do *sistema de arquivos real do computador,* ao qual está vinculado. Geralmente um servidor tem acesso a uma área restrita da máquina e só permite a visualização dos arquivos lá contidos. O sistema de arquivos virtual usa uma notação diferente daquela usada pelo sistema real. Por exemplo, considere o seguinte sistema de diretórios no Windows:

```
C:\
C:\Windows
C:\Documentos
C:\Documentos\Web\
C:\Documentos\Web\Imagens
C:\Documentos\Web\Videos
```

Suponha que um servidor HTTP foi instalado nessa máquina. Na instalação, ele é configurado para administrar um sistema de diretórios a partir de um certo diretório. Suponha que esse diretório é C:\Documentos\Web\. Para o servidor, isto é seu diretório raiz. No sistema de diretórios virtual, o diretório raiz de um servidor é chamado de / (barra). O sistema de arquivos virtual (a parte que um browser poderá ter acesso) é:

```
/                (C:\Documentos\Web\)
/Imagens         (C:\Documentos\Web\Imagens)
/Videos          (C:\Documentos\Web\Videos)
```

Um browser jamais terá acesso ao diretório Windows, por exemplo. A principal função de um servidor Web é, portanto, *administrar* um sistema de arquivos e diretórios virtual e *atender à requisições dos clientes* HTTP (os browsers), que, na maior parte das vezes, enviam comandos HTTP pedindo que o servidor devolva um ou mais arquivos localizados nesses diretórios. Os pedidos são feitos através de uma sintaxe especial chamada de URI.

URIs (URLs[1])

Todas as comunicações na plataforma Web utilizam uma sintaxe de endereçamento chamada *URI - Uniform Resource Identifier* - para localizar os recursos que são transferidos. O

[1] URIs também são frequentemente chamadas de URLs (Uniform Resource Locators). A URL é um tipo particular de URI mas, para a nossa discussão, essa distinção é irrelevante. A documentação HTML (especificação) sempre refere-se à essa sintaxe como URI.

serviço HTTP depende da URI que é usada para localizar *qualquer coisa* na Internet. Contém duas informações essenciais: 1) COMO transferir o objeto (o protocolo); 2) ONDE encontrá-lo (o endereço da máquina e o caminho virtual). URIs tipicamente são constituídas de três partes:

- mecanismo (protocolo) usado para ter acesso aos recursos (geralmente HTTP)
- nome da máquina (precedido de `//`) onde o serviço remoto é oferecido (e a porta, se o serviço não estiver em uma porta padrão) ou outro nome através do qual o serviço possa ser localizado (sem `//`).
- nome do recurso (arquivo, programa) na forma de um caminho (no sistema de arquivos virtual do servidor) onde se possa encontrá-lo dentro da máquina.

Sintaxe típica:

```
protocolo://maquina:porta/caminho/recurso
```

As URIs mais comuns são os endereços da Web, que utilizam o mecanismo HTTP para realizar a transferência de dados:

```
http://www.maquina.com.br/caminho/para/minha/página/texto.html
```

Veja algumas outras URLs:

- `ftp://usuario:senha@maquina.com/pub/arquivo.doc`
  Acesso a servidor FTP que exige usuário e senha para fazer download de arquivo.doc
- `nntp://news.com.br/comp.lang.java`
  Acesso a servidor de newsgroups para ler o grupo comp.lang.java
- `news:comp.lang.java`
  Acesso ao grupo comp.lang.java através de servidor default (definido localmente)
- `http://www.ibpinet.net/`
  Acesso à página default disponível no diretório raiz do servidor Web de www.ibpinet.net
- `http://www.algumlugar.com:8081/textos/`
  Acesso à página default disponível no diretório textos do servidor Web que roda na porta 8081 da máquina www.algumlugar.net
- `http://www.busca.com/progbusca.exe?opcoes=abc&pesquisa=dracula`
  Passagem de parâmetros de pesquisa para programa de busca progbusca.exe que terá sua execução iniciada pelo servidor HTTP que roda na porta 80 (default) de www.busca.com.
- `http://www.ibpinet.net/helder/dante/pt/inferno/notas_4.html#cesar`
  Acesso à uma seção da página HTML notas_4.html identificada como "cesar", localizada no subdiretório virtual /helder/dante/pt/inferno/ do servidor Web de www.ibpinet.net.
- `mailto:helder@ibpinet.net`
  Acesso à janela de envio de e-mail do cliente de correio eletrônico local.

Browser

O browser é um programa que serve de interface universal a todos os serviços que podem ser oferecidos via Web. É para a plataforma Web o que o sistema operacional (Windows, Linux, Mac) é para o computador. A principal função de um browser é *ler* e exibir o conteúdo de uma página Web. A maior parte dos browsers também é capaz de exibir vários outros tipos de informação como diversos formatos de imagens, vídeos, executar sons e rodar programas.

Um browser geralmente é usado como *cliente HTTP* – aplicação de rede que envia requisições a um servidor HTTP e recebe os dados (uma página HTML, uma imagem, um programa) para exibição, execução ou *download*. Browsers também podem ser usados *off-line* como aplicação local do sistema operacional para navegar em sistemas de hipertexto construídos com arquivos HTML (sem precisar de servidor HTTP). Nesse caso, não se comportam como clientes HTTP (já que não estão realizando operações em rede) mas apenas como *visualizadores de mídia interativa* capazes de visualizar HTML, imagens, sons, programas, etc.

Como os browsers precisam interpretar vários tipos de código (código de imagens GIF, JPEG, código de programas Java e Flash, códigos de texto HTML ou texto simples) é preciso que ele saiba identificar os dados que recebe do servidor. Isto não é a mesma coisa que identificar um arquivo carregado do disco local, onde ele pode identificar o tipo através da extensão. Quando os dados chegam através da rede, a extensão não significa nada. O servidor precisa informar ao browser o que ele está enviando. Na Web, isto é feito através de uma sintaxe padrão para definir tipos chamada *MIME - Multipart Internet Mail Extensions*.

