UNIVERSIDADE SALVADOR – UNIFACS
PROGRAMA DE PÓS-GRADUAÇÃO EM REDES DE COMPUTADORES
MESTRADO PROFISSIONAL EM REDES DE COMPUTADORES

JOÃO WERTHER FILHO

PPW: UMA TECNOLOGIA WEB DE PÁGINAS DINÂMICAS BASEADA EM PASCAL

Salvador
2006

JOÃO WERTHER FILHO

PPW: UMA TECNOLOGIA WEB DE PÁGINAS DINÂMICAS BASEADA EM PASCAL

Dissertação apresentada ao Programa de Pós-Graduação em Redes de Computadores da Universidade Salvador, como parte das exigências do Curso de Mestrado Profissional em Redes de Computadores, área de concentração em Tecnologias Web e Aplicações Distribuídas, para obtenção do título de “Mestre”.

Orientador: Prof. Dr. Celso Alberto Saibel Santos

Salvador
2006

FICHA CATALOGRÁFICA
(Elaborada pelo Sistema de Bibliotecas da Universidade Salvador - UNIFACS)

Werther Filho, João
PPW: uma tecnologia WEB de páginas dinâmicas baseada em pascal / João Werther Filho. - 2006.
120 f.: il 27 cm.

Dissertação - (mestrado) - Universidade Salvador – UNIFACS, Mestrado em Rede de Computadores, 2006.

Orientador: Profa. Dr. Celso Alberto Saibel Santos.

1. World Wide Web (sistema de recuperação da informação). 2. Redes de computadores. 3. Pascal. I. Santos, Alberto Saibel Santos, orient. II. Título.

CDD: 004

TERMO DE APROVAÇÃO

JOÃO WERTHER FILHO

PPW: UMA TECNOLOGIA WEB DE PÁGINAS DINÂMICAS BASEADA EM PASCAL

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em Redes de Computadores, Universidade Salvador – UNIFACS, pela seguinte banca examinadora:

Celso Alberto Saibel Santos – Orientador_____
Doutor em Informatique Fondamentalle et Parallelisme, Université Paul Sabatier de Toulouse III (UPS)
Universidade Salvador – UNIFACS

Laís Nascimento Salvador_____
Doutora em Engernharia Elétrica. Universidade de São Paulo (USP)
Universidade Salvador – UNIFACS

José Maria Nazar David_____
Doutor em Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro (UFRJ)
Faculdade Rui Barbosa

Salvador, 20 de abril de 2006

Dedico este trabalho aos meus pais João (*in memoriam*) e Flor, à minha esposa e companheira Cida, ao meu pequeno sucessor Diego e a todos que tiveram a paciência em me ensinar alguma coisa e iluminar meu caminho na estrada da educação e sabedoria. Vocês fizeram de mim o que eu sou e lhes agradeço profundamente por isso.

AGRADECIMENTOS

Para realização de um bom trabalho, a caminhada é árdua, mas nunca solitária. Existem muitas ajudas que recebemos de forma direta ou indireta e também auxílios que nem percebemos. Por isso às vezes podemos não ser justos quando agradecemos especificamente às pessoas, pois nem sempre temos consciência de tudo que realmente aconteceu, da forma exata que o universo conspirou para que o trabalho pudesse ser realizado. E podemos cometer algumas injustiças, como, por exemplo, esquecer de alguém. Mas, mesmo assim, vou tentar.

Não poderia começar nenhuma lista de agradecimentos que não fosse com minha mãe, que sempre me apoiou em tudo que fiz durante toda a minha vida. Mãe, Deus lhe abençoe sempre.

Ao meu pai, que deixou para mim o seu legado como um exemplo de competência e seriedade no trabalho, sempre procurarei estar a altura do que ele esperava de mim.

Agradeço também ao meu orientador, Celso Saibel, peça fundamental para conclusão deste trabalho, que acreditou no meu projeto e sempre demonstrou boa vontade e disposição neste percurso, além de ter me dado sugestões valiosas para o projeto.

Agradeço a minha esposa, Cida, que foi bastante compreensiva nas intermináveis horas que precisei dedicar ao trabalho.

Não podia deixar de agradecer a empresa que eu trabalho, a ZCR Informática, pelas longas noites que utilizei sua infra-estrutura para desenvolvimento do projeto.

E, por fim, gostaria de agradecer a todas as pessoas e empresas que disponibilizaram informações relevantes através de web sites, que me deram muito respaldo e informações relevantes para o desenvolvimento da tecnologia. Eles estão relacionados nas referências bibliográficas.

RESUMO

Atualmente, a utilização de páginas dinâmicas no desenvolvimento de aplicações web é, certamente, a solução mais popular do mercado e amplamente difundida entre os desenvolvedores, graças a sua facilidade de edição, implementação e depuração. Porém, devido à interpretação do código dentro de *scriptlets* durante sua execução, elas perdem um pouco em desempenho, já que a página dinâmica é, em geral, interpretada toda vez que é chamada. A utilização de um ambiente CGI (*Common Gateway Interface*), que processa arquivos executáveis, funcionando teoricamente mais rápido que páginas interpretadas, ganharia em desempenho, mas perderia na praticidade de desenvolvimento oferecida pelas páginas dinâmicas. Para conjugar esta praticidade com um bom desempenho, a solução para construção e processamento de aplicações web seria o uso de um ambiente de desenvolvimento utilizando páginas dinâmicas, porém com ambiente de execução CGI. E é exatamente isso que se pretende oferecer através da tecnologia PPW (*Pascal Pages for Web*). A tecnologia PPW conjuga uma ambiente livre de desenvolvimento de aplicações web, utilizando páginas HTML dinâmicas, com um ambiente de execução baseado em processamento CGI *server-side*. Visa disponibilizar a mesma praticidade de desenvolvimento de outras tecnologias baseadas em páginas dinâmicas, como o ASP, PHP ou JSP, associada à rapidez dos executáveis de aplicações CGI. E tudo isso utilizando o Pascal como linguagem padrão.

Palavras-chave: PPW, WWW, Web, Pascal, Object – Pascal, Tecnologia, Plataforma, Páginas, Server – Side, Linguagens, Desenvolvimento, Ambiente.

ABSTRACT

Nowadays, the use of dynamic pages in development of web applications is, certainly, the most popular solution of the market and largely disseminated among the web developers, thanks to your edition, implementation and debug easiness. However, due to the interpretation of the code inside of scriptlets during your execution, they lose some performance, since the dynamic page is, in general, interpreted every time that is called. The use of a CGI (Common Gateway Interface) environment, that it runs executable files, working theoretically faster than interpreted pages, it would win in performance, but it would lose in development easiness offered by dynamic pages. To conjugate this easiness with a good performance, the solution for construction and processing of applications web would be the use of a development environment using dynamic pages, however with CGI execution environment. And it is exactly that what it intend to offer through the technology PPW (Pascal Pages for Web). The technology PPW conjugates a free environment of web applications development, using HTML dynamics pages, with an execution environment based on processing server-side CGI. It seeks to be available the same development easiness of other technologies based on dynamic pages, like ASP, PHP or JSP, associated to the good speed of executable of CGI applications. And all of this, using Pascal as pattern language.

Keywords: PPW, WWW, Web, Pascal, Object – Pascal, Technology, Platform, Pages, Server – Side, Languages, Development, Environment.

.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

1.1. APLICAÇÕES WEB

Em engenharia de software, aplicação web é uma aplicação disponibilizada aos usuários por um servidor web através de uma grande rede como a internet ou até mesmo através uma pequena rede como as redes locais. Nas aplicações web cada usuário (cliente) utiliza uma interface única, que é chamada de browser ou navegador, e o servidor web fornece as páginas HTML que são interpretadas pelo navegador. A habilidade de atualizar e manter aplicações web sem distribuição ou instalação/atualização prévia de software, exceto o navegador, para milhões de potenciais clientes é o grande segredo da sua popularidade. [1]

As aplicações web podem utilizar páginas estáticas ou dinâmicas. Nas páginas estáticas, as páginas HTML já se encontram previamente definidas e com conteúdo estabelecido e fixo. Para alterar seu conteúdo, somente através de edição direta da página. Já as páginas dinâmicas são páginas HTML com conteúdo dinâmico, ou sejam, a mesma página HTML pode ter comportamento e conteúdo distinto a depender da situação. Os conteúdos de página HTML podem ser manipulados dinamicamente através de execução de scripts *client-side* ou utilizando recursos *server-side*. [1] [2]

A utilização de scripts *client-side* altera o conteúdo da página HTML diretamente no computador do cliente (usuário), sem a necessidade de interagir com o servidor web que forneceu a página. Estes códigos são processados pelo navegador, que exibe seus resultados diretamente ao usuário na página HTML em questão. Inconvenientes no seu uso é que dificilmente diferentes navegadores suportam os mesmos aspectos da linguagem (quando suportam), ficando difícil escrever um único código para todos os navegadores comerciais. Outro problema é que nem sempre tudo pode ser resolvido na máquina do usuário, sendo necessário buscar informações adicionais no servidor web. [2]

O funcionamento de um *server-side* já é um pouco mais complicado. O navegador envia uma requisição da página, com os dados que deseja. O servidor web processa o código embutido na página requisitada e como saída do processamento é gerada a página HTML com as informações desejadas. O servidor web envia esta página gerada ao cliente requisitante. Existem muitas possibilidades e vantagens na sua utilização, mas o uso exagerado e sem critérios podem causar problemas de tráfego na rede, bem como a sobrecarga do servidor. [2]

Como grandes exemplos comerciais de tecnologias *server-side*, podemos citar ASP e JSP, as quais abordaremos na seção 2.5, além de PHP e ColdFusion, dentre outras. Como principais exemplos de scripts *client-side* podemos citar o javaScript e o VBScritp.

1.2. OBJETIVO

O objetivo deste trabalho é disponibilizar uma tecnologia para desenvolvimento de aplicações web com páginas HTML dinâmicas, baseado na linguagem Pascal [*], com desempenho e recursos comparáveis aos encontrados em tecnologias similares, tais como o ASP e o JSP.

Para desenvolver nesta tecnologia, o programador constrói suas páginas em qualquer ferramenta de página dinâmica da sua preferência e coloca código Pascal incluso nas páginas entre as tags HTML, utilizando *scriptlets* como delimitadores do código. A idéia é que o PPW forneça a mesma praticidade de desenvolvimento que possuem as aplicações que utilizam páginas dinâmicas, associada à mesma rapidez de execução das aplicações CGI. E tudo isso baseado em uma linguagem que normalmente é o primeiro contato do aluno de um curso de ciências da computação ou semelhante, que é o Pascal [3]. Esta tecnologia foi batizada de PPW, que significa *Pascal Pages for Web*.

Antes de continuar, deve-se ressaltar que este trabalho não pretende desenvolver uma tecnologia, para competir comercialmente com as já existentes e consagradas, como ASP, PHP, JSP; e sim mostrar uma alternativa viável baseada em código Pascal. É prudente que, para pensar em competir, teria que ter uma maior pesquisa e aperfeiçoamento do produto, visando sua otimização, confiabilidade e extensão de funcionalidades. Mas nada impede de compará-la com estas tecnologias em alguns

[*] Apesar da linguagem base da tecnologia ser o Object Pascal, o autor entende que o Object Pascal é uma extensão do Pascal e, como conseqüência, baseado em Pascal.

quesitos, principalmente no desempenho, com foco maior no tempo de resposta da aplicação.

Em relação à eficiência foi realizada uma análise comparativa de desempenho do PPW com o JSP e o ASP, ao final do estudo. Entretanto, este estudo restringiu-se ao tempo de resposta e o atendimento de requisições sob stress, o qual foi considerado muito significativo para medida de desempenho. Claro que para uma análise de eficiência mais precisa dever-se-ia medir a utilização de recursos, como CPU, memória e disco, e ainda analisar taxas de processamento.

O trabalho também não pretende desenvolver nenhum compilador para a linguagem Pascal ou *Object Pascal*. A transformação do código Pascal embutido nas páginas em código executável, como será mostrado mais adiante, é realizada através do uso de um compilador Pascal já existente. A tecnologia PPW apenas gerencia o processo de transformação das páginas PPW até fontes pascal e daí até uma aplicação final.

O Pascal adotado para o projeto PPW não foi o Pascal ANSI/ISO. A linguagem adotada foi o *Object Pascal*, que é basicamente a linguagem Pascal com extensão de orientação a objeto. Para compilador foi selecionado o compilador da **Borland Corp**., baseado no *Borland Delphi 7*. Este compilador, além de ser compatível com o Pascal ANSI/ISO [4], e possuir de uma boa aceitação no mercado, oferece diversas funcionalidades e classes utilitárias que auxiliam neste trabalho, e como o autor é bastante familiarizado com ele, é possível desenvolver o projeto com maior agilidade. Além disso, suas funcionalidades e classes são chamadas de *Cross-*

Platforms, ou seja, podem ser compiladas e executadas tanto em ambiente Windows como Linux [5]. Apesar dessas facilidades, foi decidido que serão utilizadas apenas poucas classes oriundas do *Delphi 7*, visando uma maior compatibilidade futura. A maioria das funções e classes necessárias à tecnologia foi construída. Basicamente, do *Delphi 7*, só foi utilizado:

- *SysUtils*: a *unit* de utilitários gerais;
- *SQLExpr*: a *unit* de utilitários de acesso à banco de dados, que dá suporte ao *dbExpress* do *Delphi 7* (componente de interface com banco de dados);
    - Todo acesso a banco de dados da tecnologia PPW é feita pelo *dbExpress*;
- *IdTCPServer* e a *IdTCPClient*: as *units* que simulam servidores e clientes TCP.

O trabalho também não pretende desenvolver um servidor HTTP. Foi utilizado o servidor HTTP Apache 2.0.48 da **Apache Software Foundation** como servidor HTTP de páginas HTML acoplado ao servidor de aplicações PPW.

1.3. MOTIVAÇÃO

E por que o Pascal? Primeiro, porque é uma linguagem que foi projetada como uma linguagem de ensino, possuindo também uma notável combinação de simplicidade e expressividade [6]. Segundo, porque é uma das linguagens mais adotadas nos primeiros semestres dos cursos de Ciências da Computação e similares [3]. Neste período, normalmente, é a linguagem mais familiar ao aluno e ele tende a dominá-la relativamente bem.

Em semestres mais avançados quando o aluno é introduzido ao ambiente web, o Pascal, que foi muito importante no seu aprendizado de programação, além de ter servido como base de aprendizado de outras disciplinas no curso até então, é normalmente posta de lado. Infelizmente, não existe uma tecnologia suficientemente difundida em meios acadêmicos ou no mercado para a programação web baseada em Pascal. Como conseqüência, a familiaridade e os conhecimentos adquiridos nos primeiros anos de formação são perdidos na migração natural dos alunos para o ambiente web. Então, ao aluno é apresentado um novo ambiente de programação acoplado com uma nova linguagem que possa interagir com o ambiente web. E para isso, ele tem que aprender esta nova linguagem de programação, que pode ser o java/JSP, ASP e/ou outras similares. Neste contexto, cabem ao aluno dois aprendizados simultâneos: do ambiente web e da nova linguagem de programação.

Porém, se fosse possível ao aluno continuar com a linguagem Pascal e apenas tivesse a mudança de ambiente para web, seu aprendizado seria então totalmente direcionado para o ambiente web. Como não estaria dividindo esforços no aprendizado, já que a linguagem continuaria sendo o Pascal, seu aprendizado no ambiente web poderia ser mais rápido. Depois sim, ele poderia partir vantajosamente para aprendizado de outras linguagens. Isso poderia contribuir para um maior aproveitamento do aluno.

No PPW, a linguagem utilizada é baseada no Pascal, possuindo extensões de Orientação a Objeto (O.O.) – *Object Pascal*. A opção por O.O. se deve ao fato de que tal metodologia incorpora um conceito de herança que “aumenta muito a reutilização potencial dos softwares desenvolvidos, oferecendo, portanto, a

possibilidade de significativos aumentos na produtividade de desenvolvimento de software” [6].

Como o PPW oferece muitos recursos que ASP, PHP e JSP oferecem,. se ele fosse adotado como a primeira tecnologia web a ser vista academicamente, a necessidade de aprendizado do aluno seria basicamente apenas do conceito das páginas dinâmicas HTML, pois o aluno já teria bons conhecimentos em Pascal. Conseqüentemente, seu aprendizado no PPW tenderia a ser mais rápido do que no ASP, PHP ou JSP. Esta é uma das principais motivações que levaram à sua criação e desenvolvimento.

Uma outra motivação para a criação da tecnologia PPW é a situação do *Delphi*, que, apesar de disponibilizar um ambiente integrado de desenvolvimento para predominantemente aplicações cliente/servidor e recentemente para plataforma .NET, sua boa popularidade ainda no Brasil [7], não reflete exatamente o que passa no mercado americano, que mostra um declínio desta ferramenta [8] [9].

Pode ser que a recente adoção exclusiva da plataforma .NET, o que teoricamente vai contra os programadores de software livre, somado ao crescimento cada vez maior da utilização de plataformas web de desenvolvimento [8], tenham contribuído para o declínio apontado, o que vai levar a desenvolvedores *Delphi* a procurarem alternativas no mercado.. Como o PPW é fortemente baseado em *Object Pascal*, é também uma tecnologia web *server-side* e, neste momento inicial, utiliza o compilador do *Delphi 7*, ele poderia se tornar bastante atraente, bem como uma boa alternativa, aos desenvolvedores *Delphi*.

