Modelagem de Hipermídia
Prof.Giangiacomo Ponzo Neto
Parte Teórica II
(obs.: pode conter trechos da wikipedia)

Hipermídia na Internet – do bit às Linguagens de Programação Web

No Princípio Era o bit

Em meados da década de 1940, surgem as primeiras máquinas que podiam ser chamadas de computadores eletrônicos, como o Mark I na Universidade de Harvard (com a IBM), o inglês Colossus (que ajudou a quebrar o código de comunicações alemão da 2ª guerra mundial) e o ENIAC (*Electronic Numeral Integrator and Calculator*) do exército americano. Essas máquinas eram imensas, pesavam toneladas, continham milhares de válvulas, consumiam um absurdo de energia e armazenavam apenas algumas dezenas de bits. Não eram mais do que calculadoras gigantescas, mas faziam vários cálculos por segundo, algo incrível para a época e que revolucionou a tecnologia militar e científica. Para termos uma idéia do quanto se ganhou em tempo, alguns cálculos de balística que levavam um dia para serem resolvidos de forma manual semi-automatizada (com réguas de cálculo e coisas do gênero) passaram a ser feitos em menos de um minuto! Alguns desses sistemas pioneiros ficaram em uso por mais de uma década.

Como era feita a programação dessas máquinas? Através de vários interruptores, que podiam ser ligados ou desligados. Ou seja: cada interruptor representava um bit (desligado=0, ligado=1). Assim se descreviam os números e os "comandos" que formavam a entrada de dados para o computador. Mídia? Nenhuma, só números... Pura linguagem de máquina, bits, bytes e nada mais, formando a chamada linguagem de primeira geração.

A linguagem de segunda geração - o Assembly ("assembler" é o programa que traduz os mnemônicos para bytes) surge apenas alguns anos depois, já na década de 1950.

Um exemplo de trecho em Assembly (atual da Intel, não o pré-histórico...) pode ser visto abaixo:

```
mov al, FFh
```

em binário: `10110000 11111111`

em Hexadecimal: `B0 FF`

O montador assembler traduz o trecho para a máquina. Só para mencionar, esse código guarda o valor FF (ou 255 na base 10) no registrador chamado "al" do processador (um Pentium, por exemplo). Cada processador pode ter sua linguagem própria, mas há muito tempo os mnemônicos básicos dentro de uma mesma linha (386, 486, Pentium...) são os mesmos.

Ao ENIAC seguiram-se muitos outros computadores gigantes (EDVAC, ORDVAC, SEAC, UNIVAC) que, aos poucos, ficavam cada vez menos gigantes e mais rápidos. Cerca de uma década após os pioneiros, os HDs começaram a surgir (o primeiro que podia ser usado por um computador IBM tinha 5 MB de capacidade e pesava cerca de 1 tonelada). Um computador já pesava um décimo do peso do ENIAC e o custo era um pouco mais "acessível" (algumas centenas de milhares de dólares). Nesta época algumas grandes empresas ou instituições já possuíam condições para tal.

O Apogeu dos Mainframes

Em 1954 surge a primeira linguagem de terceira geração bem sucedida, o Fortran (*FORmula TRANslation*, de cunho fortemente matemático e ainda em uso na área matemática e científica). Em 1958 nasce o Algol (*ALGOrithmic Language*, para aplicações científicas, a primeira linguagem a lidar com programação estruturada) e no ano seguinte o COBOL (*Commom Business Oriented Language*, destinada a operações comerciais e ainda em uso – o que foi responsável por boa parte dos problemas do chamado "*bug* do milênio"). No início da década de 1960, os transistores substituem as válvulas, o que permite que empresas como IBM, Burroughs e outras lancem máquinas mais rápidas, menores e mais baratas. Em 1965 a IBM cria o System/360, uma máquina de enorme sucesso a partir da qual se baseiam todos os

mainframes seguintes (computadores de grande porte centrais normalmente ligados a vários terminais "burros") que dominaram o mercado por décadas e, com a arquitetura tendo ganho enorme progresso, até hoje estão em atividade em determinados segmentos. Sempre foram caros, mas não mais gigantescos nem inacessíveis. Várias instituições de ensino, inclusive aqui no Brasil, possuíam *mainframes*. Eu mesmo usei um Burroughs na UFRJ com cartões perfurados em Fortran, nos anos 80. Na UFJF havia um IBM 1130 que ficou operacional por um bom tempo – hoje está no museu da Faculdade de Engenharia. A mídia? Basicamente texto; coisas como som, desenho, imagem eram praticamente inexistentes até...

O Surgimento dos Micros

Após a criação das linguagens Basic (1964), Pascal (1970) e C (1971), as vendas prosperavam e os computadores começavam a ter preços cada vez mais acessíveis. Em meados da década de 70, os aficionados em eletrônica começaram a criar pequenos *kits* de máquinas com algum poder de processamento, impulsionados pelos recém lançados microprocessadores de 8 bits da Intel: o 8008 e, melhor ainda, o 8080. O primeiro micro-computador a ser um projeto relativamente comercial e vendável foi o Altair de Ed Roberts em 1975. A Microsoft de Bill Gates e Paul Allen surgiu justamente nesse período tentando fornecer uma versão de Basic para o Altair. Apesar do pioneirismo, o Altair foi logo suplantado pelo impressionante Apple][(é... o II era escrito assim mesmo) da Apple, um projeto irretocável de Steve Wosniak sobre o microprocessador Motorola 6502 e "marketado" por Steve Jobs. A partir daí, outras empresas (como a Sinclair na Inglaterra) começaram a desenvolver seus próprios micros, baseados em microprocessadores Intel, Motorola ou Zilog (esta última criou o Z80, sobre o qual se basearia o sistema operacional CP/M de 8 bits, pré-DOS, de grande sucesso na micro-informática de então). A febre chegou ao Brasil no início dos anos 80, em plena reserva de mercado, com os *clones*, máquinas que funcionavam da mesma forma que os micros da Sinclair (ZX80, ZX81), Apple (Apple][), Radio Shack (TRS-80) e outras. A esta altura, o Basic dominava estes aparelhos como linguagem. Após desdenhar dos micros por algum tempo, a IBM rendeu-se e, correndo atrás do prejuízo, lançou em 1981 o primeiro IBM-PC (*Personal Computer*) de 16 bits, com o sistema operacional DOS da Microsoft.

Enquanto os usuários de *mainframes* tinham apenas monitor e teclado, sendo todo o processamento realizado no servidor e dividido entre eles, o micro-computador era dirigido a usuários individuais (por isso mesmo foi chamado "computador pessoal"). Aí começa a revolução multimídia. Aos poucos, os micros vão ganhando alta resolução gráfica, cores, capacidade de emitir sons e também maior memória, velocidade de processamento e armazenamento (interna e externa), pois os novos recursos multimídia demandavam um poder que não havia anteriormente. Em meados da década de 90, o "quente" era ter um micro com "kit multimídia" (CD-player, microfone, caixinhas e placa de som).

Interface Gráfica e Redes

A grande evolução das máquinas gerou novas demandas por interfaces visuais. Surgiram os primeiros ambientes operacionais gráficos. O Apple Lisa, lançado em 1983, teve vida curta mas, no ano seguinte, foi substituído pelo incrível Macintosh, que revelou ao mundo todos os elementos gráficos que hoje conhecemos. Em 1985, a Microsoft criou o primeiro Windows. As primeiras versões foram pouco usadas; o sucesso só veio com a versão 3.0, de 1990. Uma versão "multimídia" foi criada para ser vendida integrada aos "*kits* multimídia". O Windows 3.0 vendeu cerca de 10 milhões de cópias nos 2 anos anteriores ao lançamento da versão 3.1, de ainda maior sucesso. Pouco tempo depois e já de olho nas redes que começavam a aparecer, surgiu o Windows for Workgroups 3.11. Específico para uso com processadores tipo 386 (ou superiores), já que usava as novas instruções de máquina que vinham com este modelo de processador, o 3.11 dominou o mercado graças aos programas gráficos (Word, Excel, PageMaker, etc.) até meados da década de 90, muito embora os programas comerciais ainda fossem em grande parte feitos para DOS (utilizando Basic, Pascal ou a dupla dBase/Clipper – um misto de linguagem com banco de dados rudimentar de grande sucesso de meados da década de 80 até meados da década de 90).