Tipos MIME

MIME é uma sintaxe universal para identificar tipos de dados originalmente utilizada para permitir o envio de arquivos anexados via e-mail. O servidor Web possui, internamente, tabelas que relacionam os tipos de dados (na sintaxe MIME) com a extensão dos arquivos por ele gerenciados. Quando ele envia um conjunto de bytes para o browser, envia antes um *cabeçalho* (semelhante ao cabeçalho de e-mail) informando o *número de bytes* enviados e o *tipo MIME* dos dados para que o browser saiba o que fazer com a informação. A sintaxe MIME tem a seguinte forma:

```
tipo/subtipo
```

O *tipo* classifica um conjunto de bytes como imagens, textos, vídeos, programas (aplicações), etc. O *subtipo* informa características particulares de cada tipo. Não basta saber que o arquivo é uma imagem, é preciso saber qual o *formato*, pois o códigos usados para produzir imagens de mesma aparência gráfica podem diferir bastante entre si. Tanto no servidor como no browser há tabelas que relacionam extensões de arquivo a tipos MIME:

```
image/jpeg              .jpe, .jpg, .jpeg
image/png               .png
image/gif               .gif
text/html               .html, .htm, .jsp, .asp, .shtml
```

```
text/plain              .txt
x-application/java      .class
```

O protocolo HTTP

O protocolo HTTP funciona de forma semelhante ao protocolo FTP – *File Transfer Protocol* (protocolo de comunicação usado na Web para operações de transferência de arquivos). Uma máquina servidora que oferece o serviço HTTP geralmente usa a porta TCP/IP de número 80 (ou 443 para transações seguras com SSL), reservada para esse serviço. Diferentemente do FTP, o protocolo HTTP não mantém uma sessão aberta entre transferências. Cada operação consiste de um *único* envio de requisição pelo cliente, seguido de uma única resposta. Ou seja, há uma conexão, um envio de requisição, um recebimento de resposta e o fim da conexão. Não há persistência de informações entre as transações (não é possível manter o estado de variáveis entre páginas). HTTP é, portanto, um protocolo que não mantém estado nem conexão.

Usar HTTP para transferir uma página com vários objetos (imagens, componentes) exigirá vários acessos. Cada objeto transferido conta como um acesso e o servidor não conhece relação alguma entre eles. Uma página com 50 imagens precisará de 51 acessos para que possa ser exibida por completo na tela de um browser.

Comunicação entre clientes e servidores HTTP

A comunicação entre o browser e o servidor ocorre através de requisições e respostas HTTP. Cada requisição contém uma linha com um método HTTP detalhando como e o que deve ser pedido ao servidor, seguido por um cabeçalho com informações adicionais sobre a requisição. Cada requisição recebe do servidor uma resposta que contém uma linha onde informa o estado da conexão, seguida também por um cabeçalho que descreve os dados que serão devolvidos em seguida. Estes cabeçalhos também causam a definição de propriedades no sistema do servidor (*variáveis de ambiente)* que possibilitam a realização de operações interativas como o CGI. A definição dos cabeçalhos é baseada na especificação RFC822.

A tarefa do servidor é localizar o arquivo, identificar o seu tipo de dados, montar um cabeçalho que contenha informações sobre este arquivo (tipo de dados, tamanho, data de modificação, etc.) e enviar as informações para a saída padrão. O servidor não analisa o conteúdo do arquivo mas simplesmente o redireciona à porta HTTP, portanto o servidor não precisa conhecer nem ser capaz de interpretar HTML.

Quando um visitante clica em um vínculo de hipertexto ou quando um usuário de browser digita uma nova URI na barra de endereços, o browser envia uma requisição para o servidor. Este, por sua vez, enviará sempre uma resposta, havendo ou não sucesso. Uma transação típica está mostrada a seguir:

Browser envia para servidor...

```
GET /book/ch1.html HTTP/1.0
Host: volans.argo.net
```

```
Port: 80
Accept: text/html
Accept: image/jpg
User-Agent: Microsoft Internet Explorer (Mozilla 3.0 Compatible)
```

Se o servidor *encontra* o arquivo, retorna

```
HTTP/1.0 200 OK
Date: Friday, June 13, 1977
(... outros cabeçalhos...)
Content-type: text/html

<HTML><HEAD>
<TITLE> Capitulo 3</TITLE>
(...)
```

Se não acha...

```
HTTP/1.0 404 Not Found
Date: Friday, June 13, 1977
(... outros cabeçalhos...)
Content-type: text/html

<HTML><HEAD>
<TITLE>404 File Not Found</TITLE>
(...)
```

Com as informações contidas no cabeçalho enviado pelo browser, o servidor pode tomar decisões de enviar ou não o arquivo, de redirecionar, de utilizar um meio mais eficiente de retorno de dados, etc. Já o browser depende das informações que chegam no cabeçalho criado pelo servidor para saber o que fazer com os bytes que está recebendo, saber quando parar de ler a porta do servidor (quantos bytes há para ler), o tipo dos dados, se deve guardar as informações no cache, etc.

1.4. O que é CGI

O servidor sempre retornará algo para o cliente, após receber uma requisição. A resposta pode ser o recurso que ele de fato pediu ou uma mensagem de erro. Se o arquivo solicitado for uma página HTML, uma imagem GIF ou qualquer outro arquivo suportado pelo browser, este saberá como exibi-lo. No caso de outros formatos, o browser ou redireciona para outra aplicação, ou tenta salvar em disco.

Se estiver configurado para oferecer suporte a CGI, o servidor poderá, ao invés de simplesmente encontrar um arquivo e enviá-lo ao browser, tentar executá-lo, funcionando como um *gateway* para intermediar o envio e recebimento de dados entre o arquivo-programa e o browser. Este programa, ao ser executado, pode ainda comunicar-se com outros programas, estendendo o *gateway* além dos limites da máquina servidora.