1.4. VISÃO GERAL E PRINCIPAIS CONTRIBUIÇÕES

A tecnologia PPW é composta por um ambiente de desenvolvimento, baseado em páginas PPW, incluindo distribuição (*deployment*) das suas aplicações, e por um ambiente de execução *server-side* destas páginas.

As páginas PPW, assim como as páginas HTML, ASP, PHP ou JSP, possuem o formato texto e podem ser criadas e editadas em qualquer editor de textos ou de páginas HTML. Elas possuem basicamente a apresentação e formato como o exemplo mostrado na Figura 1.

```
<html>
<head>    <meta http-equiv="Content-Type" content="text/html">
<title>Teste Hello Word</title>
</head>
<body text="#000000" bgcolor="#FFFFFF" link="#0000FF" vlink="#000080" alink="#FF0000">
<font color="#000033" face="Arial, Helvetica, sans-serif">
<%var i: integer;%>
<%  for i:=1 to 10 do
    begin
%>
        HELLO WORLD!! <br>
<%
    end; %>
</font>
</body>
</html>
```

Figura 1. Exemplo básico de página PPW

Nota-se o uso de *scriptlets* delimitando os códigos em Pascal, tal qual acontece em tecnologias semelhantes como o ASP, PHP ou JSP.

Para cada aplicação PPW existe um único XML de configuração, chamado de apw.xml, que contém o nome da aplicação, bem como referências a banco de dados que podem ser conectados à aplicação, se necessário. Ele possui a seguinte estrutura básica, como mostra a Figura 2.

```
<appname>cadastro</appname>
<dbconn name="sgci">
    <dbdriver>MSSQL</dbdriver>
    <HostName>GER-WERTHER\WERTHER</HostName>
    <Database>SGCI</Database>
    <User_Name>sa</User_Name>
    <Password></Password>
</dbconn>
<dbconn name="sead">
    <dbdriver>MySQL</dbdriver>
    <HostName>localhost</HostName>
    <Database>sead_legislacao</Database>
    <User_Name>root</User_Name>
    <Password></Password>
</dbconn>
```

Figura 2. Exemplo do arquivo APW.XML

A tecnologia PPW permite ao desenvolvedor:

- Utilizar páginas dinâmicas PPW e páginas estáticas HTML;
- Utilizar qualquer ambiente de desenvolvimento para páginas HTML ou dinâmicas, como *Dreamweaver*, *HomeSite* e outros;
- Receber dados e variáveis através de POST e GET;
- Controlar, validar e expirar sessões HTTP;
- Utilizar variáveis de sessão;
- Acessar diretamente bancos de dados *MySQL, MS-SQL Server, Oracle, IBM DB2, Interbase, Informix*;
- Utilizar conexões inteligentes de banco de dados, estabelecendo apenas uma única vez durante todo o processamento da sessão;
- Definir e remover *cookies*;
- Utilizar *units* Pascal referenciadas pelas páginas PPW, facilitando a separação entre interface e código – O código não precisa estar todo na página;
- Obter alto desempenho das aplicações, devido ao processamento CGI.

Como se pode observar, a tecnologia PPW oferece uma série de recursos encontrados em uma tecnologia *server-side* para páginas dinâmicas existentes no mercado, além de um ambiente livre de desenvolvimento.

1.5. ORGANIZAÇÃO

Esta dissertação está organizada em cinco capítulos incluindo este capítulo introdutório. O capítulo seguinte aborda o referencial teórico com conceitos fundamentais para o desenvolvimento deste trabalho. O capítulo 3 descreve a estrutura e o funcionamento do projeto e tecnologia PPW. O capítulo 4 mostra comparativos da tecnologia com outras tecnologias *server-side* comerciais através de testes de desempenhos realizados e faz uma breve comparação com tecnologias web de características semelhantes, que também são baseadas em Pascal. E, por fim, o capítulo 5 contém as discussões finais sobre o trabalho seguido das referências bibliográficas.

2. REFERENCIAL TEÓRICO

2.1. PASCAL

A linguagem de programação Pascal foi criada em 1971, quando o Prof. Niklaus Wirth projetou uma linguagem como trabalho escolar para estudantes de programação da *Technical University* de Zurique, Suíça. O nome Pascal foi uma homenagem ao filósofo e matemático do século XVII, Blaise Pascal, inventor de uma das primeiras máquinas lógicas. [10] [11]

O Pascal foi baseado em algumas linguagens estruturadas existentes na época, como ALGOL e PL/1, tentando facilitar ao máximo o seu aprendizado. Pascal é bastante orientado a dados, dando ao programador a capacidade de definir tipos de dados personalizados. Com esta liberdade veio a verificação de tipos, que garantiu que tipos não se misturariam. Pascal pretendia ser uma linguagem educacional, e foi amplamente adotada como tal. Pascal é de escrita mais livre, diferente de FORTRAN, por exemplo, assim os estudantes não tinham que se preocupar com formatação. Além disso, Pascal se parece muito com uma linguagem natural, tornando muito fácil o entendimento do código escrito com ele. [11]

O Pascal somente ganhou popularidade quando foi adotado pela Universidade da Califórnia, San Diego (UCSD), em 1973. Lá surgiu o *UCSD P-System* um sistema

operacional baseado em Pascal e a linguagem *UCSD Pascal*. Ambos se tornaram popular e foram amplamente usados em universidades. [11]

Contudo somente ao final do ano de 1983, é que a empresa americana **Borland Software Corp.** lançou o *TurboPascal* para microcomputadores da recém chegada arquitetura IBM-PC. Uma novidade na época, o *TurboPascal* reunia em um único ambiente, editor de texto para os programas, compilador Pascal e um link-editor para gerar executáveis. Daí o *TurboPascal* evoluiu, acrescentando novas funcionalidades, introduzindo rotinas compatíveis para acesso a arquivos e diretórios no MS-DOS e agregou ambiente de depuração na versão 5.0. [10] [11]

2.2. OBJECT PASCAL

O *Object Pascal* foi criado por Niklaus Wirth, com o auxílio de Larry Tesler no ano de 1985, na **Apple Computer**. Ele surgiu da adição de estruturas para suportar extensões de orientação a objetos à linguagem Pascal. Foi criado basicamente para suportar o *MacApp*, um framework de aplicações da Macintosh. [12]

Em 1989, na versão 5.5 do *TurboPascal* foi incrementada também uma estrutura de linguagem orientada por objeto, dando origem assim a primeira implementação comercial do *Object Pascal*. A partir daí a **Borland Software Corp.** mudou seu foco para Windows, abandonando o DOS, e investiu no grande sucessor do *TurboPascal*, o Delphi, também baseado no *Object Pascal*. [12] [11]

Atualmente existem vários compiladores para *Object Pascal*, como o *Free Pascal* [13] e o *GNU Pascal* [28]. Existem publicações sobre padrões do *Object Pascal* elaborados por *Technical Committee X3J9, Programming Language Pascal*, um sub-comitê do *Accredited Standards Committee, X3*m atual INCTIS (Comitê Internacional para Padrões de Tecnologia da Informação) [14]. [10]

2.3. DELPHI

Delphi é um ambiente integrado de desenvolvimento de software produzido pela **Borland Software Corp.** baseado no *Object Pascal* e inicialmente voltada para plataforma Windows. Quando lançado em 1995 para a plataforma Windows 16 bits, foi o primeiro a ser descrito como ambiente RAD (Desenvolvimento Rápido de Aplicações). A linguagem utilizada pelo *Delphi*, o *Object Pascal*, a partir da versão 2005 passou a ser denominada de *Delphi Language*. O *Delphi* originalmente direcionado apenas para a plataforma Windows, começou a desenvolver aplicações nativas para Linux através do Kylix, uma espécie de variante de versões do *Delphi* para o Ambiente Linux. Notadamente a versão 7 (utilizada neste trabalho) possui uma variedade de bibliotecas que processam tanto na plataforma Windows como Linux e é totalmente compatível com o Kylix. Mas, o *Delphi*, em suas versões mais recentes, está voltado exclusivamente para a plataforma Microsoft .NET. [12]

2.4. APLICAÇÕES CGI

CGI (*Common Gateway Interface*) é uma importante tecnologia *server-side* da web que permite que um cliente (navegador) requisite informações provenientes de um

programa executável dentro do servidor web. CGI especifica padrões para o envio de requisições entre o servidor web e o executável utilizado. [15]

A linguagem Perl é frequentemente associada com CGI, mas uma das características do CGI é que ela independe da linguagem. O servidor web não precisa saber nada sobre qualquer linguagem em questão. De fato, os programas CGI podem ser escritos em qualquer linguagem, contanto que possa ser executada diretamente pelo sistema. Além de Perl, exemplos de utilização de CGI podem incluir C/C++, Phyton, Pascal, *Object Pascal* e *Visual Basic*. [15] [16]

Existe definição prévia das áreas onde estarão os programas CGI dentro de um servidor web. Quando uma requisição para uma determinada URL pertencente à área CGI do servidor é recebida, o programa correspondente é chamado passando qualquer dado que o cliente enviou como entrada e o programa então é executado. Como saída ele fornece uma página HTML que é enviada de volta para o cliente, através do servidor web. [15]

O CGI era amplamente utilizado até a explosão da Web comercial. Neste novo contexto, as aplicações CGI podem causar um aumento linear de carga (processos ou *threads* do SO na memória) e trazer conseqüente impacto na performance do servidor, podendo tornar-se lento e, a depender do mau uso, inviável de se utilizar profissionalmente. [17] [16]

Outro ponto interessante é que os programas CGI terminavam sua execução tão logo encerrassem seu processamento, ou seja, não guardavam qualquer tipo de

estado. Quando um programa CGI executa uma tarefa que consome tempo como abrir uma conexão de rede, estabelecer comunicação com o banco de dados, autenticar-se, efetuar uma consulta e externar o resultado, logo após ele fecha todos os arquivos, encerra as conexões de banco, limpa páginas de códigos em memória, e encerra seu processamento. Casos, após alguns instantes, houver a mesma requisição, existirá exatamente o mesmo ônus de abrir a conexão, estabelecê-la e tudo o mais que for necessário. Não existe nenhum tipo de cache, otimização ou inteligência por trás do sistema, pelo menos não de uma forma automática. [17]

2.5. PÁGINAS DINÂMICAS

Foi visto antes que as páginas dinâmicas são páginas HTML com conteúdo dinâmico, cujo comportamento e conteúdo distinto podem variar, a depender da situação. [2]

Será abordado a seguir duas das principais tecnologias de páginas dinâmicas. As duas possuem características *server-side*, cuja explicação simplista também foi mostrada anteriormente. São elas, o ASP e o JSP.

2.5.1. ASP

Microsoft Active Server Pages (ASP) é uma tecnologia de codificação *server-side* utilizada para criar aplicações web dinâmicas e interativas, desenvolvida pela **Microsoft Corp.**. Funciona como um ambiente de execução server-side do Microsoft

Internet Information Server (IIS), servidor web, que permite a utilização de códigos ActiveX nas suas páginas e componentes de servidor ActiveX, no servidor web. [18]

Uma página ASP é um página HTML que contém códigos *server-side* que são processados pelo servidor web antes de serem enviados ao navegador do usuário requisitante. Quando um navegador envia uma requisição para um arquivo .asp (página ASP) do servidor web, a tecnologia ASP é então chamado pelo servidor web, o qual processa o arquivo requisitado, do início ao fim, e executa qualquer código ASP existente na página. [18]

Segundo a **Microsoft Corp.**, é possível a utilização do ASP utilizando componentes COM (*Component Object Model*) e XML (*Extensible Markup Language*). A utilização do COM aumenta a capacidade de promover um compacto, reutilizável e seguro meio distribuído de acesso à informação desejada, estando ela disponível onde estiver. O XML é uma meta-linguagem de marcação que fornece formatos para descrever, armazenar e processar dados estruturados através de um conjunto de tags. Facilita muito a comunicação com outros sistemas e base de dados. [18]

2.5.2. JSP

O JavaServer Pages (JSP), desenvolvida pela **Sun Microsystems**, é uma tecnologia baseada em Java que provê um simplificado e rápido processo de se criar páginas web que exibem conteúdos gerados dinamicamente. Uma página JSP é uma página que inclui a tecnologia JSP e tags personalizados em combinação com outros tags estáticos (HTML ou XML). Uma página JSP possui a extensão .jsp ou

.jspx; esta extensão sinaliza ao servidor web que a máquina JSP (*JSP engine*) irá processar os elementos da página. Usando o descritor de distribuição, o web.xml, extensões adicionais pode ser associadas ao *JSP engine*. [19]

Páginas JSP possuem códigos encapsulados escritos na linguagem Java que são os responsáveis por gerar o conteúdo da página. Com JSP, os designers da web e programadores podem rapidamente incorporar elementos dinâmicos em páginas da web, utilizando código Java. [19] [20]

Quando uma página JSP é solicitada por um usuário, o servidor compila a página, de forma semelhante que compila um programa java, executa o código gerado (*bytecode*), utilizando-se para isso da JVM (*Java Virtual Machine*) do servidor de aplicação e transmite o resultado (página HTML gerada) para o usuário. O processo de compilação da página é feito somente na primeira requisição à página, ou então em caso de alteração da mesma. [19]

JSP oferece diversos benefícios como um sistema para geração de conteúdo dinâmico. Primeiramente, sendo uma tecnologia baseada na linguagem Java, ela se aproveita de todas as suas características e vantagens, como orientação a objetos, herança, encapsulamento, tratamento de exceções e gerenciamento de memória (*garbage colection*). [19] [20]

Pelo fato de haver uma API padrão publicada para JSP, e pelo fato do código Java compilado (*bytecode*) ser portável através de todas as plataformas que aceitam uma JVM, o uso de JSP não restringe ao uso de uma plataforma de hardware ou sistema

operacional. Se for necessário uma mudança do sistema operacional ou hardware em relação ao inicialmente desenvolvido e/ou utilizado, todas as páginas JSP e classes de Java associadas podem ser migradas no estado em que se encontram. Ou seja, o JSP também segue a filosofia java: "*write once, run anywhere*" (escreva apenas uma vez e processe em qualquer lugar). [19] [20]

2.6. TECNOLOGIAS BASEADAS EM PASCAL PARA AMBIENTE WEB.

Em pesquisa feita pela internet foram encontradas duas tecnologias de características aparentemente semelhantes ao PPW, inclusive baseadas em Pascal. Apesar de semelhantes, deve ficar claro que o desenvolvimento da tecnologia PPW apresentada neste artigo não foi baseado em nenhuma destas tecnologias.

2.6.1. Pascal Server Pages (PSP).

Pascal Server Pages é um projeto da *Nemesis Pascal* [21] que permite também a inclusão de códigos Pascal em páginas HTML através do uso de *scriptlets*.

O PSP utiliza o compilador do projeto JEDI, baseado em *Delphi*. Sendo assim, possui características muito semelhantes ao *Delphi*, inclusive no seu acesso a bancos, quando utiliza o BDE e/ou o *dbExpress*. Para acesso a banco de dados também utiliza outros suporte de acesso proveniente de outros projetos *open source*, como *Zeos-lib* e *MyDB*. [21]

Com o funcionamento também baseado em CGI, o *Pascal Server Pages* possui três tipos básicos de documentos, termo utilizado para sua página PSP [21]. Para o entendimento destes tipos de documentos convém apresentar duas definições utilizadas:

- Avaliações: São blocos de códigos em Pascal (*begin/end*) que devolvem um valor de retorno na variável reservada *Result*. Este valor de retorno é automaticamente exibido na tela.
- Expressões: São códigos em pascal que não possuem blocos (*begin/end*) nem utilizam a variável de retorno *Result*.

Sendo assim, eis os tipos básicos de documentos do PSP:

1. Documento Monoprograma: Somente contém uma expressão pascal. Exemplo:

```
<% sqrt(24) %>
```

2. Documento Simples: Código HTML com uma expressão pascal. Exemplo:

```
<html><body>
<% sqrt(24) %>
</body></html>
```

3. Documento Complexo: Código HTML com uma expressão pascal e uma avaliação pascal. Exemplo:

```
<html><body>
<p>A raíz quadrada de 24 é <% sqrt(24) %></p><hr>
<%
begin
    if DateTimeToStr(Now) = '01/01/2001 08:00:00' then begin
      Result := 'Data corrente confere';
```

```
    else
      Result := 'Data corrente não confere';
    end;
end;
%>
</body></html>
```

2.6.2. IPSCRIPT

IPSCRIPT é um compilador Pascal para páginas HTML [22]. Os códigos em pascal são inseridos entre *scriptlets* dentro da página, que tem a extensão .ips.

Quando um usuário, através de um navegador qualquer, chama uma destas páginas, o servidor HTTP chama o compilador fornecendo a página como parâmetro. O compilador verifica se a compilação da página é necessária (i.e., se a página foi alterada desde a sua última compilação). [22]

O novo código gerado ou o antigo, caso não precise de uma nova compilação, é executado no servidor, através de um módulo de execução do IPSCRIPT.

O IPSCRIPT é muito semelhante à linguagem Pascal[22], exceto por algumas mudanças, como:

- Só existe um único tipo de variável, o tipo *Variant* (como no *Delphi*);
- O IPSCRIPT converte variáveis e funções para adaptar tipos de operação;
- Pode ser usado procedures, funções, variáveis locais e globais, etc.