Correndo por fora, após anos de relacionamento com a Microsoft, a IBM tentou se desvencilhar do "monstro" que havia ajudado a criar e lançou o seu próprio SO gráfico: o OS/2, que muitos julgavam ser melhor que o Windows, mas, como sabemos, não conseguiu o intento de rivalizar com a outrora parceira Microsoft. A Apple também continuou com seus SOs próprios, amados pelos usuários fiéis, mas sempre restritos à plataforma própria. O Linux, SO *free* sobre o Unix (já consagrado, desde a década de 1970, em plataforma não micro) surge em 1991. De início, não assusta a concorrência, mas aos poucos ganha terreno, principalmente no "lado servidor".

Por falar em servidores, em 1992 surge o Windows NT, com a proposta clara de ser um SO específico para servidores. Os micros já tinham "crescido" o suficiente para destronarem os *mainframes*, mesmo em grandes corporações. Os servidores agora podiam ter sistemas operacionais gráficos e serem ligados não a terminais "burros", como

normalmente os *mainframes* o eram, mas a computadores mais simples, porém ainda assim, com poder de processamento suficiente para dividir tarefas com o servidor.

Em 1995 surge o Windows 95, que finalmente consegue usar direito o poder dos processadores de 32 bits. Suas grandes melhorias em relação ao Win3.x possibilitam que linguagens com IDEs gráficos passem a ser a sensação para as aplicações *desktop*. Ferramentas como Visual Basic, Delphi, PowerBuilder e outras passam a ser amplamente usadas, enquanto ambientes não gráficos (Turbo C, Turbo Pascal e Clipper) começam a sumir do mercado. O modelo de desenvolvimento cliente-servidor baseado em interface gráfica atinge seu auge e dura até quase o fim do milênio, quando o desenvolvimento sobre *web* começa a destroná-lo.

Hipermídia para Todos: a Internet

Até o lançamento do Windows 95, a Microsoft parecia não acreditar muito na Internet (ou preferia não acreditar e tentar criar quase que uma "internet" própria: a MSN – depois desistiu disso e MSN virou apenas um portal dentro da Internet). A grande rede ia se estabelecendo aos poucos, com os *browsers* Mosaic e, logo depois, o Netscape. Este último dominou o mercado dos navegadores até quase o fim do milênio. Mas, em 1998, a Microsoft resolveu adotar uma política agressiva de mercado, atrelando o Internet Explorer ao Windows 98, como parte do sistema. Isso criou uma grande polêmica que chegou aos tribunais, mas o "estrago" já estava feito. Após o episódio, que ficou conhecido como "a guerra dos browsers", o IE passou a ser o navegador mais usado do planeta após 1998 e assim continua até hoje. Ironicamente, dos "restos mortais" da Netscape (Mozilla) surge o Firefox, que hoje é o principal rival do IE.

E assim chegamos à hipermídia de fato para uso geral e não restrito à comunidade acadêmico-científica. A web, desde o final do milênio, se estabeleceu como um padrão não só para exibição de páginas de hipertexto e hipermídia, mas também como ambiente para execução de programas. Surgiram linguagens de *script* para serem embutidas nas páginas, programas executáveis conectados, linguagens "*server-side*", ferramentas de conteúdo, *frameworks*, etc. Tudo isso tentando juntar o que de melhor há na Internet (facilidade de acesso, cliente "magro", ausência de instalação, etc.), na programação e no *design* de interfaces gráficas.

Desenvolvimento *Web*

Uma aplicação *web* designa, de forma geral, um sistema projetado para utilização através de um navegador, na Internet ou em redes privadas (Intranet), ou seja: um conjunto de programas que é chamado/executado em um servidor e cujo resultado é distribuído pela rede ao computador cliente. O desenvolvimento *web* simplifica a atualização e manutenção, já que o código-fonte fica em um único local, de onde ele é acessado pelos diferentes usuários evitando a instalação de programas no lado cliente, que podem ser máquinas mais simples, além de universalizar o acesso ao sistema.

No desenvolvimento *web* pode-se separar o que é *design* e o que não é. A parte de *design* fica a cargo de *web-designers* e o desenvolvimento propriamente dito com o pessoal de TI, que escreve a codificação. Isso pode variar desde uma simples página HTML estática até complexas aplicações institucionais ou de comércio eletrônico. Em grandes organizações, o time de desenvolvimento pode chegar a centenas de pessoas, enquanto pequenas empresas podem usar um simples *webmaster* para fazer tudo ou mesmo terceirizar o serviço.

Desde meados da década de 90, o desenvolvimento *web* tem sido uma das indústrias que mais crescem no mundo e tende a crescer mais ainda, impulsionada principalmente pelo comércio eletrônico e a automação de tarefas de escritório. O custo do desenvolvimento, ao contrário, tem caído consideravelmente, principalmente devido ao *hardware* mais barato e ao *software* gratuito e/ou de código aberto. Um exemplo de custo "zero" (em aquisição de *software*) é o popular conjunto LAMP (Linux, Apache, MySQL, PHP). Outro fator que contribuiu para o aumento do interesse na área foi o surgimento de vários softwares como o Dreamweaver ou o Frontpage, que facilitaram a criação de páginas, bem como os CMS (*Content Management System*) – sistemas gerenciadores de conteúdo (Drupal, Joomla, WordPress, etc.) e os *frameworks* de desenvolvimento, notadamente os de Java e .NET. Já podemos usar softwares *on-line* que normalmente seriam *desktop*, como editores de texto, planilhas, fotográficos, etc. – uma clara tendência, mas que ainda (em minha opinião) não consegue chegar ao requinte estético e técnico possível em aplicações *rich client* feitas em ferramentas tipicamente *desktop*.

Podemos dividir o desenvolvimento *web* em duas áreas principais:

- Tecnologia *Client Side* (HTML/XHTML, CSS, Javascript, AJAX, Applets Java, controles ActiveX, Flash, etc.)

? Tecnologia *Server Side* (JSP/Servlets, PHP, ASP, .NET, Scripts CGI, Perl, Python, Ruby, etc.)

Além disso, o desenvolvedor *web* pode/deve ter algum conhecimento em: *design*, no *hardware* (servidores, roteadores, *hubs*, cabeamento, etc.), no *software* (servidores *web* como Apache ou IIS, protocolos, etc.) e na segurança (política de *backup*, antivírus, *firewall*, certificados digitais SSL, etc.), caso estas tarefas fiquem a seu encargo. Pensando bem, talvez seja muita coisa para uma pessoa só...

Especificamente quanto à codificação *server side* (que é a parte mais relacionada à disciplina), vamos ver o que temos disponível em termos de linguagens:

PERL

PERL (*Practical Extracting and Report Language*) foi criada em 1987 por Larry Wall como uma linguagem de programação de uso geral voltada originalmente para manipulação de textos, sendo posteriormente usada de forma pioneira (por volta de 1994) na programação web, na forma de scripts PERL-CGI[1]. A linguagem foi feita para ser prática e simples em detrimento da elegância e regras fortes. Implementa programação procedural ou orientada a objetos, tem um poderoso suporte incluso na linguagem para processamento de textos e uma das maiores coleções de *add-ons*, o CPAN, um repositório onde se encontram módulos, classes, *scripts*, *frameworks* e quase tudo já desenvolvido para a linguagem. É mais comum em sistemas Unix, mas tem versões para vários outros SOs. PERL se enquadra na definição de software livre.