CGI é a sigla para *Common Gateway Interface* (Interface Comum de *Gateway*). É um padrão W3C suportado por todos os servidores Web que estabelece parâmetros para possibilitar a execução de aplicações através do servidor Web. É a forma mais trivial para realizar esta comunicação. O programa CGI necessita ser identificado e chamado pelo browser via uma requisição HTTP da mesma forma como ocorre com um arquivo comum. A diferença é que o servidor ao encontrá-lo vai tentar executá-lo em vez de simplesmente redirecioná-lo enviá-lo para a saída padrão. Para que o uso de CGI seja possível, é preciso que o servidor e as aplicações CGI estejam configuradas para funcionar desta forma. O browser nada tem a ver com o fato de um arquivo ser devolvido ou executado pelo servidor. Tudo depende da configuração no servidor que tanto pode mandar rodar *acesso.exe* como devolvê-lo para download.

O uso mais comum do CGI é como intermediário para aplicações mais complexas, principalmente de acesso e tratamento de dados dinamicamente. A figura abaixo ilustra a utilização da interface CGI na realização da comunicação entre uma interface do usuário baseada em página HTML com um banco de dados relacional legado acessível através da máquina onde roda o servidor Web.

Este curso pretende mostrar como configurar CGI no servidor e como desenvolver aplicações que atendam aos requisitos necessários para funcionarem como programas CGI. Não serão explorados recursos específicos das linguagens usadas com CGI.

1.5. Questões

Marque com (V) as afirmações verdadeiras e com (F) as afirmações falsas.

___ a) É preciso que um browser suporte JavaScript para entender páginas que contém código ASP ou PHP, mesmo que o servidor suporte estas tecnologias.

___ b) Para carregar uma página que possui 5 imagens e uma applet Java, o browser precisa fazer pelo menos 7 requisições independentes ao servidor.

___ c) O código HTML dos arquivos armazenados no servidor é sempre analisado e interpretado pelo servidor Web antes de ser enviado para o browser.

___ d) É preciso que o servidor Web suporte Java e JavaScript do lado do servidor (ASP por exemplo) para que possa servir páginas HTML com JavaScript e applets Java aos seus clientes.

___ e) É possível implementar um contador de acessos – que conta o número de vezes que uma determinada página foi acessada – usando *apenas* técnicas de interatividade no cliente como applets Java ou JavaScript.

2. Configuração e utilização do servidor

2.1. O servidor Apache

O servidor Web Apache está disponível para as plataformas Windows e Unix (Linux). A aplicação do servidor não possui interface gráfica para administração. Toda a sua configuração é feita através dos seus arquivos de configuração.

Após a instalação, o diretório da aplicação possui a seguinte estrutura:

```
htdocs/
bin/
modules/
icons/
logs/
conf/
cgi-bin/
proxy/
```

O diretório htdocs/ é o diretório raiz de documentos, ou seja, ele corresponde ao diretório virtual "/" quando o servidor for acessado através de uma requisição do browser contendo uma URI. Qualquer diretório criada abaixo de htdocs/ automaticamente estará disponível através da URI `http://nome.da.maquina/diretorio/`. É possível, porém, oferecer acesso a outros diretórios da máquina que não estão abaixo de htdocs/

O diretório bin/ contém o programa executável que mantém o servidor no ar. O diretório logs/ contém os históricos de acesso e de erro com todas as informações enviadas pelos clientes que os provocaram desde a primeira execução do servidor.

O diretório conf/ contém os arquivos necessários à configuração do servidor. O mais importante é httpd.conf que permite criar diretórios virtuais, implantar o CGI, mudar a porta do servidor, etc.

O diretório cgi-bin/ foi previamente configurado para guardar arquivos executáveis que funcionarão como CGI. Isto pode ser alterado no arquivo httpd.conf.

A primeira coisa que precisamos fazer com o servidor Apache instalado no laboratório é mudar a sua porta. Como na mesma máquina temos dois servidores (um servidor Microsoft e outro Apache), eles não podem simultaneamente ocupar a mesma porta. Escolheremos a porta 8080 para o servidor Apache. Isto significará que todas as URIs direcionadas a este servidor deverão conter o número 8080 precedido por ":" logo após o nome da máquina.

Para fazer a alteração, abra o arquivo httpd.conf e procure pela palavra "Port". Ela deve indicar o valor 80. Mude para 8080:

```
Port 8080
```

Agora desligue o servidor e ligue-o de novo. Para testar, tente acessar a sua home-page através da URI `http://suamaquina:8080/`.

2.2. Publicação de páginas e diretórios virtuais

Você pode publicar todo o seu site dentro de htdocs/ e substituir a página index.html por uma que você tenha criado. Se desejar, pode também alterar o DocumentRoot através do httpd.conf, assim como outras configurações iniciais como o nome do arquivo de índice (index.html).

Se você tiver outros diretórios com páginas HTML, pode publicá-las no servidor criando uma URI com um novo diretório virtual que aponte para o seu diretório fixo. Por exemplo, você pode colocar todas as páginas do curso de JavaScript no ar mapeando c:\javascript\ à URI /js/. Isto pode ser feito com Alias (procure no httpd.conf):

```
Alias   /js/  c:\javascript\
```

Você pode ter várias instruções Alias e assim mapear todos os diretórios que quiser deixar acessíveis via Web.

2.3. O servidor Personal Web Server

O servidor *Personal Web Server* possui uma interface gráfica para configuração e administração. A home-page inicial está localizada em `c:\Inetpub\Wwwroot\`. Há vários diretórios pré-configurados para permitir permissões diferentes de acesso e a execução de scripts. Para rodar o administrador do PWS clique no ícone correspondente a partir do menu Iniciar ou na barra de tarefas (caso ele já esteja no ar).

3. Usando CGI

Como foi mencionado acima, CGI, ou *Common Gateway Interface* é um mecanismo que permite que browsers e outros clientes Web executem programas em um servidor Web. São largamente utilizados para produzir páginas dinâmicas, para processar conteúdo de formulários, acessar bancos de dados entre outras aplicações.

Um servidor Web que implementa a especificação CGI é capaz de executar aplicações solicitadas por um cliente (browser) remoto. Pode ser qualquer aplicação. Por exemplo, suponha um servidor Web instalado em uma máquina ou rede que controlem a abertura e fechamento de uma porta elétrica. Se for possível construir uma aplicação local nessa máquina ou rede que abra e feche a porta, será possível construir uma aplicação CGI e instalá-la no servidor Web para que um cliente remoto, através de uma página recebida em seu browser, possa também abrir e fechar a porta remotamente.