O IPSCRIPT oferece facilidades ao programador[22], como:

- Gravar/recuperar variáveis de ambiente de servidor;
- Gravar/recuperar parâmetros de páginas e formulários HTML;

- Sessões HTTP;
- Gravar/recuperar variáveis de aplicação de uma sessão
- Gravar/recuperar *cookies* do navegador;
- *IPReport*, uma gerador de relatório para documentos PDF *on line* via IPSCRIPT.

Seguem alguns exemplos de IPSCRIPT: [22]

Exemplo 1:

```
<html><head><title>IPScript example 1</title></head><body>
This is ipscript sample code :<br>
<%
Script;
Var I;
Begin
For I: = 1 to 100 do Echo('Hello #'+I+'<br>');
End;
%>
</body></html>
```

Este exemplo anterior exibe 100 linhas de “Hello” seguido pelo número da linha.

Exemplo 2:

Este exemplo mostra cálculo de fatorial através da recursividade.

```
<html><head><title>Example n°2</title></head><body>
<form action="ex2.ips" method="post">
Input a number between 1 and 20 :
<input type="text" name="nombre">
<input type="submit" name="calculer" value="calculer">
</form>
<%
Function Fact(N);
Begin
If N = 1 Then Result := 1
Else Result := N * Fact(N-1);
End;
Script;
```

```
Var I;
Begin
If InParms('Nombre') Then
If Parms('Nombre') > 0 and Parms('Nombre') <= 20 Then
Begin
For I := 1 to Parms('Nombre') do
Echo('The result for '+I+' is '+Fact(I)+'<br>');
End
Else Echo(Parms('Nombre')+' is not between 1 an 20 !');
End; { Script }
%>
</body></html>
```

Exemplo 3:

Este é um exemplo de uso de banco de dados.

```
<html><head><title>Example n°3</title></head><body>
<h1>C:\MATABLE.DAT content</h1>
<%
Script;
Var Q;
Begin
SqlAssignDB(Q, 'c:\'); // assign query object and log to database folder
c:\
SqlQuery(Q, 'select FirstName, LastName, Date from matable');
If SqlExec(Q) Then // execute the query
  If SqlFirst(Q) Then // if there is a record in the dataset ...
    Begin
    Echo('<table border="3">');
    Repeat
      Echo('  <tr>');
      Echo('    <td>'+SqlField(Q,'FirstName')+'</td>');
      Echo('    <td>'+SqlField(Q,'LastName')+'</td>');
      Echo('    <td>'+DateToStr(SqlField(Q,'Date'))+'</td>');
      Echo('  </tr>');
    Until not SqlNext(Q);
    Echo('</table>');
    End
  Else Echo('The table is empty.')
Else Echo('Can''t execute the query: '+SqlError);
SqlFree(Q); // release query object's memory
End;
%>
</body></html>
```

O IPSCRIPT teve sua primeira versão em 18/11/1999 [22], onde possuía um funcionamento muito limitado, sendo apenas um compilador Pascal. Tratamentos

para ambiente web (sessões, parâmetros de páginas) só existiram a partir da versão 2. Hoje está na versão 3.35 (datada de 17/01/2005) [22].

3. O PROJETO PPW

O projeto PPW nasceu de uma idéia em abril/2004, com o intuito do desafio em desenvolver uma tecnologia para desenvolvimento web baseado em Pascal, contendo recursos encontrados nas tecnologias *server-side* mais populares do mercado, porém com um desafio de oferecer um desempenho equivalente às mesmas e tornar o aprendizado do ambiente web mais fácil aos alunos dos cursos de Ciências da Computação e similares. A idéia nasceu junto com a possibilidade de difundi-la academicamente, no intuito de ajudar no aprendizado, conforme exposto na seção 1.3.

O PPW é uma tecnologia com instruções simples de configuração e de fácil utilização para programadores que estejam habituados ao Pascal, como será mostrado a seguir. Possui muitas características semelhantes ao ASP, PHP e JSP, podendo ser uma alternativa a estas tecnologias no ponto de vista técnico, como será demonstrado pelos estudos de caso realizados.

3.1. ESTRUTURA

A tecnologia PPW compõe-se por um ambiente de desenvolvimento e distribuição das suas aplicações, baseado em páginas HTML dinâmicas; e outro ambiente que cuida exclusivamente da tecnologia necessária à sua execução, baseado em processamento *server-side* CGI. Ela contém dois aplicativos somados a um conjunto

de classes e componentes. Os dois aplicativos, de vital importância à tecnologia, são o *deployPPW* e o *PPWController*. Ambos são programas executáveis e foram desenvolvidos em *Object Pascal* com o *Delphi 7* utilizando apenas as classes já citadas na seção 1.2.

O *deployPPW* é o programa distribuidor de aplicações PPW. É ele que tem a função de compilar, preparar e distribuir a aplicação PPW, disponibilizando-a para execução. Ele converte as páginas PPW em executáveis e leva toda a estrutura de pastas da aplicação ao servidor PPW, em paralelo criando uma URL para a aplicação PPW. Mais detalhes sobre o funcionamento do *deployPPW* na seção 3.2.1, quando é explicado o desenvolvimento e distribuição de aplicações PPW.

O *PPWController* trabalha do lado de servidor (*server-side*), sendo portanto componente fundamental da tecnologia PPW. É responsável pelo gerenciamento de sessões, variáveis de sessão e conexões a bancos de dados. Está sempre ativo e residente na memória ouvindo a porta 81 do servidor. Ele fica respondendo, em modo *multi-thread*, através de conexões TCP, à solicitações dos programas PPW (executáveis CGI) quando são chamados pela aplicação. Ele é o responsável pelo controle de sessões, podendo incluir, remover ou alterar informações e variáveis de sessão. É também o responsável pelo estabelecimento e manutenção de conexões aos bancos de dados e a disponibilização e distribuição delas para a aplicação. Funciona como uma espécie de repositório de sessões e controlador da aplicação PPW, no coração da arquitetura *server-side*. A Figura 3 ilustra o funcionamento do *PPWController* junto ao servidor PPW, visando um melhor entendimento. Mais detalhes sobre o funcionamento do *PPWController* na seção 3.2.2.

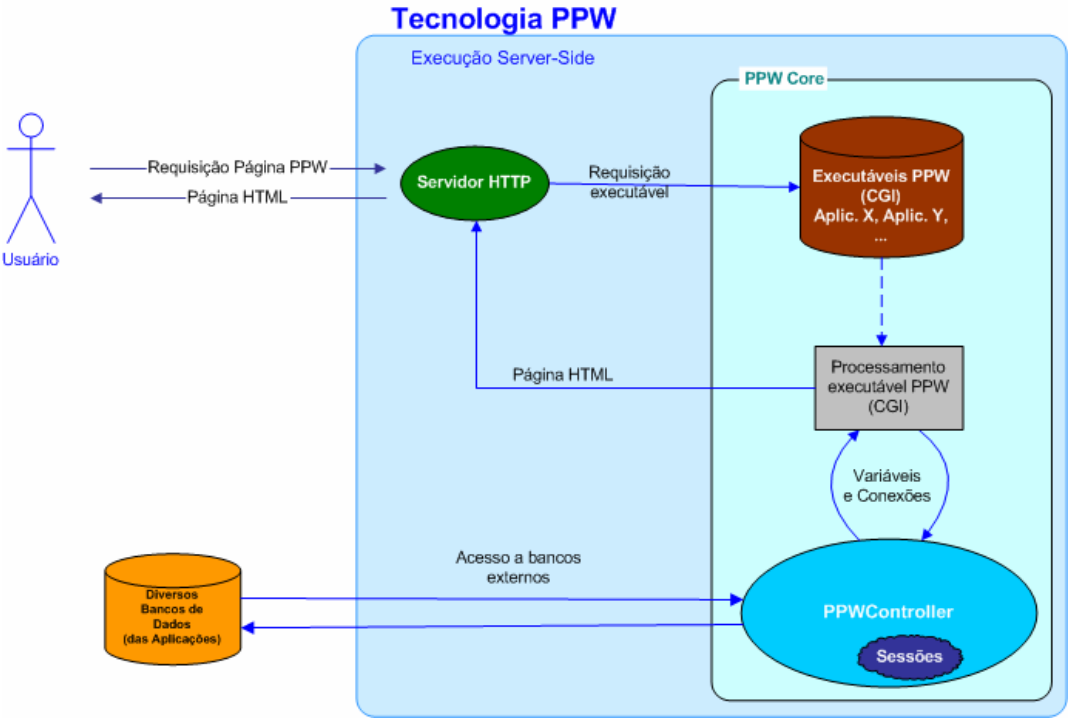

Figura 3. Arquitetura do servidor PPW

Existem componentes e classes que suportam estes dois aplicativos, além de oferecer funcionalidades extras à tecnologia PPW. Muitas dessas classes e componentes são totalmente transparentes ao programador.

Tabela 1. Classes utilizadas pelo PPW

Nome da Classe	Descrição
TRequestParameter	Trata parâmetros recebidos por POST ou GET
TClientSession	Traduz sessões estabelecidas da aplicação com máquinas clientes
TSessionVariable:	Trata variáveis de sessão
TConnectionVariable:	Corresponde a variáveis de conexão a banco, vinculadas à sessões de cliente
TCookie	Manipula os *cookies* de uma aplicação
TXMLFile	Trata de arquivos XML

TElementsList	Representa elementos (tags) de um arquivo XML
TAttributesList	Representa os atributos de elementos (tags) de um arquivo XML
TStringsList	Representa lista encadeada/dinâmica de strings
SqlExpr	*Unit* Pascal que oferece um conjunto de classes que implementam o *dbExpress* para acesso a banco de dados, disponibilizada pelo *Delphi 7*
IdTCPServer e IdTCPClient	*Units* Pascal que oferecem um conjunto de classes que implementam respectivamente um servidor TCP *multi-thread* e um cliente TCP, ambas disponibilizada pelo *Delphi 7*
TControllerServer	Classe herdeira de *TIdTCPServer*, especializando o tratamento do servidor TCP para o objeto *Servidor*, instância desta classe, que é o coração do *PPWController*

Mais informações sobre as classes desenvolvidas diretamente para a tecnologia será visto no APÊNDICE A .

3.2. FUNCIONAMENTO

Os tópicos a seguir detalham o funcionamento da tecnologia, desde a sua distribuição até o seu processamento no servidor de aplicações PPW.

Apresentam também as principais regras de desenvolvimento para o programador e como deve ser o desenvolvimento estrutural de uma aplicação PPW.

3.2.1. Desenvolvimento e Distribuição

3.2.1.1. Desenvolvimento de Aplicações PPW

O desenvolvimento de aplicações PPW tem suas peculiaridades. Serão apresentadas a seguir algumas regras necessárias ao seu bom desenvolvimento.

- Todo comando, ou seqüências de comandos *Object Pascal*, deve estar sempre entre *scriptlets* das páginas PPW e não são permitidos *scriptlets* intercalados;
- Cada página deve conter no máximo, e quando houver necessidade, uma declaração *uses*, uma declaração *type*, uma declaração *const* e uma declaração *var*, e para cada declaração desta é obrigatório o uso de um par exclusivo de *scriptlets* para cada (exemplo de utilização na Figura 4).
- Não é permitida a criação de *procedures* ou *functions* dentro das páginas, porém caso seja necessário a utilização deve-se criar uma *unit* pascal (.pas) e especificar a sua utilização na cláusula *uses* da página PPW, como mostrado na Figura 4.

```
...
<body text="#000000" bgcolor="#FFFFFF" link="#0000FF" vlink="#000080" alink="#FF0000" class="padrao">
<center>
  <p> <span class="padrao"><font size="+3">Resultado do Login</font> </span></p>
<% uses helptela in 'helptela.pas', teste in 'teste.pas'; %>
<%
   type
      TPCookie = ^TNodeCookie;
      TItemCookie = record
                      Name: string;
                      Value: string;
                    end;
%>
<% var i: integer; qry: TSQLQuery; %>
<%
    qry.Close;
    qry.SQLConnection := Session.getConnection('bdteste');
    if not Parameter.IsEmpty then
    begin
        if (Parameter.Value['nome']<>'') and (Parameter.Value['valor']<>'') then
        begin
            Session.Add(Parameter.Value['nome'],Parameter.Value['valor']);
%>
...
```

Figura 4. Exemplo de utilização de cláusulas em página PPW

Existem três objetos, fundamentais para a tecnologia, previamente instanciados e disponíveis para utilização do programador.

Tabela 2. Objetos PPW

Objeto	Classe	Descrição
Parameter	TRequestParameter	Armazena e disponibiliza todos os parâmetros recebidos pela página, via POST ou GET.
Session	TClientSession	Armazena e disponibiliza todas as informações da sessão e das variáveis da sessão, inclusive conexões a banco de dados, que são tratadas como variáveis de sessão.
Cookie	TCookie	Armazena e disponibiliza as informações sobre os *cookies*, permitindo manipular valores e datas de expiração.

As descrições das classes que pertencem estes objetos, juntamente com seus métodos, estão abordados no APÊNDICE A . Alguns exemplos de utilização destes objetos estão na Figura 4 e na Figura 5.

```
<%
    if not Parameter.IsEmpty then
        Cookie.Add(Parameter.Value['nome'],Parameter.Value['valor'],Parameter.Value['exp']);
%>
<%
    Writeln('<B><BR><BR>COOKIES existentes: </B>');
    Writeln(GetEnvironmentVariable('HTTP_COOKIE'));
%>
...
  <p>Vari&aacute;vel <%=Parameter.Value['nome']%> possui o valor
  "<%=Session.GetValue(Parameter.Value['nome'])%>"</p>
<%
        end;
        sTempo_seg := Parameter.Value['tempos'];
        if sTempo_seg<>'' then
        begin
            nTempo_dias := StrToInt(sTempo_seg);
            Session.SetTimeToExpire(nTempo_dias);
%>
```

Figura 5. Exemplo de utilização dos objetos *Parameter*, *Cookie* e *Session*.

Existem mais classes e funções disponíveis que se encontram nas bibliotecas específicas da tecnologia. Elas também foram abordadas na seção 3.1 e estão descritas no APÊNDICE A .

As páginas PPW, e qualquer arquivo pertencente à aplicação, podem ficar em quaisquer pastas, desde que estejam abaixo ou no diretório principal da aplicação. Exceção se faz nas *units* do *Object Pascal*, que podem ficar em qualquer lugar, pois não vão para o servidor de aplicação. Elas só são utilizadas no processo de compilação das páginas.

É importante não esquecer do arquivo *apw.xml*, arquivo de configuração da aplicação PPW, que tem que estar no diretório principal da aplicação. Nele estão especificados o nome da aplicação (que também serve como *alias* de acesso no servidor PPW) e os bancos a serem conectados, bem como as formas e parâmetros de conexão.

Um exemplo do arquivo apw.xml pode-se ver na Figura 2. Mais detalhes de configuração deste arquivo estão devidamente explicados no APÊNDICE A .

Depois de construída as páginas e estabelecido corretamente as configurações do *apw.xml*, o próximo passo é executar o *deployPPW* para a aplicação ser preparada e distribuída em um servidor PPW.

3.2.1.2. O Processo de Distribuição PPW

O processo de distribuição, gerenciado e disparado pelo aplicativo *deployPPW*, executa as seguintes etapas, conforme esquema simplificado ilustrado na Figura 6 .

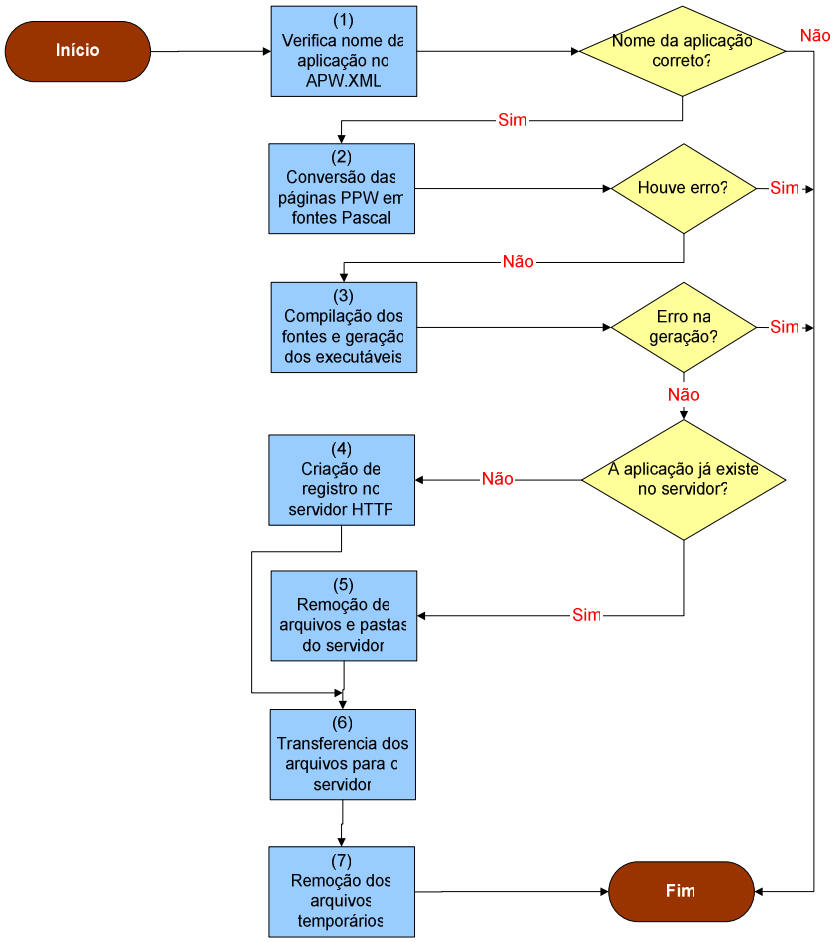

Figura 6. Processo simplificado de execução do DeployPPW

(1) Verifica se o nome da aplicação está correto, se corresponde ao nome que está no arquivo *apw.xml*;

(2) Converte todas as páginas PPW (.ppw), inclusive as que estiverem em sub-pastas dentro da aplicação, em programas fontes Pascal (.pas);

- Na conversão de páginas PPW em fontes Pascal são adicionadas linhas de código essenciais ao uso da tecnologia e de seus objetos disponibilizados. Essas linhas servem apenas para auxiliar e efetivar as funcionalidades das páginas diante da tecnologia, nunca alterá-las na sua essência.