A sintaxe de PERL se assemelha a de outras linguagens de terceira geração. Além disso, foi bastante influenciado por linguagens de *shell script*, como o *bourne shell* do Unix. Utilizando um *mix* de conceitos de outras linguagens (listas de Lisp, tabelas hash de AWK, expressões regulares de Sed, etc.) facilita a interpretação e tratamento de textos e dados em geral. PERL possui gerenciamento de memória automático e tipificação dinâmica. Sua interface de integração com base de dados (DBI) suporta diversos bancos. PERL tem módulos para trabalhar com HTML, XML, e outras linguagens de marcação de texto e seu interpretador pode ser embutido em outros sistemas.

PERL é uma das linguagens mais populares de programação *web*, devido a sua capacidade de manipulação de textos e rápido ciclo de desenvolvimento. O módulo Perl CGI.pm, parte da distribuição padrão, faz com que a manipulação de formulários HTML seja muito simples, inclusive para transações de comércio eletrônico. Ultimamente, tem perdido terreno para outras linguagens e ambientes *web*, mas ainda tem adeptos aguerridos. O conceito de resolver problemas de várias formas possíveis, aliado a regras por vezes fracas (como no caso da tipificação de variáveis) fez com que PERL fosse criticada por boa parte da comunidade de programação, sendo depreciada como "write-only language". Até o autor concorda que PERL é meio "bagunçada", segundo ele tal qual a realidade que nos cerca.

Para saber mais: www.perl.org.br/

Python

Python é uma linguagem de programação de alto nível, interpretada, interativa, orientada a objetos e de tipagem dinâmica e forte, lançada por Guido van Rossum em 1991, embora a versão oficial 1.0 seja de 1994. Atualmente possui um modelo de desenvolvimento comunitário e aberto. A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional, priorizando a legibilidade do código sobre a velocidade. Combina uma sintaxe concisa e clara com recursos poderosos de sua biblioteca padrão e por módulos e *frameworks* desenvolvidos por terceiros. O nome Python teve a sua origem no grupo humorístico britânico Monty Python.

Atualmente a linguagem é usada em diversas áreas, como servidores de aplicação e computação gráfica. Está disponível como linguagem script em aplicações como OpenOffice (Python UNO Bridge) e pode ser utilizada em *stored procedures* no PostgreSQL (PL/Python).

[1] CGI (*Common Gateway Interface)* foi a primeira forma de gerar páginas *web* dinâmicas. Scripts CGI geram programas executáveis residentes no servidor que recebem uma solicitação via *browser* e geram uma página em resposta. Embora a linguagem tipicamente associada ao CGI seja o PERL, ele foi concebido para ser independente de linguagem. ASP.NET, PHP ou mesmo o Delphi podem utilizá-lo, embora o seu uso esteja em declínio.

Python foi desenvolvido para ser uma linguagem de fácil leitura, com um visual agradável, frequentemente usando palavras e não pontuações como em outras linguagens. Para a separação de blocos de código, a linguagem usa espaços em branco e indentação ao invés de delimitadores visuais como chaves ou palavras (begin, end). Diferente de linguagens com delimitadores visuais de blocos, em Python a indentação é obrigatória. O aumento da indentação indica o início de um novo bloco, que termina na diminuição da indentação. Se usarmos um editor de texto comum é muito fácil existir erros de indentação, então o recomendado é configurar o editor conforme a análise léxica do Python ou utilizar uma IDE. Todas as IDE que suportam a linguagem fazem indentação automática.

Python também usa "bytecodes", mas de alto nível, ou seja, é mais legível aos seres humanos que o código dos *.class de Java. O código fonte é traduzido pelo interpretador para o formato *bytecode*, que é multiplataforma e pode ser executado e distribuído sem o fonte original.

Python possui uma grande biblioteca padrão, geralmente citada como um dos maiores trunfos da linguagem, fornecendo ferramentas para diversas tarefas como escrever aplicações para a Internet, contando com diversos formatos e protocolos como MIME e HTTP. Também há módulos para criar interfaces gráficas, conectar em bancos de dados relacionais e manipular expressões regulares.

Existem ferramentas Python para diversas plataformas, incluindo celulares e consoles de jogos, bem como integrações: Jython para a Plataforma Java e IronPython para .NET.

Alguns dos projetos que utilizam Python incluem o YouTube e o cliente original do BitTorrent. Para diversos sistemas operacionais a linguagem já é um componente padrão, estando disponível em diversas distribuições Linux. O Red Hat Linux usa Python para instalação, configuração e gerenciamento de pacotes.

Para saber mais: www.pythonbrasil.com.br

Ruby

Ruby é uma linguagem interpretada, com tipagem dinâmica e forte, orientada a objetos (tudo é objeto – até números...) com vastas semelhanças com Perl, SmallTalk e Python.

Projetada tanto para a programação em grande escala quanto para codificação rápida, tem um suporte a orientação a objetos simples e prático. A linguagem de script foi criada no Japão por Yukihiro Matsumoto (mais conhecido como "Matz"), que aproveitou as melhores idéias das outras linguagens da época (ficou pronta no fim de 1994) e ainda "comanda" o desenvolvimento da linguagem. Ele queria uma linguagem mais poderosa que PERL e mais orientada a objeto que Python. O projeto (de código aberto) sobrevive de doações feitas pelos usuários satisfeitos e por empresas que conseguiram aumentar sua produtividade utilizando Ruby.

A linguagem possui vastos repositórios de bibliotecas disponíveis em sites como Ruby Forge e Ruby Application Archive (RAA).

Ruby se tornou reconhecida no ocidente depois que Dave Thomas escreveu um dos mais completos livros sobre a linguagem, o "Programming Ruby". Ultimamente, devido a grande exposição de um *framework* web feito em Ruby, o Ruby on Rails, a linguagem tem sido foco da mídia especializada justamente pela sua praticidade, que é um dos conceitos básicos da linguagem.

Outras características de Ruby são: a pouca necessidade de pontuações; as variáveis são objetos, onde os "tipos primitivos" (inteiro, real, etc.) são classes (herança de SmallTalk); tipagem dinâmica, mas forte (as classes podem ser alteradas dinamicamente, em tempo de execução); e os "mixins" – que simulam herança múltipla sem alguns de seus problemas.

Ruby está disponível para diversas plataformas, além de ser executável em cima da máquina virtual do Java (através do JRuby).

Para saber mais: www.rubyonbr.org

Java/JSP

Já conhecemos Java e já vimos JSP nesta disciplina, mas vamos falar um pouco da sua história. Java foi concebida como uma linguagem de programação orientada a objeto na década de 1990 por uma equipe de programadores chefiada por James Gosling da Sun Microsystems. Em 1991 foi iniciado o *Green Project*, cujo objetivo não era a criação de uma nova linguagem de programação, mas antecipar e planejar a "próxima onda" do mundo digital. Eles acreditavam que, em algum tempo, haveria uma convergência dos computadores com os equipamentos e eletrodomésticos comumente usados pelas pessoas no seu dia-a-dia. Mas o projeto acabou gerando uma linguagem que foi inicialmente batizada de "Oak".