CGI oferece uma interface de baixo nível para o controle das requisições e respostas HTTP recebidas e enviadas por um servidor Web. Para programar em CGI, portanto, é preciso conhecer um pouco de como o browser se comunica com o servidor e vice-versa. Ou seja, conhecer o funcionamento do protocolo HTTP, a sintaxe de suas requisições, métodos, respostas e cabeçalhos.

Pode-se usar *qualquer* linguagem de programação para escrever programas que irão executar como programas CGI. A especificação só determina regras para a entrada e saída de dados, portanto, qualquer linguagem que atenda aos requisitos mínimos exigidos pode ser usada. Em outras palavras, um programa CGI é uma caixa-preta que espera receber, na entrada, os dados em um certo formato padrão, realizar quaisquer operações necessárias com ou sem esses dados e retornar, na saída, dados em um outro formato padrão. Não interessa, portanto, a linguagem de programação usada. Deve-se usar aquela que ofereça os melhores recursos para as tarefas que desejamos realizar, e que tenha recursos suficientes para ler variáveis de ambiente do sistema (dados de entrada) e gerar dados em formato binário. Java, embora seja bastante popular em outras aplicações Web, é bastante fechada quanto ao acesso a propriedades do sistema, tornando o desenvolvimento de certas aplicações CGI mais difícil. Linguagens que só são capazes de gerar texto como saída também não se adaptam bem à essa finalidade.

Perl tem se tornado um padrão de fato por estar disponível em quase todas as plataformas que possuem servidores Web, ser interpretada, tornar fácil o desenvolvimento de *scripts* simples com transformação de dados e apresentar uma sintaxe parecida com C e linguagens de

administração de sistemas Unix. Na maior parte dos exemplos de CGI apresentados neste curso, usaremos programas escritos na linguagem Perl.

3.1. Implantando o CGI

Para que o servidor decida tentar executar um objeto solicitado pelo browser em vez de enviá-lo à saída padrão (para download) é necessário configurá-lo para suportar CGI.

O comando GET simplesmente solicita um objeto a um browser. Se o link de GET for para um programa chamado "contador.exe" em um servidor *Windows*, ele irá enviar o programa para o browser, que, por sua vez, apresentará uma janela ao usuário perguntando se ele deseja fazer o download do programa. Mas se o servidor estiver configurado de tal forma a identificar "contador.exe" como sendo um *programa CGI*, este tentará executar o programa e, caso tenha sucesso, enviará para a saída padrão a informação gerada na saída do mesmo.

A forma de configurar CGI varia de servidor para servidor. Na maioria dos servidores pode-se definir um programa CGI como sendo qualquer executável que esteja em um diretório especial, configurado para permitir a execução de programas CGI, ou definir CGI como um *tipo de dados*. Com esta última configuração, um programa CGI pode residir em qualquer lugar e será identificado por uma extensão (por exemplo ".cgi").

Servidores Apache (Unix e Windows)

Para definir uma área de programas CGI em servidores HTTP tradicionais (CERN, NCSA, Apache) é necessário modificar um ou mais arquivos de configuração. Um diretório virtual chamado `/cgi-bin/` para programas CGI armazenados em `/dev/lib/httpd/cgi-bin/` pode ser definido em um servidor Apache, acrescentando no arquivo de configuração `httpd.conf` (ou `srm.conf`) a linha:

```
ScriptAlias   /cgi-bin/   /dev/lib/httpd/cgi-bin/
```

Esse arquivo pode ser encontrado abaixo do subdiretório `conf/` de qualquer instalação do Apache, seja *Windows* ou *Linux*. Abra o arquivo em um editor de textos e procure por "ScriptAlias". Geralmente já existe um diretório configurado ou a instrução pode estar comentada (precedida de um #). A alteração acima fará com que programas armazenados em `/dev/lib/httpd/cgi-bin/` sejam tratados como programas CGI pelo servidor, e, consequentemente, executados quando o browser os requisitar através da URI `http://suamaquina/cgi-bin/`. Em servidores Apache para Windows, o procedimento é idêntico:

```
ScriptAlias   /progs/   c:/Apache/cgi-bin/
```

Desta vez o diretório `c:\Apache\cgi-bin\` foi mapeado à URI `http://suamaquina/progs/` (considerando que a porta do servidor seja 80). Para definir CGI como um *tipo de dados*, acrescente (ou remova o comentário) no `httpd.conf` (ou `srm.conf`):

```
AddHandler    cgi-script    .cgi
```

Servidores Microsoft

Nos sistemas Windows existem três tipos diferentes de CGI. A maior parte dos servidores não é capaz de distinguir entre eles automaticamente e é necessário configurar o CGI para o tipo correto a ser usado. Em servidores Microsoft, alguns tipos exigem uma configuração bem mais complexa que outros.

O primeiro tipo é o *Win-CGI*, que suporta programas que rodam sob *Windows* (e não DOS). O *DOS-CGI* também chamado simplesmente de cgi-bin suporta programas que rodam através do *prompt do MS-DOS* como arquivos batch, e executáveis DOS. Os servidores Microsoft lidam com os dois. Arquivos que devem ser executados através da interface CGI devem ser corretamente configurados e colocados dentro de um diretório, administrado pelo servidor Web, que tenha permissão de execução. Em outros servidores Windows como os servidores Netscape e O'Reilly, é preciso distinguir os programas que rodam em ambiente DOS dos programas que rodam em Windows.

Finalmente o *Shell-CGI* suporta programas que precisam de outro programa para executar, como programas interpretados que exigem que se chame primeiro o sistema de *runtime*. Exemplos são programas em Perl, Java ou Basic interpretado. Para fazer com que esses programas funcionem em servidores Windows é preciso configurá-los no sistema Windows (fazendo uma associação do tipo de arquivo com o interpretador) e no servidor (não existe uma maneira uniforme). Entre os servidores disponíveis para Windows, o que suporta CGI em Perl de forma mais simples é o *O'Reilly Website Server*. O servidor PWS da *Microsoft* exige não só a associação da extensão .pl com o interpretador, como a alteração e criação de chaves no Registro do *Windows*.