(3) Compila todos os programas fontes Pascal gerados, criando executáveis, um para cada programa fonte;

- Em caso de erro, a mensagem é apresentada em tela e o programa fonte (.pas) que gerou o erro não é apagado; fica disponível para melhor inspeção e depuração posterior.

(4) Se o processamento até aqui está isento de erros e a aplicação ainda não existir antes, é criado um registro no arquivo de configuração do servidor HTTP especificando o *alias* da aplicação e criado uma pasta específica para ela no servidor PPW;

(5) Se o processamento até aqui está isento de erros, mas a aplicação já existia previamente, ou seja é apenas uma atualização da mesma, o diretório principal da aplicação e todo o seu conteúdo no servidor PPW é completamente removido e é criada novamente uma pasta específica para a aplicação no servidor PPW;

(6) Todos os arquivos do diretório principal de desenvolvimento e todas as suas sub-pastas são trazidos para o servidor e colocados na pasta específica da aplicação;

(7) No final, são removidos automaticamente todos os arquivos temporários do ambiente de desenvolvimento (.exe) e do servidor (.pas, .dcu, .obj) gerados no processo;

Todo o processo é controlado, etapa por etapa. Caso ocorra algum problema, é sempre apresentada uma mensagem de erro clara, personalizada e vinculada à etapa que ocorreu o problema.

Em caso de sucesso, a aplicação está distribuída e disponível no servidor PPW em *http://<url_do_servidor>/<nome_da_aplicação>*.

3.2.2. O servidor PPW

É chamado de servidor PPW a parte *server-side* da tecnologia que gerencia o processamento da aplicação PPW requisitada. Uma representação adequada do seu funcionamento é mostrada na Figura 3. Será apresentado a seguir suas características básicas. Maiores detalhes são encontrados no APÊNDICE A .

O servidor PPW trabalha em uma estrutura de diretórios independentes ao do servidor HTTP utilizado, no caso, o Apache. Isto visa, em futuro próximo, torná-lo independente de servidor HTTP e facilitar a sua flexibilidade em usar outros servidores HTTP, pois bastaria substituir as rotinas de configuração do servidor Apache, pelas do servidor HTTP desejado. Mas atualmente ele trabalha junto ao servidor HTTP Apache, conforme mencionado na seção 1.2.

Para o funcionamento do servidor PPW, o programa *PPWController* tem que estar em operação, junto,é claro, com o servidor HTTP. O *PPWController* é o responsável pelo controle de sessões e conexões, como já explicado antes. Funciona como um repositório central de informações para a aplicação/cliente. Enquanto o programa

está em processamento na memória, fica escutando a porta 81, através do protocolo TCP, e respondendo em caráter *multi-thread* (multi-tarefa) às requisições das aplicações PPW.

O servidor PPW trabalha com um objeto *Sessions*, que pertence à classe *TList*. A classe *TList*, já proveniente do *Delphi 7*, representa uma lista (ou coleção) on-line de objetos. Neste caso, cada objeto da lista corresponde a uma sessão web de uma determinada aplicação PPW, guardando todas as informações referentes à sessão, inclusive conexões e variáveis. Ele é identificado pelo número de identificação da sessão e aplicação PPW correspondente.

Para uma aplicação PPW iniciar é necessário a requisição a uma página PPW. Quando isto acontece, o servidor HTTP chama o aplicativo CGI correspondente à página PPW, que possui o mesmo nome da página, apenas acrescentado da extensão *.exe*. O aplicativo chamado é executado e, se necessário for para sua execução, captura as informações de parâmetros passadas através do GET e POST utilizando o objeto *Parameter*, conforme citado na seção 3.2.1.1. O aplicativo CGI em execução, doravante chamado de aplicativo PPW, pode também interagir com o *PPWController* através de três tipos de requisições: sessão, variável de sessão e variável de conexão a banco de dados.

3.2.2.1. Requisições ao Servidor PPW

A requisição do tipo sessão possui três subtipos: nova sessão, alteração de tempo de expiração da sessão e validação da sessão. As requisições e os respectivos retornos são feitos e tratados pelo objeto *Session*.

A requisição de nova sessão fornece um novo número de sessão ao cliente. Esta requisição é feita automaticamente quando o cliente faz a sua primeira requisição de página ao servidor.

A alteração do tempo de expiração refere-se à alteração do tempo máximo de tolerância para requisição ao servidor em uma sessão válida. O tempo padrão de expiração de sessão é de 30 (trinta) minutos quando a sessão é criada.

A requisição de validação de sessão testa se a sessão ainda é válida, analisando especificamente o tempo de expiração. Se o tempo entre requisições é maior do que o tempo marcado para expiração, a sessão é invalidada. Possui também a requisição que faz com que a sessão seja invalidada automaticamente, alterando para isso o tempo de expiração para -1.

A requisição do tipo variável de sessão possui quatro subtipos: criação de variáveis, alteração de valores de variáveis, remoção da variável e consulta ao valor da variável. Esses valores são armazenados de acordo com o número de identificação da sessão e respectiva aplicação.

A requisição do tipo variável de conexão a banco de dados funciona da seguinte maneira: quando a requisição é feita, o *PPWController* tenta estabelecer uma conexão com o banco de dados, conforme configuração informada no *apw.xml* da aplicação. Caso consiga, a variável de conexão obtida é armazenada no *PPWController* referente a sessão que o requisitou. Depois é convertida em string e enviada ao aplicativo PPW requisitante, que a reverte em um objeto de conexão. Em toda requisição de uma mesma conexão por uma mesma sessão (aplicação/cliente) ao servidor, a conexão não é refeita, pois a variável de conexão já existe armazenada no *PPWController*. Ela é apenas enviada para o requisitante. A variável de conexão só pode ser criada quando ainda não existe para uma determinada sessão. Mais detalhes estão abordados no APÊNDICE A .

3.2.2.2. Tratamento de Cookies

Todos os *cookies* da aplicação, porventura existentes na máquina cliente, são capturados pelo servidor PPW, deixando todos eles disponíveis nas páginas PPW através do objeto *Cookie*. É possível a qualquer momento criar um novo *cookie* de sessão ou com uma determinada data de expiração. Mais detalhes de implementação das funcionalidades do objeto *Cookie*, são encontrados na classe *TCookie* descrita no APÊNDICE A . Lembrando apenas que o *cookie* só é criado de fato na máquina cliente depois da submissão da página PPW que requisitou sua criação. É bom ressaltar que a geração efetiva de *cookies* está condicionada às definições de privacidade do navegador HTTP do cliente.

4. COMPARATIVOS

4.1. DESEMPENHO COMPARATIVO COM ASP, JSP E PPW

Foram realizados comparativos entre desempenhos de aplicações ASP, JSP e PPW, com o objetivo de avaliar a performance de execução da tecnologia PPW, comparando-a com tecnologias de características similares existentes e consagradas no mercado.

Dois tipos de desempenhos foram analisados:

- Através de um processamento complexo em uma única requisição, onde foi verificado o tempo de resposta em várias situações;
- Através de várias requisições simultâneas de processamento simples, onde foi verificado o número de atendimento requisições, caracterizando um teste de desempenho sob stress.

A seguir, um detalhamento maior dos testes realizados.

4.1.1. Testes de Desempenho com Processamento Complexo

Neste tipo de teste, foram elaborados três testes de desempenho. O primeiro baseou-se apenas no processamento das páginas com alguma complexidade matemática, o segundo baseou-se no acesso a banco de dados e o terceiro fez uma

análise mista de processamento de páginas com acesso a banco. No nosso trabalho, o desempenho foi medido como o tempo total necessário para exibir uma página em resposta a uma solicitação com graus variados de complexidade.

O único parâmetro utilizado para medida de desempenho foi o tempo de resposta da aplicação a uma requisição HTTP, com o grau de complexidade da requisição aumentando gradativamente.

Foram desenvolvidas três aplicações para realizar estes testes de desempenho, uma para cada tecnologia. Elas possuem telas de entradas iguais para o processamento destes três testes. Aliás, todas as páginas de todas as aplicações são rigorosamente iguais em suas páginas HTML. Cada opção de submissão de tela é correspondente ao tipo de teste realizado. A tela inicial corresponde à Figura 7.

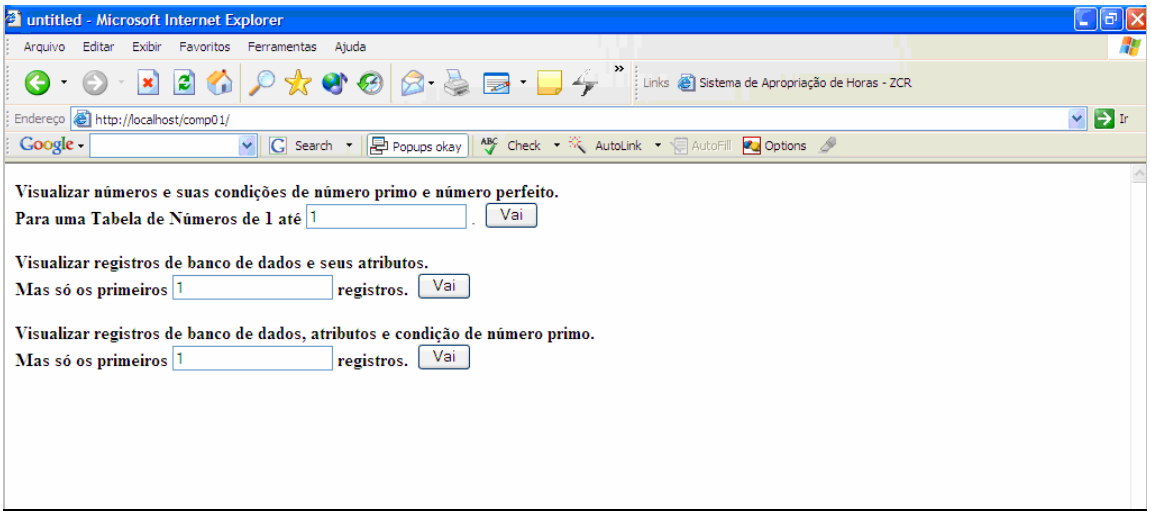

Figura 7. Tela de abertura da aplicação de testes de desempenho

O primeiro botão “vai” realiza o primeiro teste, o segundo “vai” realiza o segundo teste e o terceiro “vai” realiza o terceiro teste. Cada botão deste se encontra em

formulários HTML distintos. Esta página contém três formulários, um para cada teste.

4.1.1.1. O Primeiro Teste

O primeiro teste consiste em exibir uma tabela de números entre 1 e um outro número desejado, atribuindo sim ou não, se o número fosse primo e/ou perfeito. É acionado pelo primeiro botão da tela de abertura. Um exemplo do resultado obtido entre 1 e 10 segue na Figura 8.

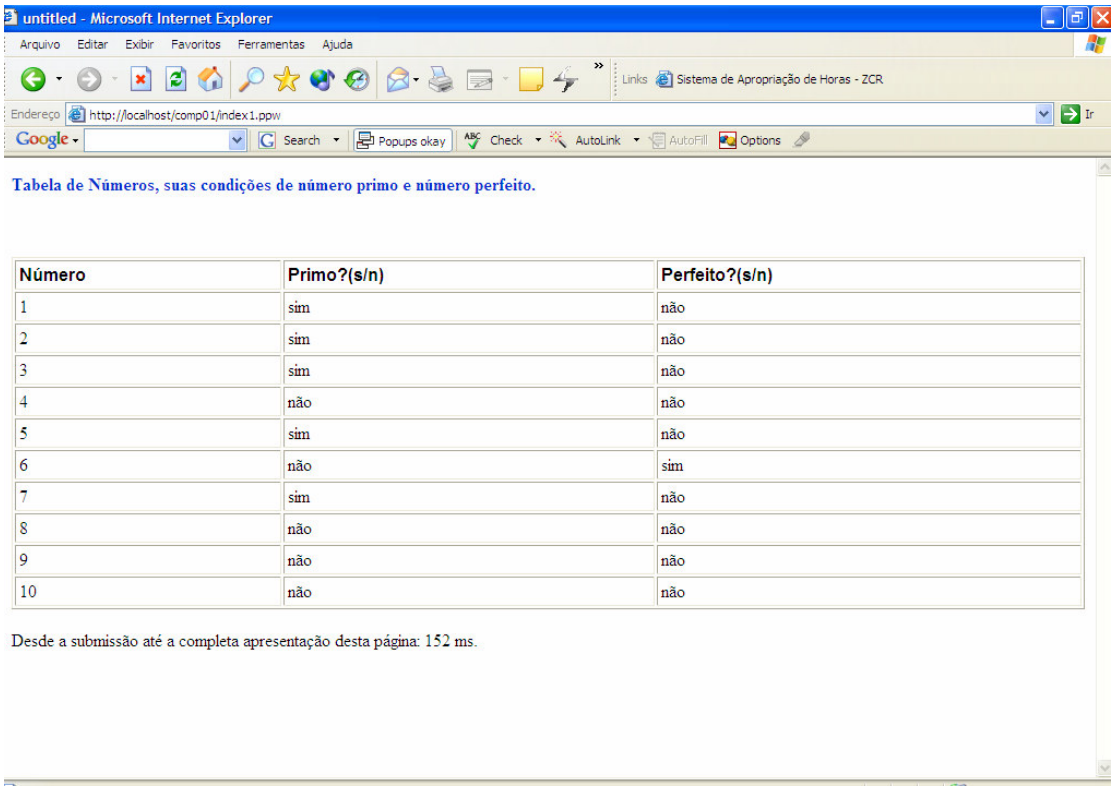

Figura 8. Tela de resultados da tabela de números

A última frase da tela "Desde a submissão até a completa apresentação da tela: 152ms" representa o tempo total decorrido entre o clique do botão "vai" da tela de abertura e

a completa montagem e apresentação desta tela de apresentação de resultados. Este tempo é fundamental para medida de desempenho.

Para obter este tempo total, o botão “vai” foi definido da seguinte maneira:

```
<input type="button" value="  Vai  " onClick="submitForm(0);">
```

É definido também um campo escondido da seguinte maneira:

```
<input type="hidden" name="datasub" value="">
```

A função *javaScript* acionado no clique do botão é a seguinte:

```
function submitForm(ind){
  document.forms[ind].datasub.value = new Date();
  document.forms[ind].submit();
}
```

Nesta função, ele preenche o campo “datasub”, do formulário enviado, com a data/hora atual e submete o formulário para a página que contém a tela de apresentação da tabela de números.

Ao final do preenchimento desta tela de apresentação da tabela de números existe o seguinte código *javaScript*:

```
  var difjs = new Date() - new Date("<%=jsDataSub%>");
  document.write("<p>Desde  a  submiss&atilde;o  at&eacute;  a  completa
apresenta&ccedil;&atilde;o desta p&aacute;gina: "+difjs+" ms.</p>");.
```

Onde *jsDataSub* é obtido do valor do campo “datasub” submetido do formulário anterior. Como este código é a última linha da tela, isto garante o tempo total decorrido de processamento e exibição da tela, desde a sua submissão.

O objetivo deste primeiro teste é medir o tempo que cada tecnologia leva para exibir páginas de resultados, somente utilizando o processo de submissão de página, preparação da página a mostrar, com processamento de cálculos matemáticos.