O estouro da Internet aconteceu e Gosling foi incumbido de adaptar o Oak para a Internet. Em janeiro de 1995 foi lançada uma nova versão que foi finalmente chamada de Java. Aplicações poderiam ser executadas dentro dos *browsers* nos Applets Java e tudo seria disponibilizado pela Internet instantaneamente. A velocidade dos acontecimentos seguintes foi assustadora, o número de usuários cresceu rapidamente e logo grandes fornecedores de tecnologia, como a IBM, anunciaram suporte para a tecnologia Java. Desde seu lançamento, em maio de 1995, a plataforma Java foi adotada mais rapidamente do que qualquer outra linguagem de programação na história da computação. Neste mesmo ano, em conjunto com a Netscape, a Sun licenciou o uso do nome "Java" para agregar funcionalidade de script junto ao lado cliente na linguagem Javascript (que também já conhecemos), embora não seja parte estrita da especificação Java. Em 2003, atingiu a marca de 4 milhões de desenvolvedores em todo mundo, continuou crescendo e hoje é uma referência no mercado de desenvolvimento de *software*. Java tornou-se popular pelo seu uso na Internet e hoje possui seu ambiente de execução presente em web browsers, mainframes, SOs, celulares, palmtops e cartões inteligentes, entre outros. A partir de 2007, a Sun abriu quase todo o código do Java.

Orientação a objeto, tradução intermediária para bytecodes, portabilidade, recursos de rede, segurança, sintaxe similar a C, facilidade de internacionalização e coletor de lixo automático são algumas de suas principais características. Java possui vários *frameworks* e IDEs que ajudam no desenvolvimento, além de vir em 3 edições: SE (*Standard Edition* – padrão para aplicações desktop, *client-server* convencional, texto ou gráfico), ME (*Micro Edition* – para dispositivos móveis de pouca capacidade computacional, como celulares) e EE (*Enterprise Edition* – para aplicações de grande porte, distribuídas, com tecnologia *web*, etc.). Esta última edição (EE) é que contém a especificação de Servlets e JSP que são as tecnologias Java para uso específico no lado servidor de uma aplicação *web* e foram vistas no início do curso.

Para saber mais: www.java.com/pt_BR

PHP

PHP (que hoje significa *PHP: Hypertext Preprocessor*) é uma linguagem interpretada, de código livre e muito utilizada para gerar conteúdo dinâmico na web. A linguagem surgiu por volta de 1995, como um pacote de programas CGI criados por Rasmus Lerdof, com o nome original de *Personal Home Page Tools*, para substituir um conjunto de scripts PERL que ele usava no desenvolvimento de sua página pessoal. Em 1997 foi lançado o novo pacote da linguagem com o nome de PHP/FI, trazendo a ferramenta *Forms Interpreter*, um interpretador de comandos SQL. Mais tarde, Zeev Suraski desenvolveu o analisador do PHP 3 que contava com um primeiro recurso (rudimentar) de orientação a objeto. Pouco depois, Zeev e Andi Gutmans, escreveram o PHP 4, dando mais poder à linguagem e maior número de recursos de orientação a objeto, embora ainda não de forma completa, o que veio a ser resolvido no PHP 5.

PHP é ideal para instalação e uso em servidores *web*. É parecida em sintaxe e mesmo funções, com C e C++. Pode ser embutida no código HTML, desde que o servidor tenha o interpretador instalado. Existem versões disponíveis para vários SOs. Simplicidade e facilidade de aprendizado, inclusive no uso com bancos de dados, são os apelos de PHP. Um dos casos de sucesso é a Wikipedia, que funciona sobre um software inteiramente escrito em PHP, usando bases de dados MySQL. Embora o PHP esteja muito ligado à *web*, existem iniciativas para utilizá-lo como linguagem de programação desktop; a mais notável é a PHP-GTK.

Algumas características de PHP (como a tipagem fraca e a fragilidade da orientação a objeto em versões anteriores) levaram alguns críticos a menosprezarem a linguagem, embora seja nítido o esforço que seus entusiastas têm realizado ao longo dos anos para transformá-la de uma linguagem "pessoal" (veja o nome inicial dela) em uma linguagem profissional.

Para saber mais: www.phpmania.org

ASP

O ASP (*Active Server Pages*) da Microsoft surgiu em fins de 1996 não como uma linguagem, mas uma estrutura de programação em script que se utiliza de linguagens como VBScript[2] (a mais comum), JScript[3] ou PerlScript[4] processadas pelo lado servidor para geração de conteúdo dinâmico na *web*.

O script roda nativamente em servidores Windows, através do serviço chamado de IIS (*Internet Information Service*) - o servidor web da Microsoft, ou do PWS (*Personal Web Server*) em ambientes com Windows 98. Embora raro, pode rodar também em outras plataformas, como Linuxno servidor Apache, desde que se tenha um módulo específico instalado. O ASP disponibiliza ligações com outras tecnologias (como consultas a bancos de dados), através da biblioteca de componentes ActiveX.

O uso do ASP original e do conjunto de tecnologias ActiveX está em pleno declínio, dada a maturidade da tecnologia .NET. O ASP vem sendo gradativamente substituído pelo ASP.NET que proporciona uma gama maior de recursos e um melhor desempenho.

ASP.NET

ASP.NET surge no início de 2002 como a plataforma padrão da Microsoft para o desenvolvimento de aplicações *web* e o sucessor da tecnologia ASP. É um componente do servidor *web* IIS que permite, através de uma linguagem de programação integrada no *framework* .NET, criar páginas dinâmicas.

Sendo baseado no *framework* .NET (o contraponto da Microsoft ao Java), ASP.NET aproveita todas as suas características. Por isso, as aplicações para essa plataforma podem ser escritas em várias linguagens, como C#[5] e VB.NET[6]. As páginas ASP.NET conhecidas como "*web forms*", são os componentes principais no desenvolvimento de uma aplicação. Os *web forms* são definidos em arquivos ASPX, que são tipicamente HTML ou XHTML juntamente com controles web onde são descritos os conteúdos da página e, adicionalmente, pode haver também codificação que roda no servidor, no mesmo estilo do PHP ou JSP.

Embora se possa desenvolver aplicações ASP.NET utilizando somente um editor e o compilador .NET, o ambiente de desenvolvimento mais comum das aplicações ASP.NET é o Visual Studio.NET que possui algumas características que facilitam o trabalho do programador, como os componentes visuais para criação de formulários de páginas *web*.

Uma aplicação para *web* desenvolvida em ASP.NET pode reutilizar código de qualquer outro projeto escrito para a plataforma .NET, mesmo que em linguagem diferente. Uma página ASP.NET escrita em VB.NET pode chamar componentes escritos em C# ou *web services* escritos em C++, por exemplo. Ao contrário da tecnologia anterior (ASP), as aplicações ASP.NET são compiladas antes da execução, trazendo sensível ganho de desempenho.

As aplicações *web* ASP.NET necessitam do *framework* .NET e do servidor IIS para rodar, pelo menos na plataforma Windows. Este é um dos pontos mais criticados, principalmente pelos defensores do *software* livre, já que quase todas as demais soluções para o desenvolvimento *web* estão no reino do código aberto. Existe um esforço para permitir que aplicações ASP.NET (na verdade toda a plataforma .NET) possam rodar em outras plataformas, como o Linux, conhecido como projeto Mono. A favor do .NET existe o argumento de que, apesar de ser tecnologia não aberta, há uma variedade de softwares distribuídos gratuitamente (como o Visual Studio Express Edition) além da já tradicional preocupação da Microsoft em facilitar a vida do desenvolvedor com seus inúmeros *templates*, *wizards*, etc.

Para saber mais: www.aspbrasil.com.br

[2] Uma linguagem de script da Microsoft que mistura JavaScript com Visual Basic e faz parte do conjunto de tecnologias conhecido como ActiveX.
[3] JScript é o JavaScript da Microsoft, que não usa o nome para não ter questões de royalties.
[4] Uma derivação de PERL.
[5] C# é uma linguagem criada pela Microsoft como uma substituta do C++.
[6] VB.NET é a sucessora do VB (Visual Basic).

Conclusão

Estas não são as únicas linguagens web *server-side*. A todo o momento, cria-se algo novo. Vale citar o ColdFusion (CFML), por exemplo, que nasceu na casa do Flash, a Macromedia, em meados dos anos 90 – hoje de posse da Adobe, atuando mais recentemente (de 2004 para cá) numa "dobradinha" com Flex (linguagem *client-side*) de forma muito harmônica no DreamWeaver. Mas creio que seja mais que suficiente por enquanto. A seqüência natural para nosso curso, já que começamos com Java no 5º período, foi continuar na linha do JSP.