Devido à complexidade da instalação CGI com Perl em servidores Microsoft, usaremos apenas o servidor Apache para desenvolver aplicações Web com CGI. As aplicações devem funcionar tanto em sistemas Linux como em Windows. Voltaremos ao servidor PWS quando formos demonstrar e desenvolver aplicações ASP, servlets e JSP.

3.2. Exercícios

1. Configure o servidor que está instalado na sua máquina para que passe a suportar CGI.
2. Usar programas prontos em shell, perl, C, etc. (disponíveis no laboratório) para verificar se o CGI está OK.
3. Veja a configuração CGI do servidor Unix instalado no laboratório lendo o arquivo httpd.conf (depende da instalação). Quantos diretórios CGI foram definidos? Onde (em que caminho absoluto) os programas podem ser armazenados e como podem ser chamados via uma URL?

3.3. Limitações do CGI e outras alternativas

CGI é uma portátil, flexível e suportado por todos os servidores Web. Há, porém, diversas limitações na tecnologia CGI que podem até inviabilizar certos tipos de aplicação.

Para cada requisição de um cliente, um novo processo é criado, mesmo que seja para executar a mesma aplicação e fazer as mesmas coisas. Quando raramente há mais de um cliente conectado, ou quando a requisição não ocupa tempo do servidor, isto pode não ser um problema. Porém quando há muitos clientes realizando tarefas demoradas, o desempenho despenca e pode paralisar o servidor.

Uma conseqüência desse modelo é que não há compartilhamento de recursos em CGI. Cada processo independente abre seus arquivos, roda seus programas, armazena seus dados separadamente. Não é possível implementar persistência entre requisições pois cada requisição está associada a um programa diferente.

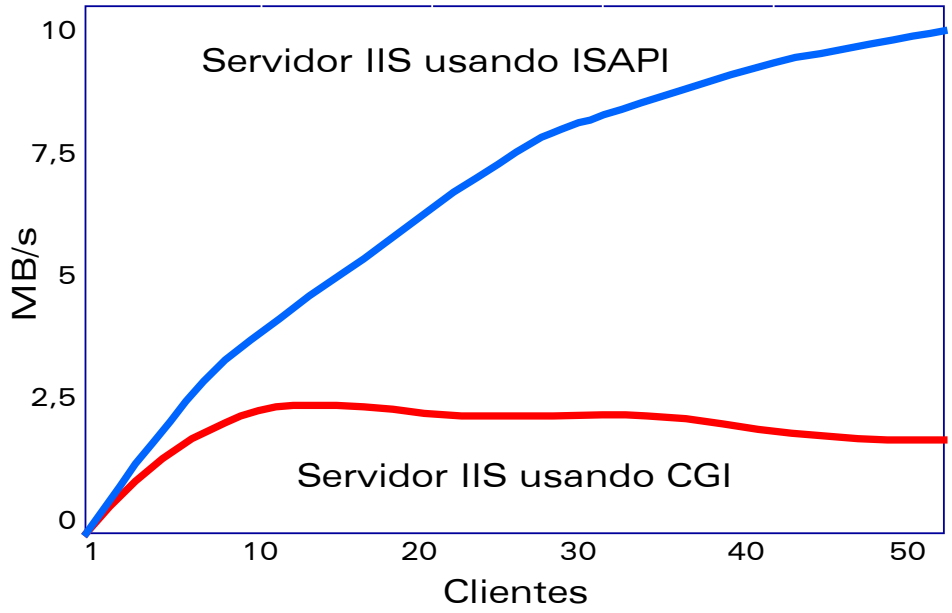
Programas CGI, como são externos ao servidor, não podem aproveitar recursos, plug-ins ou componentes do servidor. Também não é possível refletir o estado do servidor se este mudou depois que o programa foi executado.

Para eliminar ou atenuar essas limitações, têm surgido várias alternativas. As mais eficientes são proprietárias e exigem conhecimento de linguagens como C++. A *Netscape* foi a primeira a oferecer uma API para a programação do servidor, chamada NSAPI. Pouco depois, a *Microsoft* fez algo semelhante, e chamou de ISAPI. Todas eram APIs C++, e incompatíveis, porém, poderiam obter o máximo de eficiência já que faziam parte do servidor, não causavam o início de novos processos e podiam compartilhar recursos entre si.

As Server APIs (SAPI) não substituíram o CGI por não ter a mesma portabilidade e serem dependentes de fabricante. Isto sem mencionar o fato que exigem conhecimento de linguagens específicas. Outras alternativas que surgiram foram os *scripts* (roteiros) de servidor, como ASP, Livewire, Cold Fusion e componentes, como os servlets e ActiveX .

Os *scripts* são embutidos em páginas HTML especiais. Quando o browser faz uma requisição pedindo uma dessas páginas, ele primeiro analisa todo o código a procura de blocos de linguagem que deve interpretar. Embora interpretados, como os *scripts* são executados pelo próprio servidor, costumam ser mais rápidos e eficientes que CGI. Os *scripts* podem desde alterar a apresentação e informação da página até realizar conexões a bancos de dados. Os *scripts* geralmente são dependentes de fabricante de servidor, mas através de plug-ins e extensões é possível reutilizá-los em outros servidores.

Componentes agem como módulos dinâmicos do servidor. Podem aliar a eficiência das SAPIs com a portabilidade do CGI. Os componentes Java, chamados de servlets são os mais portáteis pois podem ser usados em servidores instalados em plataformas diferentes sem necessitar de compilação.