A seguir, até para um melhor entendimento da tecnologia, a página PPW que mostra os resultados deste teste.

```
<html><body><p> <font color="#0033cc"><strong>Tabela de N&uacute;meros,
suas condi&ccedil;&otilde;es de n&uacute;mero primo e n&uacute;mero
perfeito.</strong ></font> </p>
<table cellspacing="2" cellpadding="3" border="1" width="100%">
 <tr> <td> <font face="Arial"><STRONG>N&uacute;mero</strong> </font>
 </td><td> <font face="Arial"><STRONG>Primo?(s/n) </strong> </font>
 </td><td> <font face="Arial"><STRONG>Perfeito?(s/n) </strong> </font> </td> </tr>
<% var i,j,somaDiv: integer; sPerf,sPrimo,jsDataSub: string; t1: TDateTime;%>
<%
  t1 := Now;
  for i:=1 to StrToInt(Parameter.Value['valor_max']) do
  begin
    somaDiv:=0; sPerf:='não'; sPrimo:= 'não';  j := 2;
    while ((i mod j)<>0) and (j<i) do j := j + 1;
    if (i=j) or (i=1) then sPrimo := 'sim';
    for j:=1 to i-1 do
      if (i mod j)=0 then somaDiv:=somaDiv+j;
    if somaDiv=i then sPerf := 'sim';
%>
 <font color="#0000ff"><strong>
 <tr> <td> <%=i%> </td>
 <td> <%=sPrimo%> </td>
 <td> <%=sPerf%> </td> </tr>
 </strong></font>
<%
  end;
  t1 := (Now - t1)*24*3600*1000;
  writeln('<P> Constru&ccedil;&atilde;o desta p&aacute;gina: ',t1:1:0,' ms.</P>');
  jsDataSub := Parameter.Value['datasub'];
```

```
%>
</table>
</body></html>
<script>
  var difjs = new Date() - new Date("<%=jsDataSub%>");
  document.write("<p>Desde a submiss&atilde;o at&eacute; a completa apresenta&ccedil;&atilde;o
desta p&aacute;gina: "+difjs+" ms.</p>");
</script>
```

4.1.1.2. O Segundo Teste

O segundo teste consiste em exibir uma tabela resultante de consulta a registros de banco de dados. Foi escolhida uma tabela que refletisse um documento qualquer com um grande volume de dados (aproximadamente 500.000 registros) e mostrasse alguns de seus valores. Esta tabela continha um campo do tipo *booleano* (verdadeiro ou falso) e para exibi-lo foram escolhidos duas imagens do tipo GIF, uma para a condição verdadeira e a outra para a condição falsa, o que enriqueceria mais ainda o teste. Esta tela de resultados mostrada a seguir, é acionada pelo segundo botão “vai” da tela de abertura. Um exemplo do resultado obtido de 10 registros é mostrado na Figura 9.

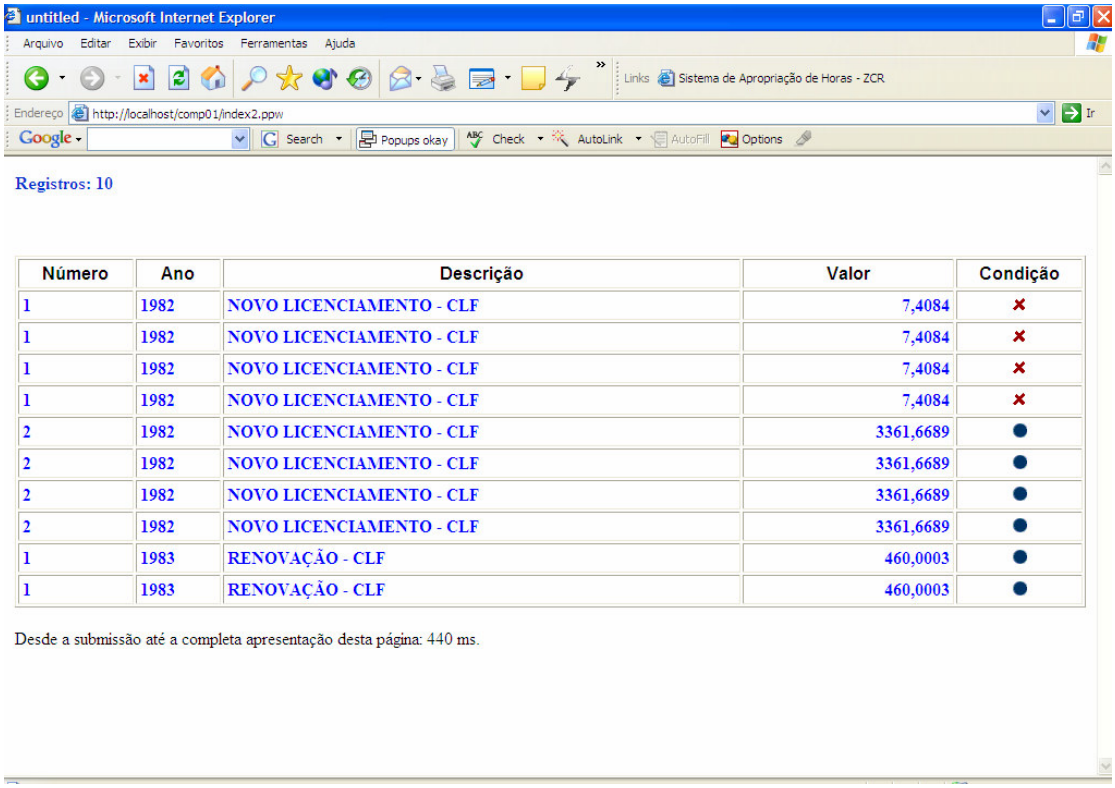

Figura 9. Tela de resultados dos registros consultados

O tempo total decorrido é obtido de maneira idêntica à descrita no primeiro teste.

O objetivo deste segundo teste é medir o tempo que cada tecnologia leva para exibir páginas de resultados, utilizando valores obtidos de consultas a registros de tabelas de banco de dados, além do processo de submissão de página e preparação da página a mostrar, acrescida de imagens.

A seguir a página PPW que mostra os resultados deste teste.

```
<html> <body>
<% var qry: TSQLQuery; t1:TDateTime; jsDataSub: string; %>
<p> <font color="#0033cc"><strong>Registros:
<%=Parameter.Value['max_linhas']%></strong></font> </p>
<table cellspacing="2" cellpadding="3" border="1" width="100%">
  <tr> <td width="11%"> <div align="center"><font face="Arial"> <strong>
N&uacute;mero</strong> </font> </div></td>
```

```
  <td width="8%"> <div align="center"><font face="Arial"><strong>Ano</strong>
        </font> </div></td>
    <td width="49%"> <div align="center"> <font face="Arial">
<strong>Descri&ccedil;&atilde;o</strong> </font> </div></td>
    <td width="20%"> <div align="center"> <font face="Arial"><strong>Valor</strong>
        </font> </div></td>
    <td width="12%"> <div align="center"> <font face="Arial">
<strong>Condi&ccedil;&atilde;o</strong> </font> </div></td>  </tr>
<%
  t1 := Now;
  qry := TSQLQuery.Create(Application);
  qry.Close;
  qry.SQLConnection := Session.GetConnection('sgci') ;
  qry.SQL.Text := 'select top ' + Parameter.Value['max_linhas'] +
                  ' Numero,Exercicio,Descricao,ValorTotal,Pago ' +
                  ' from dam d join tipodam td on' +
                  ' d.codtipodam=td.codtipodam order by Exercicio,Numero';
  qry.Open;
  while not qry.Eof do
  begin
%>
  <font color="#0000ff"><strong> </strong></font>
  <tr>
    <td> <font color="#0000ff"><strong><%=qry.FieldByName('Numero').Value%>
</strong></font></td>
    <td> <font color="#0000ff"><strong><%=qry.FieldByName('Exercicio').Value%>
</strong></font></td>
    <td> <font color="#0000ff"><strong><%=qry.FieldByName('Descricao').Value%>
</strong></font></td>
    <td> <div align="right"><font
color="#0000ff"><strong><%=qry.FieldByName('ValorTotal').Value%>
        </strong></font></div></td>
    <td> <div align="center"><font color="#0000ff"><strong>
<%
    if qry.FieldByName('Pago').Value then
      begin
%>
 <img src="pago.gif">
<%     end
    else
      begin
%>
 <img src="naopago.gif">
<%
      end;
%>
 </strong></font></div>
  <font color="#0000ff"><strong></strong></font></td></tr>
  <font color="#0000ff"><strong> </strong></font>
<%
    qry.Next;
```

```
  end;
  t1 := (Now - t1)*24*3600*1000;
  writeln('<P> Constru&ccedil;&atilde;o desta p&aacute;gina: ',t1:1:0,' ms.</P>');
  jsDataSub := Parameter.Value['datasub'];
%>
</table>
</body></html>
<script>
  var difjs = new Date() - new Date("<%=jsDataSub%>");
  document.write("<p>Desde a submiss&atilde;o at&eacute; a completa apresenta&ccedil;&atilde;o
desta p&aacute;gina: "+difjs+" ms.</p>");
</script>
```

4.1.1.3. O Terceiro Teste

O terceiro teste consiste em exibir uma tabela resultante de consulta a registros de banco de dados, combinados com processos matemáticos. Novamente foi escolhida uma tabela que refletisse um documento qualquer com um grande volume de dados (aproximadamente 500.000 registros), mostrasse alguns de seus valores e estabelecesse se um determinado campo (no caso, a coluna “Número”) seria um número primo ou não. Este seria o processo matemático combinado. Semelhante ao segundo teste, foram também utilizadas duas imagens do tipo GIF, uma para a condição verdadeira e a outra para a condição falsa, pois enriqueceria mais ainda o teste. Esta tela de resultados mostrada a seguir, é acionada pelo terceiro botão “vai” da tela de abertura. Um exemplo do resultado obtido de 10 registros é mostrado na Figura 10.

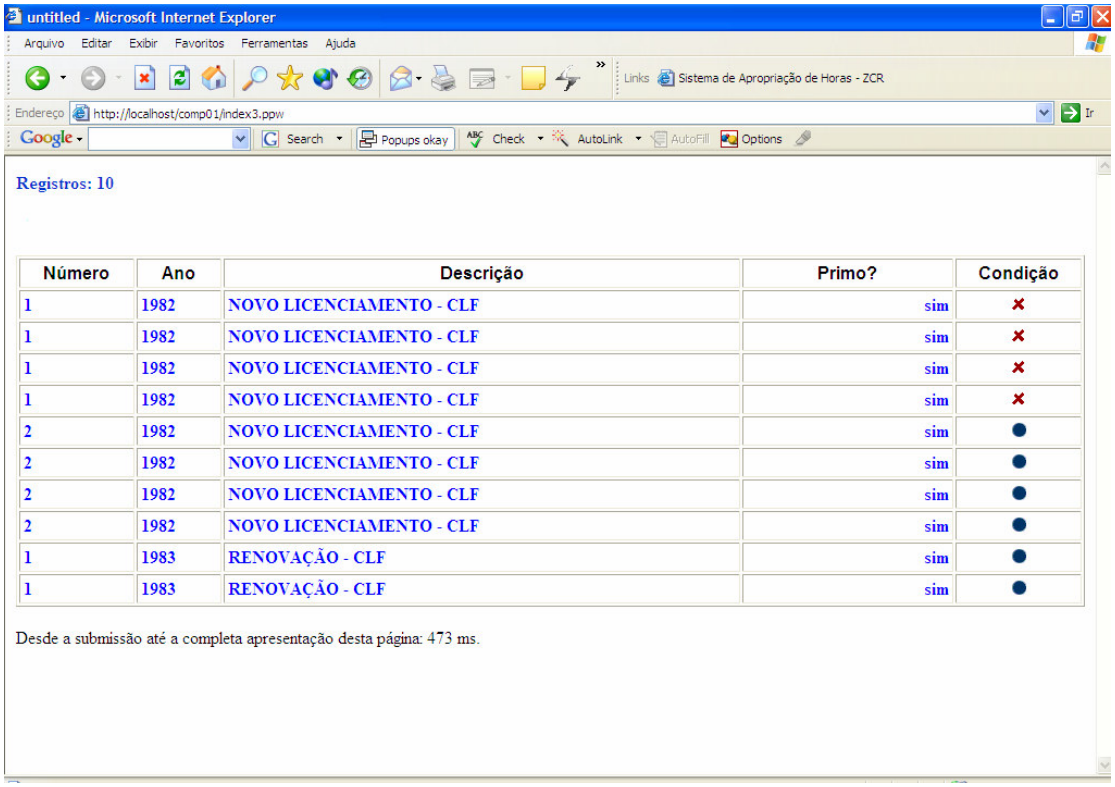

Figura 10. Tela de resultados dos registros consultados e combinados

O tempo total decorrido é obtido de maneira idêntica à descrita no primeiro teste.

O objetivo deste terceiro teste é medir o tempo que cada tecnologia leva para exibir páginas de resultados, utilizando valores obtidos de consultas a registros de tabelas de banco de dados, combinados com processamento matemático, além do processo de submissão de página e preparação da página a mostrar, acrescida de imagens.

A seguir a página PPW que mostra os resultados deste teste.

```
<html>
<body>
<% var qry: TSQLQuery; t1:TDateTime; jsDataSub,sPrimo: string; i,j: integer;%>
<p> <font color="#0033cc"><strong>Registros:
<%=Parameter.Value['max_linhas']%></strong></font> </p>
<table cellspacing="2" cellpadding="3" border="1" width="100%">
```

```
  <tr> <td width="11%"> <div align="center"> <font face="Arial">
<strong>N&uacute;mero</strong> </font> </div></td>
  <td width="8%"> <div align="center"><font face="Arial"><strong>Ano</strong>
        </font> </div></td>
    <td width="49%"> <div align="center"> <font face="Arial">
<strong>Descri&ccedil;&atilde;o</strong> </font> </div></td>
    <td width="20%"> <div align="center"><font face="Arial"><strong>Primo?</strong>
        </font> </div></td>
    <td width="12%"> <div align="center"> <font face="Arial">
<strong>Condi&ccedil;&atilde;o</strong> </font> </div></td> </tr>
<%
  t1 := Now;
  qry := TSQLQuery.Create(Application);
  qry.Close;
  qry.SQLConnection := Session.GetConnection('sgci') ;
  qry.SQL.Text := 'select top ' + Parameter.Value['max_linhas'] +
                  ' Numero,Exercicio,Descricao,ValorTotal,Pago ' +
                  ' from dam d join tipodam td on' +
                  ' d.codtipodam=td.codtipodam order by Exercicio,Numero';
  qry.Open;
  while not qry.Eof do
  begin
    sPrimo:= 'não';
    j := 2;
    i := qry.FieldByName('Numero').Value;
    while ((i mod j)<>0) and (j<i) do j := j + 1;
    if (i=j) or (i=1) then
      sPrimo := 'sim';
%>
  <font color="#0000ff"><strong> </strong></font>
  <tr> <td> <font color="#0000ff"><strong><%=qry.FieldByName('Numero').Value%>
</strong></font></td>
    <td> <font color="#0000ff"><strong><%=qry.FieldByName('Exercicio').Value%>
</strong></font></td>
    <td> <font color="#0000ff"><strong><%=qry.FieldByName('Descricao').Value%>
</strong></font></td>
    <td> <div align="right"><font color="#0000ff"><strong><%=sPrimo%>
        </strong></font></div></td>
    <td> <div align="center"><font color="#0000ff">
<strong>
<%
   if qry.FieldByName('Pago').Value then
   begin
%>
 <img src="pago.gif">
<% end
   else
   begin %>
 <img src="naopago.gif">
<% end; %>
 </strong>
```

```
</font></div>
 <font color="#0000ff"><strong></strong></font></td> </tr>
 <font color="#0000ff"><strong> </strong></font>
<%
    qry.Next;
  end;
  t1 := (Now - t1)*24*3600*1000;
  writeln('<P> Constru&ccedil;&atilde;o desta p&aacute;gina: ',t1:1:0,' ms.</P>');
  jsDataSub := Parameter.Value['datasub'];
%>
</table>
</body>
</html>
<script>
  var difjs = new Date() - new Date("<%=jsDataSub%>");
  document.write("<p>Desde a submiss&atilde;o at&eacute; a completa apresenta&ccedil;&atilde;o
desta p&aacute;gina: "+difjs+" ms.</p>");
</script>
```

4.1.1.4. O Processo de Coleta

Para a coleta de dados foram utilizadas as mesmas máquinas e configurações, tanto servidor como cliente, para cada tecnologia. Segue as características de cada uma.

Tabela 3. Equipamentos e características utilizados nos testes

Máquina	Configuração	Sistema Operacional	Carga Memória
Cliente	Pentium IV 2.66GHz e 512MBRAM	Windows XP Professional	212MBRAM
Servidor	Pentium IV 2.80GHz e 1GBRAM	Windows 2000 Server	362MBRAM
MS-SQL	Pentium IV 1.4GHz e 1GBRAM	Windows 2000 Server	451MBRAM

Importante salientar que, em todo o processo de teste, nenhuma dessas máquinas estava usando recursos para outros programas e/ou usuários. As máquinas estavam dedicadas a estes testes.

Para a tecnologia ASP, foi utilizado o servidor IIS 5.1 da **Microsoft Corp.**. Para a tecnologia JSP foi utilizado o JBOSS 3.2.1, da **JBoss Inc.**, conjugado ao TOMCAT 4.1.24, da **Apache Software Foundation**, ambos utilizando a máquina virtual Java do JDK1.4.2 (*Java Development Kit* v.1.4.2) da **Sun Microsystems**.

O banco utilizado foi o MS-SQL Server 2000, da **Microsoft Corp.**. A conexão ao banco pelo ASP foi realizada através do ADO (*ActiveX Data Objects*). No JSP foi utilizado um *driver* JDBC específico do MS-SQL. Para o PPW, como já mencionado, a conexão foi feita através do *dbExpress*.

Os testes foram realizados no ambiente de rede local da empresa **ZCR Informática**, na madrugada do dia 7 de fevereiro de 2005, quando não havia ninguém utilizando a rede interna, mas a rede estava em pleno funcionamento e, obviamente, com os seus processos de gerenciamento e controle ativos.

Para todos os três testes, foi estabelecida uma variação de apresentação de tabelas de 1.000 (mil) resultados até 100.000 (cem mil) resultados.