Em LP6, já tínhamos conhecido algumas tecnologias *client-side* (HTML, Javascript, Applets Java). Agora vimos o "outro lado" e sabemos as diferenças e possibilidades. A primeira pergunta que vem à cabeça talvez seja: "Qual o melhor caminho?". A resposta, como quase sempre, depende de um monte de fatores. As perguntas-chave que devem ser respondidas são:

- Posso gastar dinheiro com isso? Quanto?
- Qual o nível de complexidade do problema a ser resolvido?
- Que intimidade eu tenho com estas linguagens e ambientes?
- De quanto tempo eu disponho para aprender algo totalmente novo para mim?
- Já existe na empresa outra solução baseada na *web*? Em que linguagem?

De acordo com as respostas, algumas opções podem ser descartadas e, no fim, pode-se chegar a uma conclusão. Eventualmente, mais de uma tecnologia pode ser usada. Na verdade, é o que quase sempre ocorre. No caso da UFJF, por exemplo, optou-se, anos atrás, por usar primordialmente o PHP debaixo de um *framework* (MIOLO). Mas coexistem dentro do sistema outras tecnologias, como programas externos em Delphi e Java, relatórios em ferramenta Java (Jasper Reports) e, dentro das páginas geradas, o DHTML sempre se encontra (HTML+CSS+Javascript). O banco de dados é acessado por meio de uma camada de persistência, mas, seja como for, em algum nível há SQL. Enfim, não há uma regra que nos impeça de misturar tecnologias, desde que o façamos de forma planejada e consciente. O "melhor caminho" é tirar proveito de tudo que conhecemos. E quanto mais conhecermos, mais recursos teremos.

Servindo Hipermídia

O Servidor *Web*

O servidor *web* (o *software* – não confundir com o computador servidor) é o programa responsável por aceitar pedidos HTTP vindo de clientes, normalmente navegadores, e servi-los com respostas no mesmo protocolo, geralmente páginas *web*. O primeiro servidor *web* foi o usado por Tim Berners-Lee no CERN, centro europeu de pesquisa nuclear, em 1990, quando ele criou também o primeiro *browser*, que chamou de WorldWideWeb.

O processo se inicia com a conexão *on-line* entre o computador cliente e o computador servidor, onde está instalado o servidor *web*. Quando é feita uma solicitação pelo cliente, o servidor retorna os dados correspondentes, que podem ser estáticos (antecipadamente prontos, como páginas HTML fixas ou arquivos de imagem, texto, etc.) ou dinâmicos (conteúdo gerado dinamicamente através do processamento de algum programa).

O servidor *web* deve ser instalado na máquina servidora, configurado, adicionado dos *plug-ins* que forem necessários (para suportar uma linguagem não nativa, por exemplo) e, depois de se definirem os diretórios apropriados, é que podemos colocar as páginas, imagens, programas, etc. em seus locais específicos. Quando o cliente pede um arquivo (chamando seu endereço no navegador), o servidor *web* recebe o pedido e envia esse arquivo. Se for um *script* ou um arquivo executável no servidor (e o *plug-in* correspondente estiver devidamente instalado), o servidor o executa e, caso alguma saída seja especificada, envia esta saída ao cliente. Um servidor pode dar suporte a várias interfaces/linguagens, tais como: SSI, CGI, SCGI, FastCGI, JSP, PHP, ASP, ASP.NET, Server APIs como NSAPI ou ISAPI e outras.

Vejamos um exemplo simples, caso o pedido do cliente seja de um script ASP. No exemplo, o script alo.asp armazenado no diretório específico de um servidor com o IIS (*Internet Information Services*, da Microsoft) instalado:

```
<html>
<body>
  <%
    for i=1 to 3
      Response.Write("<p>alô mundo!</p>")
    next
  %>
</body>
</html>
```

O pedido, no navegador, seria algo como o endereço http://www.algum_lugar_na_web.com/alo.asp . A resposta seria processada no IIS e geraria a saída abaixo, que seria enviada ao cliente (na verdade, o protocolo HTTP exige que se envie um cabeçalho adicional na resposta, mas que não vem ao caso aqui).

```
<html>
<body>
  <p>alô mundo!</p><p>alô mundo!</p><p>alô mundo!</p>
</body>
</html>
```

O cliente veria, na tela do navegador, o resultado da página: três parágrafos de alô mundo. Muito semelhante com o que vimos em JSP, passando pelo servidor Tomcat embutido no NetBeans.

Embora os servidores *web* difiram em detalhes, praticamente todos têm algumas características em comum: receber e responder requisições via HTTP, manter *logs* (registros de uso), permitir autenticação através de *login* e senha, manipular conteúdo estático ou dinâmico, suportar conexão segura (criptografada) HTTPS, prover *virtual hosting* para servir vários sites usando um único endereço IP, etc.

Uma das maiores preocupações acerca do servidor *web* é quanto ao seu limite de carga, pois ele só pode lidar com um número limitado de conexões concorrentes de clientes (normalmente configurado como algo de 500 a 1000 conexões por porta, dependendo do porte do computador servidor) e só pode servir um número máximo de requisições por segundo. Quando os limites são atingidos, o servidor pode até parar de responder. As causas mais freqüentes de sobrecarga são: tráfego excessivo (picos), "pragas" virtuais como os ataques do tipo DDoS (*Distributed Denial of Service*), *worms* ou vírus

XSS, má configuração ou problemas de lentidão na rede. Quando algum desses fatores ocorre, o resultado é a demora na resposta, o retorno de erros HTTP ou a recusa de novas conexões. Para diminuir o risco de sobrecarga, costuma-se usar técnicas de controle de tráfego através de *Firewalls*, uso de *web cache*, mais de um domínio ou computador para separar requisições por tipo ou tamanho de arquivo, aumentar memória ou HD, etc.

O IIS, da Microsoft, e o Apache da Apache Software Foundation, de código livre, são os dois servidores *web* mais utilizados. Normalmente, para o mundo do *software* "proprietário" o primeiro e para o mundo do *software* livre, o segundo. Juntos, dominam quase 90% do mercado (pesquisa de abril/2008).

Tomcat

O Apache Tomcat, que utilizamos neste curso, é um servidor web "puro Java[7]" que serve como contêiner de *Servlets* Java, implementando especificações para rodar *Servlets* e JSP. O Apache Tomcat não deve ser confundido com o servidor *web* Apache, implementado em C, embora ambos sejam de código aberto e venham da mesma "casa" (ASF). O Tomcat nasceu da junção do código do Java Web Server, doado pela Sun à ASF, com o JServ, da própria ASF, em 1999 e gerou a versão 3.0 (a inicial para o público). Após várias alterações, revisões e adições, estamos (2008) no Tomcat 6.0.

Dando prosseguimento natural à estrutura do curso na linha do Java, as nossas páginas JSP e, consequentemente, os *Servlets* gerados por elas, foram servidos pelo Tomcat.

Conclusão

Há que se deixar claro duas coisas:

1. Se nosso curso seguisse a linha da Microsoft (.NET) ou do PHP, por exemplo, os servidores *web* seriam outros (normalmente o IIS para o .NET e o Apache para o PHP).
2. O NetBeans, IDE Java usado no curso, pode vir com o servidor Tomcat embutido, para facilitar a vida do desenvolvedor. Esse, no entanto, pode não ser o quadro encontrado em qualquer empresa que use JSP.