Fonte: PCMagazine Labs

São várias as opções para desenvolvimento de aplicações do lado do servidor. Qual usar? Depende de vários fatores: segurança, velocidade, custo, simplicidade. Se o objetivo é fazer um sistema simples, tipo uma busca leve, um guestbook, formulário de feedback, etc., talvez não valha a pena investir tempo em soluções muito complicadas, podendo-se aproveitar o oceano de recursos CGI disponíveis na rede. Por outro lado, se o desempenho é um fator crítico, convém analisar outras alternativas como ISAPI e *scripts* de servidor. A tabela a seguir apresenta um resumo das tecnologias mais populares, em ordem crescente de complexidade e eficiência:

Tecnologia	**Linguagens suportadas**
CGI	Perl, C, C++, Basic, Pascal, Delphi, Bourne-Shell, Java, etc.
Fast-CGI	Perl, C, C++, Basic, Pascal, Delphi, Bourne-Shell, Java, etc.
ASP e LiveWire	ASP: VBScript, JavaScript; LiveWire: JavaScript
ASC e Servlets	ASC: C++, VB, Delphi, Java; Servlets: Java
ISAPI/NSAPI	C++, VB e Delphi

CGI, porém, continua como base para estas tecnologias. A interface CGI lida com detalhes da programação do servidor em baixo-nível, se preocupando com o formato exato de cabeçalhos, requisições, etc. Conhecendo-se CGI, a escolha e uso de uma outra tecnologia torna-se muito mais fácil, pois todas utilizam os mesmos conceitos, embora ofereçam funções e métodos que permitem a programação em um nível mais alto. Aprender CGI, portanto, será útil mesmo se você decidir usar outra tecnologia mais sofisticada.

Os capítulos seguintes abordarão a interface CGI em maiores detalhes e com exercícios práticos. O foco será na sua especificação. Evitaremos entrar em detalhes quanto à implemen-

tação em uma determinada linguagem de programação, tendo a introdução à linguagem Perl somente a finalidade de facilitar a compreensão dos exemplos.

3.4. Questões

Marque com (V) as afirmações verdadeiras e com (F) as afirmações falsas.

___ a) Programas que serão executados pelo servidor usando a tecnologia CGI podem ser escritos em qualquer linguagem de programação, não apenas Perl.

___ b) Programas CGI são interpretados pelo servidor Web.

___ c) Para cada cliente que usa uma aplicação Web beseada em CGI o servidor precisará iniciar um novo processo de forma que se há 100 clientes conectados a uma aplicação de banco de dados, há pelo menos 100 aplicações rodando cuja execução foi iniciada pelo servidor Web.

___ d) A única forma de fazer um servidor Web se comunicar com um banco de dados ou outra aplicação no cliente é através da interface CGI, ou seja, é preciso que um programa CGI faça a intermediação entre a aplicação e o servidor Web.

4. Formulários

Os componentes de formulário são campos de entrada de dados dentro de um bloco HTML, como botões, caixas de seleção, caixas de texto e botões de "rádio". Eles são a principal forma de entrada de dados disponível no HTML.

Os elementos de formulário são apenas três: `<INPUT>`, `<SELECT>` e `<TEXTAREA>`, mas produzem 12 efeitos diferentes. Todos devem ser usados dentro de um bloco `<FORM>`.

O objetivo deste capítulo é apresentar esses elementos do HTML que são essenciais no desenvolvimento de aplicações Web usando CGI. Não se trata de um texto detalhado sobre o assunto mas um pequeno guia de referência.

4.1. Elemento <FORM>

O elemento `<FORM>` é o mais importante dos elementos de formulário. Ele define o "bloco" de formulário que deve ser atrelado a um programa no servidor. A sua sintaxe está mostrada abaixo:

```
<FORM
  ACTION="url para onde será enviado o formulário"
  METHOD="método HTTP (pode ser GET ou POST)"
  ENCTYPE="formato de codificação"
  TARGET="janela alvo de exibição da resposta do formulário"
 " >
      ... corpo do formulário ...
</FORM>
```

A sintaxe do elemento `FORM` elemento contém no mínimo dois atributos, para uso com CGI e demais aplicações de servidor: `ACTION`, que indica a URL do programa CGI que irá processar o conteúdo do formulário e, `METHOD`, que irá indicar o método de requisição HTTP que será usado na transferência de informações. Os outros elementos que são usados dentro do bloco `<FORM>`, sempre têm como atributo obrigatório um campo `NAME` que identifica a variável que irá armazenar um valor fornecido pelo usuário/visitante da página Web. Os dados são sempre passados ao servidor através de pares `nome=valor`, interligados pelo caractere "`&`", onde o "nome" é o identificador da variável usado pelo programa CGI e o valor do atributo `NAME` do componente de formulário:

Eis um exemplo de uso de <FORM>:

```
<FORM ACTION="http://www.acme.com/cgi-bin/pedido.pl" METHOD="POST">
    <P>Seu Nome: <INPUT TYPE="text" NAME="nome">
    <P>Seu endereço: <INPUT TYPE="text" NAME="endereco">
  (...)

</FORM>
```

Este bloco de formulário terá seus valores de entrada processados pelo programa `http://www.acme.com/cgi-bin/pedido.pl`, que será requisitado pelo browser usando o método `POST`. Na tela aparecerão dois campos de texto. Suponha que o usuário digite como nome "João Lobão" e como endereço "Rua Dois, 2". Os dados serão enviados para o programa CGI em uma string, contendo:

```
nome=Jo%E3o+Lob%E3o&endereco=Rua+Dois%22+2
```

É função do programa CGI, extrair as informações essenciais dessa string.

A hierarquia de componentes (elementos HTML) da interface de formulários é a seguinte:

```
<form>...</form>
  |__ <input>
  |__ <select>...</select>
  |       |__ <option>...</option>
  |__ <textarea>...</textarea>
```

Os elementos <input> são dez, diferenciados pelo seu atributo type, e se apresentam na tela do browser de forma bastante diferente. O elemento <select> representa uma caixa de seleção com uma ou mais opções contidas entre os descritores <option> e </option>. <textarea> representa uma caixa de texto. Maiores detalhes sobre cada componente são apresentados nas seções a seguir.