O primeiro teste foi o início da bateria. Foram realizados todos os testes primeiro com a tecnologia JSP, depois com a tecnologia PPW (execução *server-side*) e por último com a tecnologia ASP. Foi estabelecido que nunca seria considerada a primeira vez em cada teste e seria considerado como resultado válido o menor resultado de tempo total obtido entre três solicitações sucessivas a partir da primeira.

Com estas considerações foi obtida a seguinte tabela de coleta de dados para o primeiro teste:

Tabela 4. Processamento de dados (tempo em ms)

Qtd Dados	1.000	5.000	10.000	25.000	50.000	75.000	100.000
ASP	921	19.035	73.890	426.086	N/D	N/D	N/D
JSP	185	815	2.036	9.603	34.701	75.347	131.422
PPW	234	767	1.863	9.085	33.509	73.782	129.817

A tecnologia ASP, a partir de 25.000 registros neste teste, já possuía números bastante elevados e, portanto, não foi necessário prosseguir a partir daí, pois deturparia a escala gráfica do resultado final.

Foram seguidas as mesmas considerações para o segundo teste, apenas modificando a ordem das tecnologias. Foram realizados todos os testes primeiro com a tecnologia ASP, depois com a tecnologia JSP e por último com a tecnologia PPW.

Foi obtida a seguinte tabela de coleta de dados para o segundo teste:

Tabela 5. Consulta em BD (tempo em ms)

Qtd Dados	1.000	5.000	10.000	25.000	50.000	75.000	100.000
ASP	684	4.656	6.815	12.212	22.410	34.525	71.374
JSP	5.174	9.788	11.118	16.371	28.985	40.231	51.962
PPW	711	4.868	6.563	12.226	22.529	32.969	44.369

Para o terceiro teste, novamente foram seguidos as mesmas considerações dos testes anteriores, apenas modificando a ordem das tecnologias. Foram realizados todos os testes primeiro com a tecnologia PPW, depois com a tecnologia ASP e por último com a tecnologia JSP.

Foi obtida a seguinte tabela de coleta de dados para o terceiro teste:

Tabela 6. Consulta em BD com processamento de dados (tempo em ms)

Qtd Dados	1.000	5.000	10.000	25.000	50.000	75.000	100.000
ASP	802	6.631	10.117	22.777	47.408	78.150	105.299
JSP	5.219	9.210	11.933	16.095	28.272	41.041	53.610
PPW	690	4.567	6.820	12.088	21.671	33.140	45.478

4.1.1.5. Análise dos Resultados

Após os testes, todos os resultados foram colocados em gráfico para uma melhor análise. O impacto visual mostra-se bastante útil na visão geral.

No primeiro teste foi obtido o resultado mostrado na Figura 11.

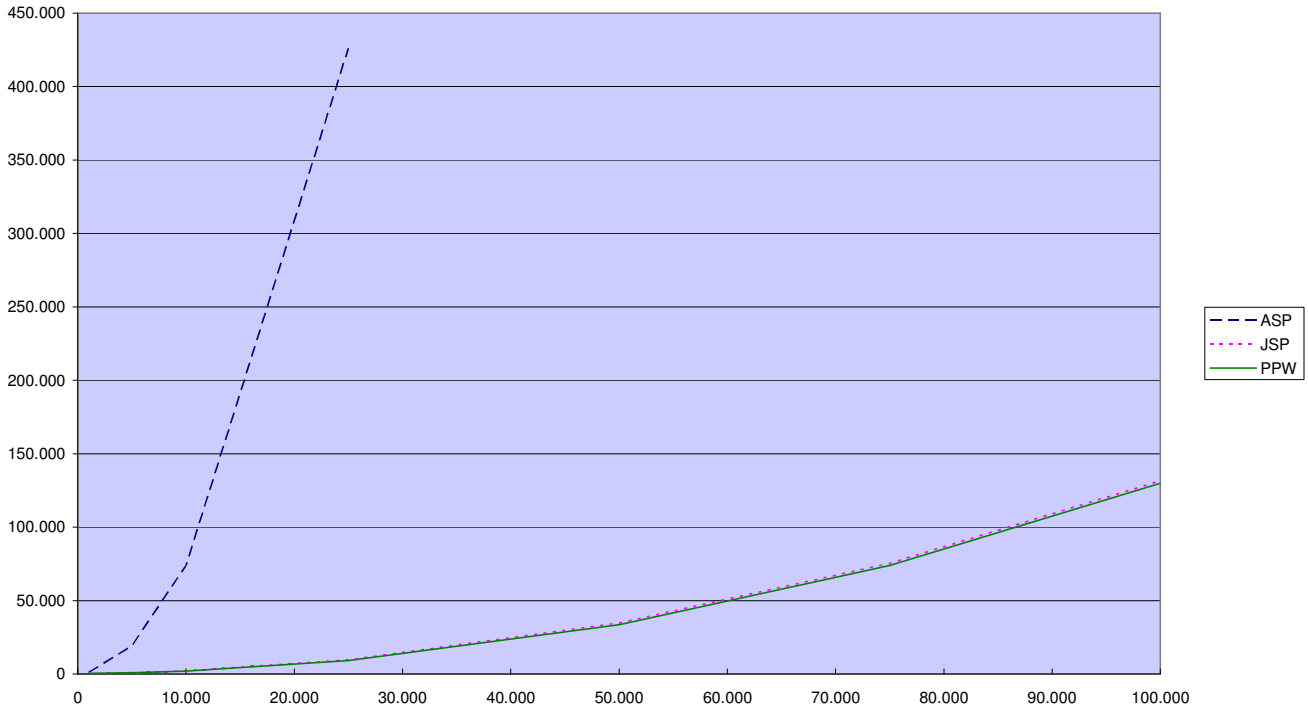

Figura 11. Gráfico de resultados do primeiro teste

É bom observar que a escala gráfica foi deturpada pelo baixo desempenho do ASP neste teste. Isto fez com que não fosse possível ver uma diferença real entre o desempenho do JSP e do PPW.

Considerando apenas as tecnologias JSP e PPW, este gráfico fica como mostrado na Figura 12.

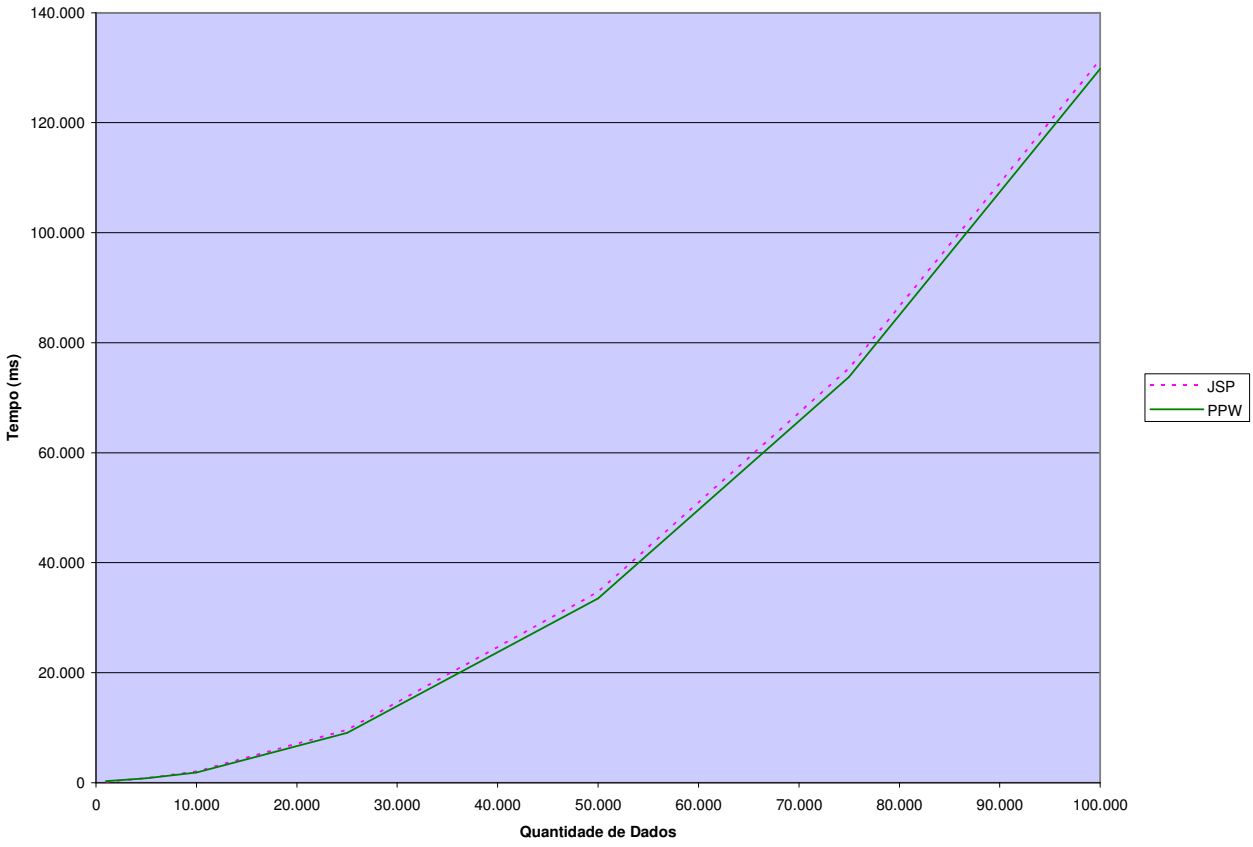

Figura 12. Gráfico de resultados do primeiro teste, apenas JSP e PPW

Pode-se ver que o PPW teve um comportamento um pouco melhor do que o JSP, notadamente a partir de 25.000 registros de dados. Este comportamento foi quase uma constante, como podemos notar pela similaridade e eqüidistância das curvas. A diferença máxima de tempo aconteceu no processamento de 100.00 registros,

quando a diferença chegou a 1.605 ms a favor do PPW. Isto significa uma diferença relativa de 1,22%. A maior diferença relativa aconteceu no processamento de 10.000 registros, onde alcançou 8,50%, a favor do PPW. Curiosamente no processamento de 1.000 registros o JSP foi mais rápido, conseguindo uma diferença de 26,50% ao seu favor, mas isso significa apenas uma diferença de 49 ms, que pode ter acontecido por condições diversas fora do controle do escopo de teste, pois não se repetiu com a continuidade deste teste.

Conclusão do primeiro teste: O PPW foi ligeiramente melhor do que o JSP e as duas acentuadamente melhores do que o ASP, aliás, este último teve um desempenho muito inferior às demais tecnologias neste primeiro teste. A diferença foi muito pouca entre PPW e JSP, quase imensurável, mas como se manteve constante em quase todo o teste, o PPW foi a tecnologia *server-side* de melhor desempenho neste primeiro teste.

No segundo teste obtivemos o resultado conforme mostrado na Figura 13.

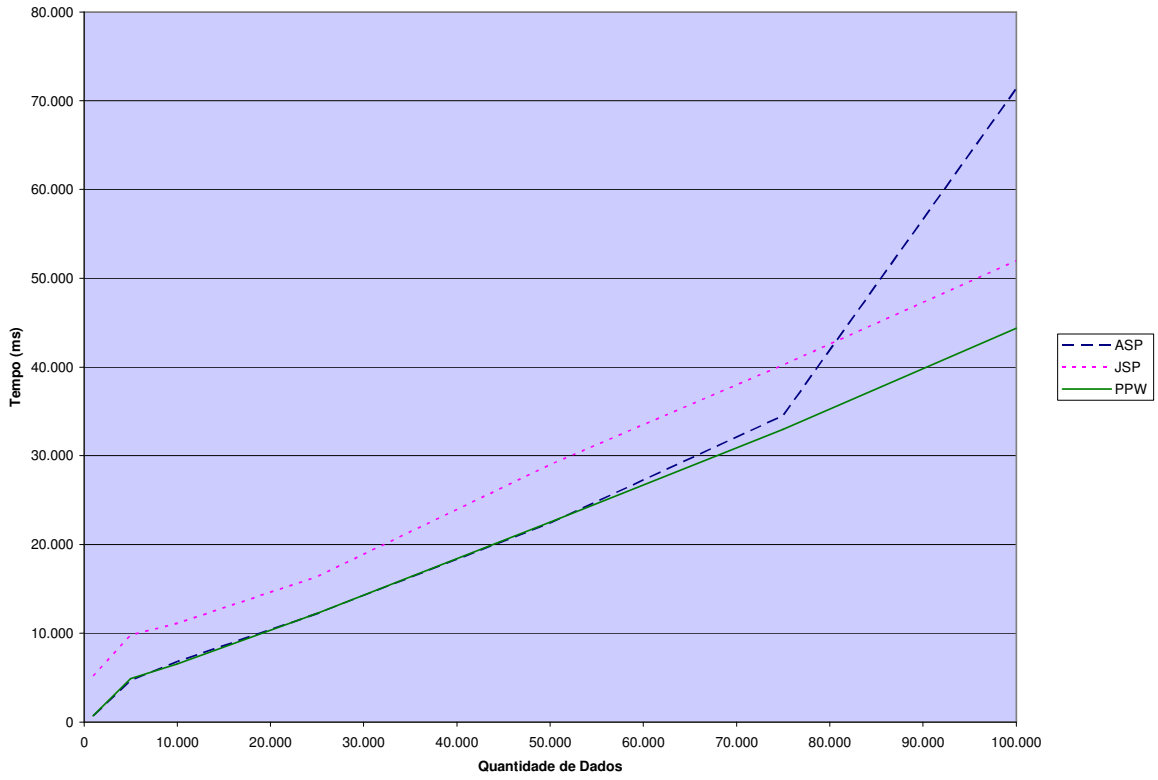

Figura 13. Gráfico de resultados do segundo teste

Neste gráfico é possível ver que o ASP estava com um bom comportamento até 75.000 registros de consulta, mas a partir daí seu desempenho caiu vertiginosamente. Inclusive no seu começo, até os 5.000 registros consultados, ele chegou a superar o PPW em até 212 ms. O JSP teve um desempenho abaixo do PPW ao longo de todo o teste, e só conseguiu superar o ASP após a queda súbita de desempenho deste último. O PPW conseguiu uma boa superioridade em relação ao JSP, conseguindo estabelecer uma diferença média relativa de 36,82% e uma diferença média absoluta de 5.628 ms.

É notável e curiosa a queda vertiginosa do ASP após a consulta de 75.000 registros. Não foi encontrado nenhum registro ou observação do gênero no site da MSDN (*Microsoft Developer Network*) [18] . Mesmo antes disso ele não conseguia superar o

PPW, exceto somente até os 5.000 registros. Porém, o ASP mantinha um desempenho muito próximo ao do PPW até os 75.000 quando, a partir daí, o ASP ficou com um desempenho tão baixo que, aos 100.000 registros consultados a diferença absoluta chega a 27.005 ms, quase meio minuto..

Conclusão do segundo teste: O PPW foi superior ao JSP em toda a extensão deste teste e superior ao ASP a partir dos 5.000 registros consultados. O ASP teve baixo desempenho em grandes volumes de registros. Portanto, neste segundo teste, o PPW registrou também um melhor desempenho do que o ASP e o JSP.

No terceiro teste, uma espécie de misto dos dois anteriores, foi obtido o resultado conforme mostrado na Figura 14.

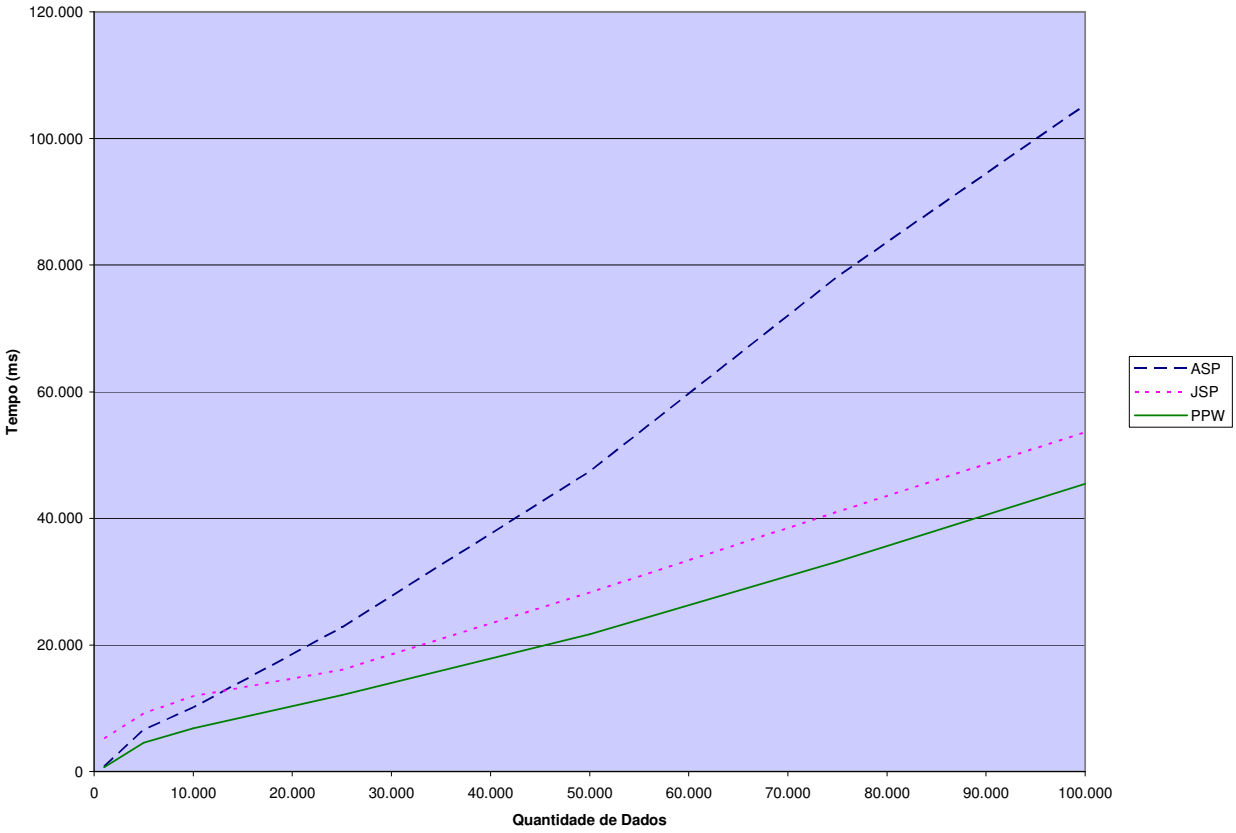

Figura 14. Gráfico de resultados do terceiro teste

Um resultado com esse perfil era o esperado, pois este teste absorve as características dos outros dois testes anteriores, pois possui consulta a registros de banco e também processamentos matemáticos.