Portanto, cada caso é um caso e, obviamente, não temos condição de cobrir todo e qualquer panorama possível de ser encontrado. Assim sendo, se você entrar num time de desenvolvimento *web*, que passos serão necessários para essa entrada ser a mais suave e proveitosa? Siga os conselhos abaixo (os dois primeiros são específicos da nossa área, os demais são úteis para qualquer carreira):

- Procure, aos poucos, conhecer TODO o processo de desenvolvimento: como é feita a análise, que tipo de padrão de projeto é usado, que tipo de documentação é feita e, finalmente – o mais importante, se você começar programando – que ferramentas de codificação são usadas (*frameworks*, IDEs, editores, geradores de relatório, etc.).
- Estude os ambientes e linguagens utilizadas, a forma de distribuição, onde, como e em qual servidor *web* instalar os programas.
- Siga as regras e convenções usadas por outros membros mais experientes do time. Não invente nada de diferente no início. Criatividade é super importante, mas não no seu primeiro dia. Adquira a confiança necessária para depois sugerir melhorias e inovações.
- Demonstre interesse e seja "pró-ativo" (esse neologismo significa o "contrário" de "reativo" – aquele que só se preocupa com um problema depois que ele já aconteceu). Ou seja: tente antever situações que possam dar problema e já solucioná-las antes mesmo que ocorram.
- Não siga "maus" exemplos de colegas, mesmo que pareça "normal". Lembre-se de que, quando eles começaram suas carreiras, provavelmente tiveram que "ralar" bastante também. Seja pontual, respeitoso, eficiente e solícito.
- E, por último, mas não menos importante, pergunte sempre que tiver dúvidas. Não tente adivinhar nada. Seja humilde o suficiente para dizer que não sabe e precisa de ajuda (mas demonstre bastante atenção e tente aprender das primeiras vezes – se você perguntar 20 vezes a mesma coisa e continuar sem saber, aí complica...).

[7] Esta designação é dada a produtos que não usam outra tecnologia a não ser o próprio Java em sua construção.

Frameworks

Um *framework* é uma estrutura de suporte sobre a qual um projeto de software pode ser organizado e desenvolvido. Nesta estrutura encontramos bibliotecas de código, classes prontas ou semi-prontas, utilitários e todo o tipo de peça de *software* capaz de facilitar a codificação de sistemas. Uma parte de um *framework* é fixa, contendo serviços prontos para uso, de grande confiabilidade. Outra parte depende de extensões a cargo dos programadores, pois necessitam da complementação de funcionalidades que devem ser implementadas.

Os *frameworks* surgiram da necessidade de ter modelos semi-prontos, reutilizáveis, padronizados e de fácil acesso para agilizar a codificação, tornando as aplicações mais consistentes entre si e robustas, pois os blocos construtores de *software* que fazem parte de um *framework* via de regra já passaram por vários testes antes de serem liberados para uso. O propósito central é justamente acelerar e auxiliar o desenvolvimento, bem como unificá-lo de modo a que os vários projetos sigam um mesmo padrão, simplificando sua manutenção e fazendo com que os desenvolvedores se preocupem mais com o projeto do que com detalhes de implementação.

Não existe um "carimbo" que um *software* ganhe para ser chamado de *framework*. Na verdade, quase todas as linguagens em seu auge tiveram ferramentas de auxílio à codificação que poderiam ser chamadas de *frameworks*, num sentido mais amplo. Outras necessidades, como o mapeamento objeto-relacional, por exemplo, podem ter *frameworks* específicos. Mas, de modo mais estrito, quando nos referimos a um *framework* estamos identificando um produto de grande espectro de utilidade, que é mais do que uma simples biblioteca de funções ou classes, pois contempla um padrão de uso destas funções/classes de uma forma mais direcionada. Também é mais do que um padrão de projeto, pois estes apenas indicam caminhos, enquanto os *frameworks* provem formas de implementação.

Um bom *framework* deve fornecer: grande possibilidade de utilização em várias necessidades, ferramentas acessórias produtivas, bibliotecas internas bem documentadas, facilidade de aprendizado e uso, extensibilidade, segurança, robustez e eficiência. Se o *framework* contiver tudo isso, teremos enorme redução de custo e tempo de produção de sistemas e mais consistência e compatibilidade entre eles.

No entanto, nem tudo são flores... Um *framework* 100% adequado às necessidades de uma empresa pode não existir pronto. E fazer um é bem complicado, pois exige muito mais tempo do que apenas fazer um sistema. Então poderemos lidar com duas tarefas: fazer o *framework* e o sistema dentro dele. Além disso, temos um clássico problema de "cobertor curto" com relação ao *framework*: se ele for muito simples de usar, provavelmente será pouco útil em problemas além do trivial; se resolver qualquer problema imaginável, tenderá a ser super-complexo e difícil de aprender e usar.

Alguns exemplos de *frameworks*: Ruby on Rails (para Ruby), Zend (para PHP), JSF (*JavaServer Faces* - mais recentemente, o JSF originou o Visual Web JSF, que incorpora facilidades de *design* gráfico ao JSF), Apache Struts (Java), Microsoft .NET Framework.

Aqui em Juiz de Fora, na UFJF, estamos ajudando a desenvolver um *framework* chamado MIOLO, de código aberto em PHP. Trata-se de uma parceria com várias outras instituições, que teve início na Univates, uma universidade do Rio Grande do Sul e sofreu várias atualizações na UFJF, principalmente sob a "batuta" do nosso colega Ely, que também é professor aqui na Universo. O MIOLO fornece objetos para manipulação de bases de dados, apresentação, trata da segurança de usuários, encapsulamento e modularização de aplicações. O SIGA (Sistema Integrado de Gestão Acadêmica) da UFJF é feito sobre o MIOLO.

Sociedade da Informação

"Sociedade da Informação" é um termo - também chamado de Sociedade do Conhecimento, relacionado à Nova Economia ou até à Globalização - que surge no fim do Século XX. Este tipo de sociedade encontra-se em processo de formação e expansão.

A sociedade humana está em constante mutação, num processo em que as novas tecnologias são as principais responsáveis pela mudança. Alguns autores identificam um novo paradigma de sociedade que se baseia num bem precioso, a informação, daí o termo "Sociedade da Informação".

Até algum tempo, saber ler, escrever e fazer algum cálculo matemático era suficiente para se viver em harmonia e bem-estar na sociedade. O cenário mudou e as necessidades de qualificações profissionais e acadêmicas aumentaram consideravelmente. O ser humano tem a capacidade de se adaptar e como tal, as pessoas devem desenvolver uma atitude flexível, com conhecimentos generalistas, capazes de se formar ao longo da vida. A sociedade exige pessoas com uma formação ampla, mas com especializações pontuais, com um espírito empreendedor e criativo, com o domínio de uma ou várias línguas estrangeiras e grande capacidade de resolução de problemas.

Este novo modelo de organização das sociedades se desenvolve num modo de desenvolvimento social e econômico onde a informação, como meio de criação de conhecimento, desempenha um papel fundamental na produção de riqueza e na contribuição para o bem-estar e qualidade de vida dos cidadãos. A condição para o avanço é possibilitar o acesso geral às tecnologias de informação e comunicação (inclusão digital).

Vários programas governamentais ou de ONGs ao redor do mundo se preocupam com a questão e ressaltam a importância da inclusão digital. A idéia do "computador popular" a preço de custo é uma das várias iniciativas possíveis. Outras possibilidades incluem computadores públicos (como os "Infocentros" da UFJF).

A sociedade tenderá a ser cada vez mais competitiva, criando mais riqueza e consequentemente qualidade de vida, tornando-se uma sociedade mais livre. Mas para que isto seja possível e não se criem maiores desigualdades sociais, as políticas educacionais desempenham um papel primordial. Assim, a escola assume um papel fundamental na Sociedade da Informação, que é de dotar o homem de capacidades para competir com o avanço tecnológico, condicionando-o, de maneira a que este avanço não seja autônomo, e possa ser controlado, para que sejam as nossas necessidades a corresponder ao desenvolvimento tecnológico e não o desenvolvimento tecnológico a moldar as nossas necessidades.