4.2. Elemento <INPUT>

O elemento <INPUT> é usado para construir botões, caixas de texto, botões de rádio, caixas de checagem e para armazenar variáveis invisíveis. O que distingue um objeto do outro é o seu atributo `TYPE`. A sintaxe geral de <INPUT> é:

```
<input atributos>
```

onde os atributos podem ser diferentes, dependendo do valor do atributo `type`, que pode ter os valores:

- `type=text`
- `type=password`
- `type=radio`
- `type=checkbox`
- `type=submit`
- `type=reset`

- type=hidden
- type=image
- type=file

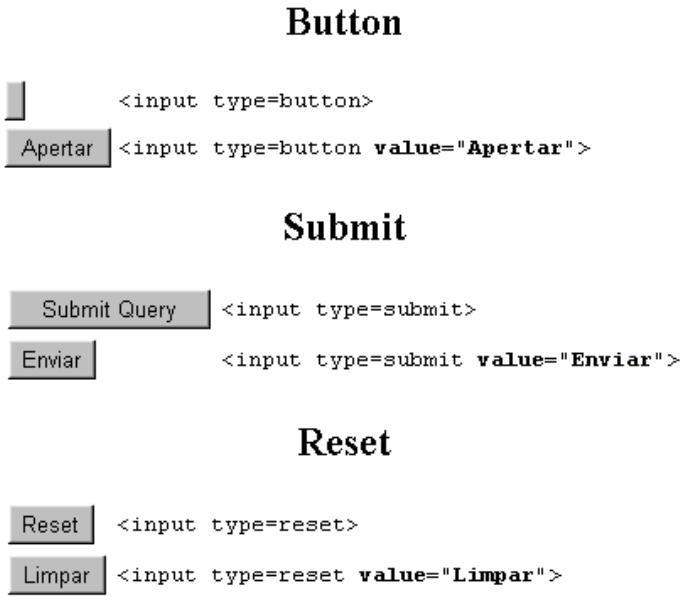
4.3. Botões (TYPE=BUTTON, SUBMIT ou RESET)

Os botões são criados com os atributos type=button, type=reset e type=submit. A sua sintaxe básica é:

```
<INPUT TYPE="button" VALUE="rótulo do botão">    ou
<INPUT TYPE="submit" VALUE="rótulo do botão">    ou
<INPUT TYPE="reset" VALUE="rótulo do botão">
```

Os botões do tipo SUBMIT e RESET reagem a eventos. O botão do tipo BUTTON nada faz. Serve apenas se for programado em JavaScript ou VBScript. Ao apertar o botão SUBMIT, o usuário envia os dados do formulário ao programa no servidor indicado pelo atributo ACTION de <FORM>. Ao apertar RESET, o usuário reinicializa o formulário com seus valores iniciais.

O atributo VALUE permite alterar o texto que aparece dentro do botão. Em botões RESET e SUBMIT, VALUE possui um valor *default*. Em objetos *Button*, o *default* para VALUE é um string vazio, portanto, a menos que este atributo seja definido, o botão aparecerá vazio. A figura ao lado mostra a aparência dos botões em um browser Netscape Navigator rodando em Windows 95 com e sem o atributo VALUE.



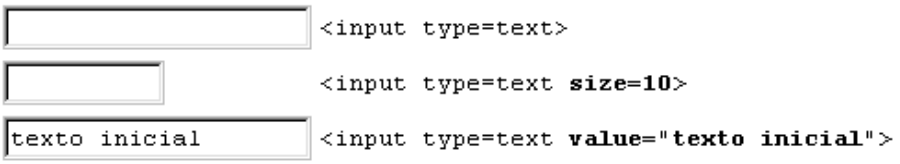
4.4. Campos de texto de uma linha (TYPE=TEXT ou PASSWORD)

Campos de texto de uma única linha são definidos pelos componentes <INPUT> que possuem atributos TYPE com os valores PASSWORD ou TEXT. Eles têm a mesma aparência, mas o texto dos objetos *Password* não é exibido na tela. A sintaxe de um elemento TEXT em HTML é a seguinte:

```
<INPUT TYPE="text"
       NAME="nome_do_campo_de_texto"
       VALUE="texto inicial do campo de textos"
```

```
SIZE="número de caracteres visíveis"
MAXLENGTH="número máximo de caracteres permitido">
```

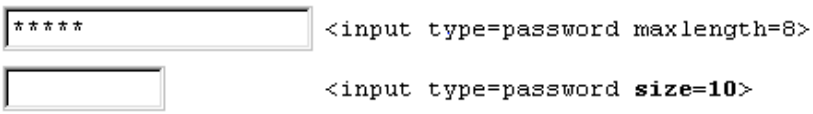
Todos os atributos, exceto o atributo TYPE são opcionais. Se SIZE não for definido, a caixa de texto terá 20 caracteres de largura. Se MAXLENGTH não for definido, não haverá limite para o número de caracteres digitado no campo de textos. A figura abaixo ilustra a aparência de componentes TEXT em um browser Netscape 4.5 rodando em Windows 95.



O elemento do tipo PASSWORD é criado da mesma forma, mas com um atributo TYPE diferente:

```
<INPUT TYPE="password" ... >
```

Os caracteres do texto digitado em componentes PASSWORD não aparecem na tela, como mostrado na figura abaixo (Windows95):



4.5. Campos ocultos (TYPE=HIDDEN)

O componente do tipo HIDDEN é um campo de entrada de dados invisível, que o usuário da página não tem acesso. Serve para que o *programador* passe informações ao servidor, ocultando-as no código HTML da página. É bastante útil na transferência de informações entre formulários distribuídos em mais de uma página. Sua sintaxe é a seguinte:

```
<INPUT TYPE="hidden"
       NAME="nome_do_campo_oculto"
       VALUE="valor armazenado" >
```

Os atributos NAME e VALUE são obrigatórios.

4.6. Botões de rádio (TYPE=RADIO)

O componente do tipo RADIO representa um dispositivo de entrada booleano cuja informação relevante consiste em saber se uma opção foi selecionada ou não. Botões de radio são organizados em grupos de descritores com o mesmo nome (atributo NAME). Cada componente aparece na tela como um botão ou caixa de dois estados: *ligado* ou *desligado*. Dentro de um grupo de componentes (todos com o *mesmo* atributo NAME), somente um deles poderá estar ligado ao mesmo tempo. A sintaxe de um componente RADIO em HTML é a seguinte:

```
<INPUT TYPE="radio"
       VALUE="valor (o valor que será enviado ao servidor)"
       CHECKED     <!-- previamente marcado -->
       > Rótulo do componente
```

A figura abaixo mostra dois grupos de botões de rádio (em um browser Netscape rodando em Windows95). Observe que os atributos NAME distinguem um grupo do outro. O atributo CHECKED indica um botão previamente ligado mas que pode ser desligado pelo usuário ao clicar em outro botão.