O PPW obteve um desempenho sempre superior às outras duas tecnologias durante todo este terceiro teste. Conseguiu uma diferença média de 5.847 ms em relação ao JSP e uma diferença média de 20.961 ms em relação ao ASP. O ASP se comportou melhor que o JSP até os 10.000 registros, pois prevaleceu seu melhor desempenho em consultas com número menor de registros. Mas, a partir deste ponto, prevaleceu seu mau desempenho em processamento matemático observado no primeiro teste e ele começou a registrar uma forte queda no desempenho e chegou aos 100.000 registros com uma diferença de 51.689 ms a favor do JSP.

Conclusão do terceiro teste: O PPW foi superior ao JSP e ao ASP em toda a extensão deste teste. A diferença de desempenho entre o JSP e o PPW foi equivalente a do teste anterior e somente o ASP teve um desempenho bem abaixo de todos. Com isso, pode-se concluir que o PPW obteve um melhor desempenho também neste terceiro teste.

4.1.1.6. Conclusão

Nos testes realizados o PPW conseguiu um melhor desempenho do que o ASP e o JSP em quase toda a sua extensão, nas condições estabelecidas. Um bom resultado para uma tecnologia que se inicia, apesar do desempenho apenas um pouco superior ao do JSP, mas muito superior ao do ASP nas condições do teste

realizado. Convém estudar possibilidades de outros tipos de testes comparativos de desempenhos para confirmar o bom desempenho desta tecnologia, inclusive com tecnologias similares.

Poder-se-ia utilizar outras máquinas virtuais *java* além do JDK da **Sun Microsystems**, mas foi preferido fazer testes com esta citada, visando manter assim a característica original da tecnologia *java*.

Cabe ressaltar que os testes envolvendo acesso de dados permitem uma comparação das tecnologias (e não apenas das linguagens) utilizadas para a implementação das soluções para a geração de páginas dinâmicas. Evidentemente, não é possível dissociar as tecnologias da sua forma de acesso ao banco de dados. Procurou-se, então, utilizar as melhores formas de acesso a banco para cada tecnologia. Em relação ao ASP, a utilização do ADO é recomendada pela **Microsoft Corp.** [18] e, tanto o ASP como o ADO, são tecnologias fundamentadas e oriundas do Windows, sistema operacional também da **Microsoft Corp.**, utilizado para a realização dos testes.

Em relação ao JSP, o *driver* JDBC utilizado para acesso ao banco nos testes realizados foi o mesmo utilizado recentemente com sucesso em sistema amplamente utilizado pelas secretarias da **Prefeitura Municipal de Salvador** [23] e desenvolvido pela **ZCR Informática**, o e-Protocolo, com bom desempenho. Inclusive este *driver* é recomendado pelo **Microsoft Corp**. para acesso ao seu banco, o MS-SQL Server 2000, e possui muito boa aceitação na comunidade java. [24]

O *dbExpress* do *Delphi 7* se comportou muito bem nos testes realizados, mas ele praticamente só é utilizado pela tecnologia PPW na hora de estabelecer a conexão. Após isto, o gerenciamento e a utilização da conexão são de exclusiva responsabilidade da tecnologia, que, pelo demonstrado nos testes, apresentaram um bom desempenho.

4.1.2. Testes de Desempenho Sob Stress

Neste tipo de teste, o stress foi simulado através de várias requisições disparadas simultaneamente a um mesmo servidor por uma mesma aplicação. Começava por uma única requisição, depois subia para 10 requisições simultâneas e iguais, depois 30, 50, 80 e finalmente 100 requisições simultâneas e iguais. A medida de desempenho foi considerada como sendo o número de requisições que conseguiam ser atendidas pelo servidor, para cada quantidade lançada.

Foram desenvolvidas três aplicações web para realizar estes testes de stress, uma para cada tecnologia: ASP, JSP e PPW. Elas possuem telas de retorno (atendimento à requisição) bastantes simples e rigorosamente iguais em suas páginas HTML. As telas de retorno obedecem ao disposto na Figura 15. O atendimento à requisição era apenas retornar uma página HTML que exibisse "número 1", "número 2" até "número 100", cada número em uma linha distinta.

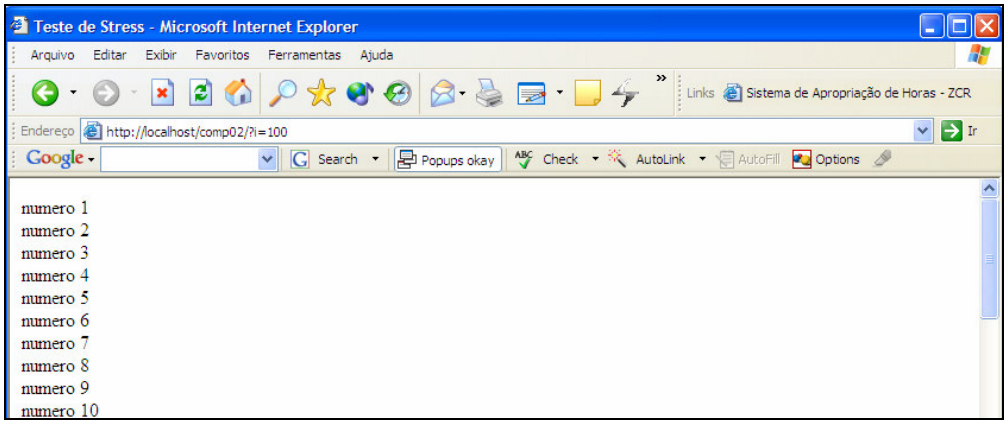

Figura 15. Amostra da tela de retorno do teste de stress

Para obter esta página de retorno, a página PPW de retorno possui o seguinte código:

```
<html> <head> <title>Teste de Stress</title> </head>
<body text="#000000" bgcolor="#FFFFFF" link="#0000FF" vlink="#000080" alink="#FF0000">
<%var v,j: integer;%>
<%
    v:=StrToInt(Parameter.value['i']);
    for j:=1 to v do
    begin
      writeln('numero ',j,'<br>');
    end;
%>
  </body></html>
```

O disparo das requisições, mais o tratamento do retorno do atendimento do servidor, foram feitos através de um programa java, utilizando as API básicas de entrada/saída, rede e utilitários. [25]

4.1.2.1. O Processo de Coleta

Para a coleta de dados foram utilizadas as mesmas máquinas e configurações, tanto servidor como cliente, para cada tecnologia. As características de cada uma, que foram as mesmas utilizadas no testes anteriores, estão descritas na Tabela 3.

Importante salientar que, em todo o processo de teste, nenhuma dessas máquinas estava usando recursos para outros programas e/ou usuários. As máquinas estavam dedicadas a estes testes.

Para a tecnologia ASP, foi utilizado o servidor IIS 5.1 da **Microsoft Corp.**. Para a tecnologia JSP foi utilizado o JBOSS 3.2.1, da **JBoss Inc.**, conjugado ao TOMCAT 4.1.24, da **Apache Software Foundation**, ambos utilizando a máquina virtual Java JDK1.4.2 da **Sun Microsystems**.

Os testes foram realizados no ambiente de rede local da empresa **ZCR Informática**, na madrugada do dia 17 de novembro de 2005, quando não havia ninguém utilizando a rede interna, mas a rede estava em pleno funcionamento e, obviamente, com os seus processos de gerenciamento e controle ativos.

Foram realizados todos os testes primeiro com a tecnologia JSP, depois com a tecnologia ASP e por último com a tecnologia PPW. Foi estabelecido que nunca seria considerada a primeira vez em cada teste e seria considerado como resultado válido o valor médio do percentual de atendimento de cinco conjuntos de requisições

enviadas. Com estas considerações foram obtidas as seguintes tabelas de coleta de dados:

Tabela 7. Servidor JSP - Requisições atendidas

JSP	Tentativas						
Num.Req.	**1**	**2**	**3**	**4**	**5**	**Valor Médio**	
1	1	1	1	1	1	1,00	100,00%
10	10	10	10	10	10	10,00	100,00%
30	30	28	25	26	25	26,80	89,33%
50	35	25	35	25	31	30,20	60,40%
80	35	35	27	37	25	31,80	39,75%
100	36	25	36	25	41	32,60	32,60%

Tabela 8. Servidor ASP - Requisições atendidas

ASP	Tentativas						
Num.Req.	**1**	**2**	**3**	**4**	**5**	**Valor Médio**	
1	1	1	1	1	1	1,00	100,00%
10	10	10	10	10	10	10,00	100,00%
30	23	21	18	24	20	21,20	70,67%
50	31	27	30	34	37	31,80	63,60%
80	30	28	45	37	33	34,60	43,25%
100	43	40	42	34	50	41,80	41,80%

Tabela 9. Servidor PPW - Requisições atendidas

PPW	Tentativas						
Num.Req.	**1**	**2**	**3**	**4**	**5**	**Valor Médio**	
1	1	1	1	1	1	1,00	100,00%
10	10	10	10	10	10	10,00	100,00%
30	30	30	30	22	30	28,40	94,67%
50	48	36	50	50	50	46,80	93,60%
80	73	62	53	59	78	65,00	81,25%
100	48	65	78	100	59	70,00	70,00%

Foram observados também os seguintes valores médios para carga de processamento da CPU. Como estes valores foram coletados empiricamente através da observação do Gerenciador de Tarefas do Windows, a sua precisão é discutível, mas dá para ter uma base do consumo dos recursos da CPU.

Tabela 10. Valor médio de carga da CPU (em % uso)

	Número de Requisições					
	1	**10**	**30**	**50**	**80**	**100**
JSP	1	1	3	5	8	13
ASP	2	5	15	18	23	35
PPW	15	65	100	100	100	100

4.1.2.2. Análise dos Resultados

Após os testes, todos os resultados foram colocados em gráfico para uma melhor análise. O impacto visual novamente mostra-se bastante útil na visão geral.

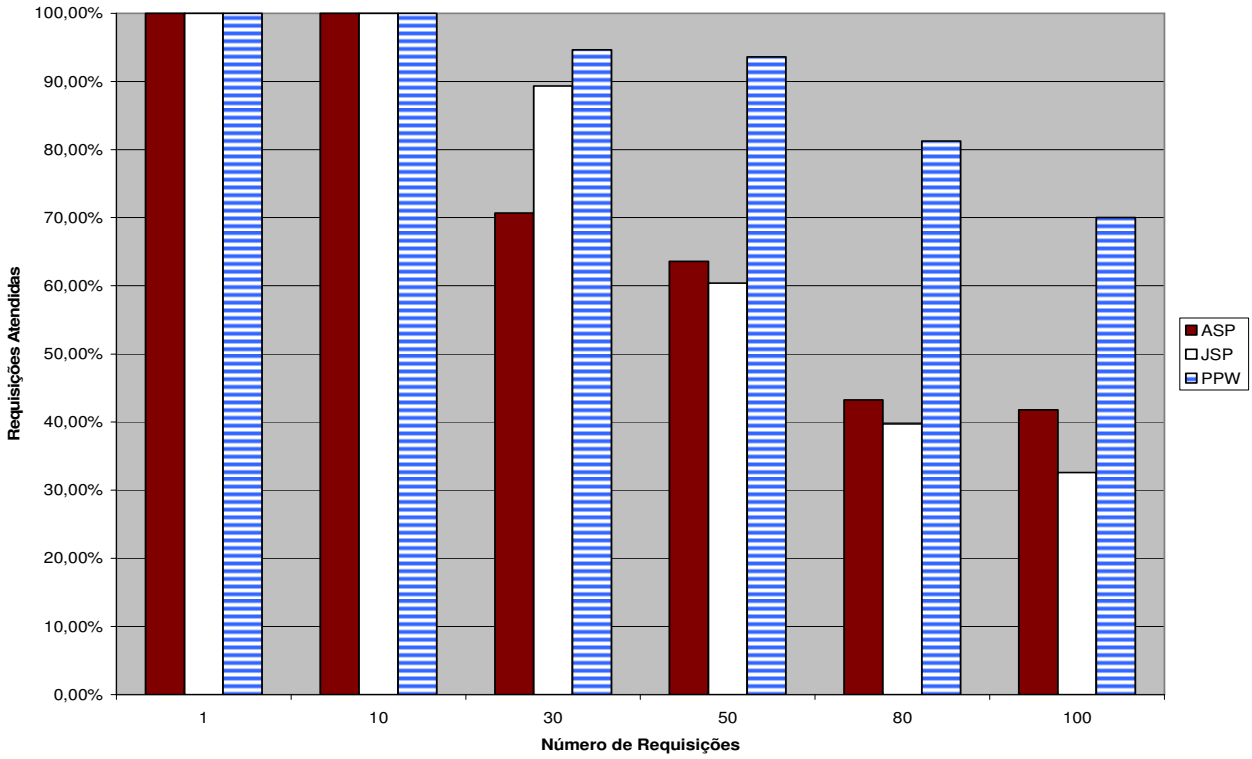

Figura 16. Gráficos comparativos do teste de stress

Como podemos ver, a tecnologia PPW sobressaiu-se em todos os números de requisições atendidas, notadamente no número alto de requisições. Conseguiu inclusive atender todas as 100 requisições em uma das tentativas.

O resultado desde teste foi, até certo ponto, surpreendente, já que não era esperado um desempenho superior do PPW, pois é sabido que a utilização excessiva ou simultânea de aplicações CGI causam um aumento linear de carga (processos ou *threads* do SO na memória) no servidor, sobrecarregando o mesmo. Este aumento

de carga pode fazer com que ele deixe de atender muitas requisições [17] [16]. Mas o fato é que, nos testes realizados, o PPW mostrou-se viável e mais confiável, em se contando apenas o número de requisições atendidas, do que o ASP ou JSP.

A sobrecarga esperada do servidor realmente aconteceu, já que após apenas 30 requisições simultâneas, a CPU do servidor PPW já mostrava utilização total (100%). O servidor JSP, que teve melhor desempenho neste quesito de carga da CPU, chegou a um máximo de apenas 13% contra 35% do ASP. Aliás o comportamento do servidor JSP foi praticamente estável e de utilização pequena da CPU em todos os testes de stress realizados. Entretanto foi o servidor que mais deixou de atender requisições, principalmente a partir de um número de 50 requisições simultâneas.

4.2. ANÁLISE COMPARATIVA COM PSP E IPSCRIPT.

O *Pascal Server Pages*, conforme citado na seção 2.6.1, é um projeto da *Nemesis Pascal* [21] que permite também a inclusão de códigos Pascal em páginas HTML através do uso de *scriptlets*.

A princípio pode-se achar o PSP com uma abordagem semelhante à tecnologia PPW, mas pelas informações obtidas no site [21], tende-se a concluir que o PSP possui uma opção diferente de projeto, não se preocupando muito em oferecer recursos extras para facilitar o trabalho de programação..

Não foram identificadas classes próprias de utilização, tal qual o PPW oferece para facilitar a programação, nem foram observados tratamentos específicos para

parâmetros POST e/ou GET, a não ser por leitura explícita a variáveis de ambiente, nem também tratamento pertinentes a sessões (abertura, variáveis, conexões, etc.). E percebe-se que para todo bloco de código é obrigatório um retorno de valor a ser exibido na página.

A abordagem do PSP deixa toda a implementação mais pesada, bem como um maior controle do ambiente, a cargo do programador.

A tecnologia PPW se mostra com uma abordagem mais completa, onde as funcionalidades e controles necessários a um ambiente web são facilitados ao programador.

Já o IPSCRIPT possui uma opção de projeto que disponibiliza uma série de facilidades ao programador, citadas na seção 2.6.2. A exceção do gerador de relatório PDF, o PPW também disponibiliza as facilidades descritas.

A vantagem do IPSCRIPT é que ele é independente, pois possui compilador próprio e rotinas de acesso próprio aos bancos de dados, ao passo que o PPW depende do compilador *Delphi 7* nesta sua primeira versão. Em contrapartida, o PPW, através deste compilador *Delphi 7*, é compatível com Pascal ANSI/ISO [4], diferente do IPSCRIPT, além de ter uma estrutura orientada ao objeto.