Com a introdução de máquinas e robôs nas indústrias tem aumentado a taxa de desemprego. Estamos assistindo mudanças profundas na sociedade. A perda de postos de trabalho, a extinção de algumas profissões e a reconversão de outras até serem substituídas por novas demanda um longo período de adaptação, que estamos vivenciando. Esse é um dos maiores desafios gerados pela própria Sociedade da Informação, junto à necessidade de promover a inclusão digital. Outro problema, gerado desde o início da revolução industrial é a crescente dependência tecnológica. Estamos cada vez mais "reféns" das máquinas e dos sistemas computadorizados. Não se trata de apelar para a ficção-científica barata, onde robôs tentam nos dominar, mas a verdade é que realmente não conseguimos sequer imaginar o mundo sem a parafernália tecnológica da qual dependemos. Só para exemplificar até que ponto pode chegar esse problema: recentemente, num fato mundialmente conhecido, um avião com mais de 200 pessoas a bordo caiu e todos morreram. Entre os fatores que podem ter contribuído para o problema, sugere-se a possibilidade de um sensor ter quebrado, levando a informação errada ao sistema computadorizado do avião, que teria tomado a decisão errada e levado à série de eventos que culminou com a tragédia da queda. Ainda que não tenha sido essa a causa, a simples possibilidade de ter sido, exemplifica bem o quão dependentes das máquinas nós estamos.

Documentação Analógica e Digital

Até meados do século passado, toda a informação gerada pela humanidade era armazenada em meios analógicos: livros, filmes, fotos, etc. Com as possibilidades surgidas após o desenvolvimento dos computadores e a crescente demanda por informação, era inevitável que as coisas se juntassem e até acabassem batizando o nome da nova ciência como "Informática". Gostemos ou não, estamos envolvidos com a informação até o pescoço. Portanto, nem só de contas matemáticas vivemos nós.

Após o aumento da capacidade de armazenamento das memórias de massa (discos rígidos, CDs, DVDs, pen-drives, etc.) o conteúdo armazenado em meio digital cresceu exponencialmente. A quantidade de texto que se pode guardar em um simples DVD, por exemplo, é enorme. Faça as contas: 4,7 GB significam quase 5 bilhões de letras (sem compactação). Era óbvio que se usasse todo esse "poder" para substituir o papel e outras mídias analógicas, com a grande vantagem de não haver degradação do conteúdo, com o tempo. Bem, quanto a isso há um probleminha... Vamos contar uma história sobre os manuscritos do Mar Morto.

Os *manuscritos do Mar Morto* são uma coleção de cerca de 850 documentos (em pergaminho), incluindo textos da Bíblia Hebraica (Antigo Testamento), que foram descobertos entre 1947 e 1956 em 11 cavernas próximo de Qumran, uma fortaleza a noroeste do Mar Morto, em Israel (em tempos históricos uma parte da Judéia). Eles foram escritos em Hebraico, Aramaico e Grego, entre o século II a.C. e o primeiro século depois de Cristo. Representam vários pontos de vista, incluindo as crenças dos Essênios e outras seitas. Os textos são importantes por serem praticamente os únicos documentos bíblicos judaicos hoje existentes relativos a este período e porque eles podem explicar muito sobre o contexto político e religioso nos tempos do nascimento do Cristianismo. Os pergaminhos contêm pelo menos um fragmento de quase todos os livros das escrituras hebraicas, de regras da comunidade, escritos apócrifos, calendários e outros documentos.

Enfim, um dos maiores achados arqueológicos da humanidade. Em documentação analógica (pergaminho). Bem... e qual o problema?

Imagine se os antigos escritores tivessem um computador. Talvez não usassem pergaminhos, mas algo como disquetes ou CDs. Aí é que está o problema: e se os encontrássemos hoje? Teríamos a versão 1.0 do "MoisesOS" para abrir os discos? Teríamos o "OpenAramaic" para ler os textos? E, mesmo se conseguíssemos reconstruir equipamentos e *softwares*, os discos ainda funcionariam, depois de séculos? Não. Seria tudo lixo. Lixo digital. Na verdade, não precisaríamos ir tão longe: se acharmos hoje uma fita cassete com dados de Apple][(é... se podia guardar dados em fita cassete) de uns 20 anos atrás, já seria quase impossível lê-la.

Já ouvimos várias vezes algum vendedor dizer "o CD dura cem anos!". Bobagem. Não dura isso tudo. A vantagem do "digital" sobre o "analógico" não é a duração da mídia, mas a não degradação da informação (textual, gráfica, sonora, etc.), mesmo que se copie da cópia da cópia da cópia, DESDE QUE AS MÍDIAS SE MANTENHAM INTACTAS. Para tal, devemos manter sempre *backups* em mídias atualizadas.

Outra balela que ouvimos muito é sobre a idéia do escritório "*paperless*" (sem papel). Nunca se imprimiu tanto. Nunca se leu tanto (em papel). É verdade que alguns jornais, por exemplo, tiveram suas tiragens reduzidas por conta do jornalismo *on-line* via *web*. Uma ou outra coisa pode acabar migrando de vez do reino analógico para o digital, mas daí a acabar com o uso do papel impresso vai uma enorme distância.

De qualquer forma, guardados os exageros, a documentação digital – e sua manutenção – ganhou importância desde meados da década de 1990. É nesse contexto que entram as técnicas de GED (Gerenciamento Eletrônico de Documentos), das quais falaremos posteriormente.

Indexação de Conteúdo Textual

Um dos maiores desafios na área da informação, devido à quantidade imensa disponível de dados digitais, é justamente encontrar o que queremos em um mar de documentos. A indexação de conteúdo textual é necessária para organizar esses dados. A idéia é a mesma de quando não havia computador: índices por ordenação alfanumérica. Só que, o que antes era feito em papel, agora é indexado em arquivos digitais.

Independentemente do tipo de arquivo a ser indexado, esta operação é sempre feita sobre textos. Por exemplo: uma foto num banco de dados de imagens é procurada pelo seu título ou por palavras-chaves agregadas ao arquivo de imagem. Embora existam alguns estudos de reconhecimento de som ou imagem pelo conteúdo (sonoro ou visual), isto ainda é considerado uma tarefa muito difícil e "humana" demais para simular eletronicamente. Qualquer criança pequena reconhece um rosto em uma foto, mas tente fazer um programa de computador para isso... Mesmo o processo de busca de imagens do Google (o sistema de busca mais usado no mundo) é "cego", ou seja: não identifica elementos na imagem, apenas texto associado a ela. Portanto, nos cabe entender o processo de indexação de texto.

A forma de ordenação já é conhecida. Vocês já viram algoritmos de "*sort*" em outras disciplinas. É a mesma coisa, porém algumas decisões de projeto devem anteceder a criação de índices. Para ordenar grandes quantidades de documentos textuais, deve-se primeiro decidir se todas as palavras de cada texto serão indexadas ou se apenas algumas

palavras-chave serão alvo de índice. Evidentemente que, se a primeira opção for tomada, a busca de palavras será mais lenta, porém mais completa, enquanto que, se optarmos por indexar apenas palavras relevantes em relação ao documento, uma busca será mais rápida e mais "focada", ou seja: os resultados serão mais expressivos. Por exemplo: se fossemos indexar este texto poderíamos guardar referências para todas as palavras dele ou só para as mais relevantes: documentação, conteúdo, sociedade da informação, indexação, texto, busca.