GRUPO I

○ pela manhã
○ à tarde
◉ à noite

```
<input type=radio name=turno value="Manhã"> pela manhã
<input type=radio name=turno value="Tarde"> à tarde
<input type=radio name=turno value="Noite" checked> à noite
```

GRUPO II

○ Masculino
◉ Feminino

```
<input type=radio name=sexo value="M"> Masculino
<input type=radio name=sexo value="F"> Feminino
```

4.7. Caixas de checagem (TYPE=CHECKBOX)

Cada elemento <INPUT> do tipo CHECKBOX aparece na tela como um botão ou caixa que pode assumir dois estados: *ligado* ou *desligado*. Diferentemente dos componentes RADIO, vários componentes CHECKBOX de um mesmo grupo podem estar ligados ao mesmo tempo, não havendo, portanto, necessidade de organizar tais objetos em um grupo. A sintaxe de um elemento CHECKBOX em HTML é praticamente idêntica à de RADIO, mudando apenas o valor do atributo TYPE:

```
<INPUT TYPE="checkbox" ...
       VALUE="valor (o valor que será enviado ao servidor)"
       CHECKED     <!-- previamente marcado -->
       > Rótulo do componente
```

A figura abaixo mostra um grupo de caixas de checagem (em um browser Netscape rodando em Windows95. O atributo CHECKED indica um botão previamente ligado mas que pode ser desligado pelo usuário ao clicar em outro botão.

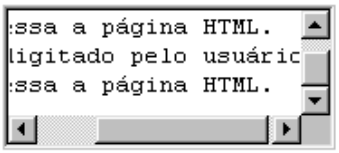
☑ Café
☐ Almoço
☑ Jantar

```
<input type=checkbox name=refeicoes value="Café" checked> Café
<input type=checkbox name=refeicoes value="Almoço"> Almoço
<input type=checkbox name=refeicoes value="Jantar"> Jantar
```

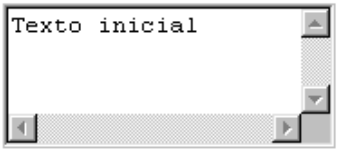
4.8. Elemento <TEXTAREA>

O elemento <TEXTAREA> é um bloco (possui descritor inicial e final) e define uma área onde se pode ler ou digitar texto em várias linhas. A sintaxe para criar um elemento <TEXTAREA> em HTML é a seguinte. Os atributos em negrito são obrigatórios:

```
<TEXTAREA
   ROWS="número de linhas visíveis"
   COLS="número de colunas visíveis"
   NAME="nome_do_campo_de_texto"
   ONBLUR="handlerText"
   ONFOCUS="handlerText"
   ONCHANGE="handlerText"
   ONSELECT="handlerText" >
        Texto inicial
</TEXTAREA>
```




```
<textarea rows=3 cols=20></textarea>
```



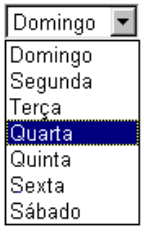

```
<textarea rows=3 cols=20>Texto inicial</textarea>
```

A figura abaixo mostra a aparência de componentes <TEXTAREA>:

4.9. Elemento <SELECT> e <OPTION>

Caixas e listas de seleção como as mostradas nas figuras ao lado são criadas com o elemento HTML <SELECT>. Este componente não contém texto mas consiste de várias opções, representadas pelo elemento <OPTION>, que pode conter texto com opções.

Os componentes <SELECT> podem ter uma aparência e comportamento diferente dependendo se possuem ou não os atributos SIZE e MULTIPLE. A figura ao lado ilustra o efeito desses




```
<select name=umdia>
    <option value="D">Domingo</option>
    <option value="S">Segunda</option>
    <option value="T">Terça</option>
    <option value="Q">Quarta</option>
    <option value="I">Quinta</option>
    <option value="X">Sexta</option>
    <option value="B">Sábado</option>
</select>
```


```
<select name=variosdias size=4 multiple>
    <option value="D">Domingo</option>
    <option value="S">Segunda</option>
    <option value="T">Terça</option>
    <option value="Q">Quarta</option>
    <option value="I">Quinta</option>
    <option value="X">Sexta</option>
    <option value="B">Sábado</option>
</select>
```

atributos, transformando uma caixa de seleção em uma lista que permite seleção de múltiplas opções. A sintaxe de um elemento HTML <SELECT> está mostrada abaixo:

```
<SELECT
   NAME="nome_do_componente"
   SIZE="número de opções visíveis"
   MULTIPLE        <!-- Suporta seleção múltipla -->
 >
      <OPTION ...> Opção 1 </OPTION>
         ...
      <OPTION ...> Opção n </OPTION>
</SELECT>
```

Todos os atributos são opcionais. A existência do atributo NAME é obrigatória em formulários que terão dados enviados ao servidor. Os elementos <OPTION> têm a seguinte sintaxe:

```
<OPTION
   VALUE="Valor da opção"
   SELECTED >
      Texto descrevendo a opção
</OPTION>
```

O atributo VALUE é opcional. Se os dados forem enviados ao servidor, o texto contido entre <OPTION> e </OPTION> é enviado somente se um atributo VALUE não tiver sido definido.

4.10. Outros elementos de formulário

A especificação HTML 4 define outros elementos que podem ser usados em seus formulários. São <BUTTON>, <LABEL>, <FIELDSET> e outros. É possível também aplicar folhas de estilo e mudar radicalmente a aparência dos formulários. Observe que esses recursos não funcionam ainda nos browsers Netscape, portanto, evite usá-los para desenvolver sites com público-alvo que inclua tais visitantes.