A abordagem da orientação a objeto é uma das grandes vantagens do PPW em relação ao IPSCRIPT que utiliza ainda o paradigma da linguagem imperativa. Com

isso, o grau de modularização e abstração de aplicativos PPW tende a ser maior, e o seu processo de reutilização mais facilitado.

Segue uma tabela comparativa dos recursos e facilidades do PPW com o PSP e IPSCRIPT.

Tabela 11. Quadro comparativo PPW x IPSCRIPT x PSP

Recurso	PPW	IPSCRIPT	PSP
Compatibilidade com Pascal ANSI/ISO	Sim	Não	Sim
Utiliza linguagem com abordagem O.O.	Sim	Não	Sim
Compilador próprio	Não	Sim	Não
Tecnologia própria de acesso a banco	Não	Sim	Não
Processamento CGI ou semelhante	Sim	Sim	Sim
Estabelece sessões HTTP	Sim	Sim	S/I
Possui variáveis de sessão	Sim	Sim	S/I
Possui variáveis de aplicação	Não	Sim	S/I
Possui tratamento de *cookies*	Sim	Sim	S/I
Tratamento de parâmetros via POST e/ou GET	Sim	Sim	Sim, mas não transparente
Utilização de procedures e functions	Sim, mas externo	Sim	S/I
Gerador próprio de relatórios em PDF	Não	Sim	Não

Como se pode notar, o PSP não traz muitas informações em sua publicação [21] e o fato do tratamento de parâmetro ser diretamente através de variáveis de ambiente, sugere que o programador pode ter mais trabalho na utilização dos seus recursos.

O IPSCRIPT está em evolução, mostra-se com muitos recursos, mas poderia ter se desenvolvido mais e ser mais divulgado e difundido, já que existe há mais de seis anos.

5. CONCLUSÃO

Este trabalho apresentou a tecnologia *Pascal Pages for Web* ou PPW, que permite um ambiente livre para desenvolvimento de suas páginas dinâmicas (no caso, páginas PPW) utilizando o *Object Pascal*, conjugado a um ambiente de execução *server-side* baseado em processamento CGI dos aplicativos PPW gerados no processo de distribuição (*deployment*) da aplicação PPW.

Ao longo do trabalho foi exposto a proposta e funcionamento da tecnologia PPW, mostrando sua facilidade de implementação através de páginas dinâmicas e linguagem *Object Pascal*. Foi mostrado também o quão eficaz se apresenta o seu método de distribuição de uma aplicação PPW e o bom desempenho do seu mecanismo de execução que atribui, de acordo com os testes realizados, um desempenho superior a tecnologias semelhantes como ASP e JSP.

Foi visto que o ambiente de execução, responsável por este bom desempenho mostrado, baseia-se na execução de programas CGI da aplicação, cuja criação é realizada durante o processo de distribuição, na proporção de um programa CGI (aplicativo PPW) para cada página PPW da aplicação. O servidor HTTP é responsável pela execução do aplicativo, relacionando-o o com a página PPW original, quando a mesma é requisitada. O *PPWController*, como mostrado, gerencia e atende solicitações desta execução, como acesso a banco de dados, manipulação de sessões e de variáveis de sessão.

Foi mostrado que o PPW pode oferecer ao desenvolvedor, grande parte das facilidades das tecnologias *server-side* para páginas dinâmicas existentes no mercado, tais como:

- Utilizar páginas dinâmicas próprias (PPW) e/ou páginas estáticas HTML;
- Receber dados e variáveis dinamicamente através de POST e GET;
- Estabelecer, controlar, validar e expirar sessões web;
- Criar, alterar, excluir e utilizar variáveis de sessão;
- Definir e remover cookies e;
- Acessar diretamente diversos bancos de dados, tais como MySQL, MS-SQL Server, Oracle, IBM DB2, Interbase, Informix;

E ainda permite ao programador:

- Utilizar qualquer ambiente de desenvolvimento para páginas HTML ou dinâmicas, como Dreamweaver, HomeSite e outros;
- Utilizar conexões inteligentes de banco de dados, estabelecendo apenas uma única vez durante todo o processamento da sessão;
- Utilizar *units* Pascal referenciadas pelas páginas PPW, facilitando a separação entre interface e código (isto é, o código não precisa estar todo na página) e ;
- Desenvolver aplicações com alto desempenho.

Nos testes realizados, pode-se concluir que a tecnologia PPW possui um bom desempenho na execução das suas aplicações, superando JSP e ASP nas condições estabelecidas, inclusive em atendimento de requisições HTTP, onde teve

um relativamente baixo índice de perdas, de acordo com testes de stress realizado. O processo de execução de aplicativo PPW para cada página PPW requisitada, mostrou-se muito eficaz.

Todos os objetivos estabelecidos neste trabalho foram atingidos e pode-se concluir que a tecnologia PPW possui todas as condições em promover sua utilização de maneira mais intensa e até profissional, visando testes mais amplos para sua evolução e amadurecimento. Para isso, a tecnologia PPW será disponibilizada inicialmente para utilização por universidades e faculdades que a desejarem, visando um processo de aprendizado acadêmico e também aperfeiçoamento do produto através de críticas e sugestões. Será também disponibilizada para algumas empresas específicas, no intuito da sua divulgação, no aperfeiçoamento do produto, e também no desenvolvimento e distribuição de aplicações PPW.

Com a disponibilização, o próximo passo para a tecnologia será estabelecer um compilador próprio e também desenvolver classes específicas do *Delphi 7* que foram utilizadas no PPW, como, por exemplo, classes de acesso a dados e de comunicação TCP cliente/servidor. Paralelamente a isso será desenvolvido um servidor PPW para processamento distribuído, prevendo com isso a possibilidade de atender a grandes quantidades de requisições HTTP sem causar uma sobrecarga à CPU do servidor, como constatado nos testes de stress. A idéia é dividir a carga de requisições em vários equipamentos, onde cada um teria o *PPWController*, através de um controle central.

Após uma maior consolidação e amadurecimento da tecnologia, poderá ser estabelecida uma ONG (organização não governamental) na WWW para desenvolvimento, melhoramentos e aperfeiçoamento constante da tecnologia em colaboração com interessados.

De acordo com o exposto na motivação para o desenvolvimento da tecnologia, pode-se presumir que o PPW, assim como Pascal ou qualquer extensão que venha dele (como o *Object Pascal* e/ou *Delphi*), pode apresentar uma curva de aprendizado menor do que as outras linguagens de mesma finalidade, o que o tornaria bastante atraente para os novos profissionais. Pode-se tornar atraente também aos profissionais atuais do Delphi, que ainda é bastante popular no Brasil [7], mas apresenta uma forte tendência de declínio no seu uso, olhando o contexto mundial[8][9]. Por estes motivos citados, a disseminação da tecnologia, principalmente em meios universitários, pode ocorrer de forma satisfatória.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] WIKIPEDIA. **Web Application**. Disponível em <http://en.wikipedia.org/wiki/Web_application>. Acesso em 24/01/2006.

[2] WIKIPEDIA. **Dynamic Web Pages**. Disponível em <http://en.wikipedia.org/wiki/Dynamic_Web_pages>. Acesso em 24/01/2006.

[3] PEREGO, C.A. **Linguagens de Programação Adotadas nos Cursos de Bacharelado em Ciência da Computação, Engenharia da Computação e Tecnologia de Informática**. Novembro de 2001. Disponível em <http://www.inf.ufrgs.br/aulas/JEduc/plteaching.html>. Acesso em 29/04/2006.

[4] WIKIPEDIA. **Pascal (linguagem de programação)**. Disponível em < http://pt.wikipedia.org/wiki/Pascal_%28linguagem_de_programa%C3%A7%C3%A3o%29 >. Acesso em 24/01/2006.

[5] BORLAND SOFTWARE CORP. **Delphi Help**. Compilado em 30/07/2002. Estados Unidos, Atlanta, 2002. Arquivo de ajuda e parte integrante do software Borland Delphi 7; 1 CD-ROM.

[6] SEBESTA, R. **Conceitos de Linguagem de Programação**. Tradução José Carlos Barbosa dos Santos. 5ª edição. Porto Alegre: Bookman, 2003. Tradução de *Concepts of Programming Language, 5th edition*.

[7] CESAR, R. **Java X .Net: Disputa Acirrada no Mercado Nacional**. Computerworld, Edição 387. Disponível em <http://computerworld.uol.com.br/AdPortalv5/adCmsDocumentShow.aspx?GUID=F0740D1E-6815-4620-82F0-50220D32557D&ChannelID=21>. Acesso em 25/01/2006.

[8] SKILL MARKET. **A daily look at in-demand tech skills**, Disponível em <http://mshiltonj.com/sm/>. Acesso em 25/01/2006

[9] TIOBE PROGRAMMING COMMUNITY INDEX. **January Headline: Java wins Programming Language of 2006 Award!** Disponível em <http://www.tiobe.com/tpci.htm>.Acesso em 25/01/2006

[10] CATAMBAY, B. **Pascal Central**. Disponível em <http://www.pascal-central.com>. Acesso em 09/05/2004.

[11] WIKIPEDIA. **Turbo Pascal**. Disponível em <http://pt.wikipedia.org/wiki/Turbo_Pascal>. Acesso em 13/10/2005.

[12] WIKIPEDIA. **Object Pascal**. Disponível em <http://pt.wikipedia.org/wiki/Object_Pascal >. Acesso em 24/01/2006.

[13] FREE PASCAL TEAM. **Free Pascal**. Disponível em <http://www.freepascal.org>. Acesso em 29/11/2004.

[14] INCITS. **InterNational Committee for Information Technology Standards**. Disponível em < http://www.incits.org>. Acesso em 24/01/206.

[15] WIKIPEDIA. **Common Gateway Interface**. Disponível em <http://en.wikipedia.org/wiki/Common_Gateway_Interface>. Acesso em 24/01/2006.

[16] NCSA CGI TEAM. **The Common Gateway Interface**. Disponível em <http://hoohoo.ncsa.uiuc.edu/cgi/>. Acesso em 05/09/2004.

[17] OLINUX.COM.BR. **Introdução ao desenvolvimento web**. Disponível em <http://olinux.uol.com.br/artigos/292/print_preview.html >. Acesso em 20/01/2006.

[18] MICROSOFT CORP.. **Active Server Pages**. Disponível em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/activeservpages.asp>. Acesso em 10/02/2005.

[19] SUN DEVELOPER NETWORK (SDN). **JavaServer Pages Technology.** Disponível em <http://java.sun.com/products/jsp/ >. Acesso em 18/11/2005

[20] MUKHI, V.; MUKHI, S.; KOTECHA, N. **Java Servlets JSP**. Tradução Ariovaldo Griesi. Revisão Técnica Álvaro R. Antunes. São Paulo: Makron Books, 2002. Tradução de *Java Servlets JSP*.

[21] NEMESIS PASCAL PROJECT. **Nemesis Pascal Project – An open source pascal plattform**. Disponível em <http://npascal.sourceforge.net>. Acesso em 21/03/2005.

[22] IPSCRIPT. **Internet Pascal Scripting Language**. Disponível em <http://www.ipscript.com>. Acesso em 25/02/2005.

[23] SECRETARIA MUNICIPAL DE ADMINISTRAÇÃO. **SEAD implanta o e-Protocolo**. Disponível em <http://www.sead.salvador.ba.gov.br/sead2/integra_noticia.asp?idt_noticia=4247>. Acesso em 26/01/2006.

[24] MICROSOFT CORP. **SQL Server 2000 Driver for JDBC Service Pack 3**. Disponível em <http://www.microsoft.com/downloads/details.aspx?familyid=07287b11-0502-461a-b138-2aa54bfdc03a&displaylang=en>. Acesso em 26/01/2006.

[25] FLANAGAN, D. **Java: O Guia Essencial**. Tradução Kátia Roque. Rio de Janeiro: Campus, 2002. Tradução de *Java in a Nutshell*.

[26] BUCKNALL, J. **Algoritmos e Estruturas de Dados com Delphi**. São Paulo: Berkeley Brasil, 2002.

[27] TOTH, PAUL. **Programmes CGI avec Delphi et Kylix**. Disponível em <http://tothpaul.free.fr/cgi.htm>. Acesso em 09/05/2004.

[28] GNU PASCAL TEAM. **GNU Pascal**. Disponível em <http://www.gnu-pascal.de>. Acesso em 25/01/2006.

[29] NETSCAPE. **Persistent Client State – HTTP Cookies – Preliminary Specification**. Disponível em <http://www.netscape.com/newsref/std/cookie_spec.html>. Acesso em 05/12/2004.

[30] BUCZEK, G. **ASP Guia do Programador**. Tradução Equipe Market Books. São Paulo: Market Books, 2000. Tradução de *ASP Developer's Guide*.

[31] NEGRINO, T.; SMITH, D. **JavaScript para a World Wide Web**. Tradução Adriana Kramer. Rio de Janeiro: Campus, 2001. Tradução de *JavaScript for the World Wide Web*.

GLOSSÁRIO

Ambiente web – Ambiente de sistemas que normalmente utilizam páginas HTML como interface com usuário. Estas páginas são visualizadas através do *browser*.

Browser – O mesmo que navegador HTTP.

Contêiner web – Conjunto de componentes de um servidor web totalmente voltados para tratamento de páginas web trabalhando sempre do lado do servidor.

Cookie – Pequenos arquivos de uma aplicação web que são gravados na máquina cliente, contendo informações da aplicação sobre o cliente. Muito utilizados para armazenar informações que poderão estar facilmente disponíveis quando o cliente retornar à aplicação.

Distribuição ou deployment – Normalmente referindo-se a uma aplicação; prepara a aplicação para ser executada no servidor de aplicação, tornando-a disponível ao servidor HTTP.

GET – Método de submissão de páginas aos servidores HTTP. Os parâmetros que são submetidos por este método estão explícitos na chamada da página, após o símbolo "?".

HTTP – *Hyper Text Transport Protocol*. É o protocolo de comunicação que facilita o transporte entre páginas HTML e servidores web.

HTML – *Hyper Text Markup Language*. É a linguagem de marcação que se utiliza para exibir formatar páginas na web, através da interpretação das suas tags pelos navegadores. Com essa linguagem, podem ser definidas páginas que contenham informações nas mais variadas formas: texto, som, imagens e animação.

IDE – *Integrated Development Environment*. É um ambiente integrado de desenvolvimento, onde simultaneamente permite editar, compilar, depurar e construir aplicações, além de outros recursos.

Multi-thread – O mesmo que multi-tarefa. Chamamos uma aplicação *multi-thread* quando ela pode realizar tarefas simultaneamente dentro da mesma aplicação.

Navegador ou Navegador HTTP – O navegador é uma ferramenta normalmente utilizada para visualizar páginas HTML em sites da web. Os mais comuns são o *Internet Explorer*, *Mozzila*, *Netscape* e *Firefox*.

Páginas dinâmicas – Páginas da web que possuem conteúdo HTML variável de acordo com a situação.

POST – Método de submissão de páginas aos servidores HTTP. Os parâmetros que são submetidos por este método, normalmente campos de formulários, são passados internamente para o servidor.

Requisição – Uma solicitação de página HTML a um servidor HTTP. Normalmente a requisição é feita quando é escrito e submetido, no topo do *browser*, o endereço da página precedido pro http://, que é a chamada ao protocolo HTTP.

Scriptlets – Símbolos `<%` e `%>` que demarcam linhas de códigos em páginas dinâmicas.

Scripts client-side – Códigos que normalmente estão dentro de páginas HTML e que são processados no *browser*, dentro da máquina cliente que requisitou a página.

Server-side – Quando o processamento que ocorre dentro do servidor web. Muito utilizado em aplicações web. para páginas dinâmicas.

Servidor de aplicação – Servidor que aborda a tecnologia *server-side* adotada para as páginas dinâmicas. Trabalha normalmente conjugado com um servidor HTTP.

Servidor HTTP – Servidor que armazena, gerencia e direciona as páginas HTML ou dinâmicas, como ASP, PHP e JSP.

Servidor web – Servidor que disponibiliza o serviço WWW (web) sendo normalmente composto por servidor HTTP e servidor de aplicação.

Sessão – Estabelecimento de um canal de comunicação entre o cliente (requisitante) e o servidor.

Sessão web – Sessão estabelecida por requisição HTTP a um servidor web.

SO – Sistema Operacional.

Tag – Elemento texto, base de marcação do HTML. Informa ao *browser* o que ele deve fazer naquele trecho e retrata a formatação da página no trecho em que se encontra.

Tipada – Referindo-se a linguagem de programação, significa uma linguagem cujos valores e objetos têm tipos bem definidos e não sofrem coerções.

Units – São arquivos que contém códigos fontes de uma linguagem.

XML – *eXtensible Markup Language*. É uma linguagem de marcação utilizada para necessidades especiais, capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da Internet.

Web – Sinônimo mais conhecido do serviço *World-Wide-Web* (WWW) da internet. É a interface gráfica da internet que torna os serviços disponíveis totalmente transparentes para o usuário e ainda possibilita a manipulação multimídia da informação. É disponibilizado na rede através do protocolo HTTP e das páginas HTML, através de um servidor web.

APÊNDICES

APÊNDICE A – Manual do Programador PPW.

Manual de referencia para o programador PPW. Possui a finalidade de orientar o programador PPW na utilização completa e suficiente dos recursos da tecnologia, bem como trazer um pouco mais do seu entendimento técnico.