Outra escolha importante no projeto de ordenação e busca textual é decidir se será possível encontrar palavras por semelhança fonética. Essa escolha é fundamental, pois define os algoritmos de indexação e de procura de palavras. Por exemplo: suponha que neste texto existisse a palavra "MONITOR". Se indexarmos todas as palavras deste texto, poderemos encontrá-la no índice. Mas se estivermos procurando por "MONITORES" (plural), vamos encontrar? E se errarmos a digitação e escrevermos "MONITWR"? Se quisermos usar procura fonética, um algoritmo muito conhecido que é usado para determinar semelhança entre palavras é o "soundex". Neste algoritmo, cada palavra recebe um código (que pode ser um char(4), por exemplo) calculado em função do som da palavra. Palavras muito parecidas recebem o mesmo código. Esse código é que fica indexado e não a palavra completa (o que, de quebra, reduz o tamanho do arquivo de índice). Para procurar uma palavra, primeiro encontra-se o seu código. De posse do código, entra-se no índice e todas as ocorrências daquela palavra (ou palavras foneticamente parecidas) estarão disponíveis.

GED / Workflow

GED e *Workflow* são mais alguns dos termos que eram muito fortes no momento da criação desta disciplina e hoje já estão um tanto ultrapassados. De qualquer modo, vamos conhecer o histórico destas tecnologias, as principais aplicações, o caminho que tomaram e por quais outras soluções estariam sendo substituídas atualmente.

GED

GED (Gestão Eletrônica de Documentos ou Gerenciamento Eletrônico de Documentos) se refere a uma tecnologia que possibilite gerar, manter e compartilhar informações documentais em meios eletrônicos (computadores). Tornou-se popular no início dos anos 1990, no auge da idéia do escritório *paperless* (do qual já falamos). Hoje, um termo mais utilizado para designar algo semelhante (porém mais abrangente) seria ECM (*Enterprise Content Management*). Gosta de sopa de letrinhas? Então tem mais... Outros termos relacionados seriam: EDM (*Electronic Document Management* – algumas vezes transcrito como *Enterprise Document Management*) que seria o GED "original", IDM (*Integrated Document Management*), WCM (*Web Content Management*), BPM (*Business Process Management*), DAM (*Digital Assett Management*), CIS (*Content Intelligence Services*), EDMS (*Engineering Document Management System*) e outros tantos.

Para não sermos esmagados por essa montanha de siglas, vamos ao mais importante: o que é exatamente esse gerenciamento? Pra quê serve?

A idéia central do GED é consolidar a informação (o conteúdo corporativo) de uma empresa ou instituição na sua totalidade (não só alguns departamentos isoladamente) e disponibilizar essa informação a todos os setores. Dentro da idéia da "Sociedade da Informação" (da qual já falamos), conhecimento é poder. E compartilhar esse conhecimento entre todos os tomadores de decisão de uma empresa é fundamental para que ela tenha sucesso. Isso inclui técnicas de inteligência artificial de análise, indexação e compreensão de conteúdo.

Um software GED que se preze, deve prover, no mínimo, captura de dados, controle de versão, publicação e consulta. Em resumo: permitir a gerência e ser o repositório de todos os dados de documentação relevantes. Tudo para garantir que os gerentes tomem decisões baseadas no maior conhecimento possível. Esse é o objetivo principal do GED e de todas as outras tecnologias correlatas (e suas siglas mirabolantes).

Em termos práticos, o GED se resumia, no seu início, quase que exclusivamente a escanear documentos para transformá-los de papel em imagens e gerenciar (eletronicamente) estes resultados. Eventualmente, o uso de um *software* OCR (*Optical Character Recognition*) permitia que se traduzisse a imagem em texto, o que possibilitava o uso de técnicas de indexação textual (das quais também já falamos). O conhecimento gerado a partir destes documentos e seu posterior compartilhamento configuravam o ambiente GED. Hoje, sistemas ECM vão além, fazendo amplo uso da *web* e suportando, além dos *scanners*, *webcams*, câmeras e gravadores digitais e outros dispositivos multimídia, lidando com imagem, som, vídeo, texto e vários tipos de documentos como planilhas ou arquivos de dados.

Alguns exemplos de ferramentas "prontas" de GED são: Alfresco, Documentum, Hummingbird DM, ISIS Papyrus, Laserfiche, Livelink, Oracle Stellent, Perceptive Software, Maarch, SharePoint, Saperion, SAP Netweaver, TRIM Context e Xerox Docushare. Mas, nada impede que uma empresa desenvolva seus próprios mecanismos de gerenciamento de documentos, de forma mais abrangente ou mais restrita, conforme suas necessidades e até inclua esses mecanismos dentro dos sistemas computacionais já existentes. Por exemplo, suponha que dentro do sistema de pessoal de uma empresa seja desejável manter o registro de imagem de alguns documentos dos funcionários (CPF, Identidade, Carteira de Trabalho, etc.). Isso é perfeitamente possível, desde que se crie uma estrutura no banco de dados com campos do tipo BLOB (*Binary Large OBjects*) ou semelhantes. Algum programa de entrada deve permitir ler os arquivos de imagem de um *scanner*, possivelmente passar um OCR para armazenar as palavras reconhecidas e guardar tudo no banco de dados. Outro programa deve fazer pesquisa sobre as tabelas envolvidas. Isso é um micro exemplo (um "GEDzinho"), mas ilustra perfeitamente a idéia. Estenda esse conceito por todos os documentos e arquivos multimídia de uma empresa, adicione um motor de busca incrementado a fim de facilitar a pesquisa e consulta sobre esses dados, disponibilize isso aos gerentes e pronto: seu GED estará montado.

WorkFlow

Ao pé da letra: "Fluxo de Trabalho", ou seja: uma seqüência de passos necessários para realizar uma tarefa. Segundo a WfMC (*Workflow Management Coalition*[8]) é a "automação do processo de negócio, na sua totalidade ou em partes, onde documentos, informações ou tarefas são passadas de um participante para o outro para execução de uma ação, de acordo com um conjunto de regras de procedimentos". O mesmo fenômeno de decadência de nomenclatura e decorrente substituição por uma sigla mais "na moda" também ocorre com o termo "Workflow", que pode ser hoje confundido com BPM (*Business Process Management*) ou outros nomes semelhantes (mais um pouquinho de sopa de letrinhas). Algumas ferramentas típicas de GED podem conter um "módulo" para tratar do *workflow*, por isso os termos aparecem juntos algumas vezes.

O importante é que, automatizando os processos de trabalho com a definição do "melhor" *workflow* em uma empresa, se ganhe em padronização e agilidade na execução das tarefas, produtividade, redução de custos, etc. Estes são os objetivos a serem alcançados, mas, na prática, isso pode ser bastante complicado de se conseguir.

O termo *workflow* possivelmente começou a ser usado na sua forma atual na indústria de *software* pela empresa Filenet, nos anos 1980, quando do lançamento do programa de automação de processos de negócios "Workflo", mas a idéia da "linha de produção" (que tem muito a ver com a pré-história do *workflow*) é muito mais antiga e remonta aos tempos de Ford.

Existem vários tipos de *workflow* e seu uso depende do ambiente onde se deseja aplicá-lo. Pode até ter partes "manuais", onde um usuário decida sobre a quem enviar determinado documento ou pode ter regras de trabalho que decidam automaticamente este fluxo , baseado no tipo de documento ou alguma característica como palavras-chave.

Seja como for, o *workflow* sempre define uma seqüência de operações, como trabalho de pessoas, mecanismos, grupos de pessoas ou grupos de mecanismos. Isso pode representar qualquer coisa do mundo real, que se possa dividir em partes constituintes de uma solução. Qualquer atividade que seja habilitada por uma organização sistemática de recursos, regras, trabalho e fluxo de informação em um processo que possa ser documentado e aprendido.

Tal como dissemos com relação ao GED, existem várias aplicações de *workflow*, integradas ou não a outras ferramentas, em áreas específicas, tratando de roteamento automático, automação de processos e integração entre diferentes elementos humanos, *softwares* e *hardwares* que contribuem com o suporte ao trabalho cooperativo. Mas o seu principal papel talvez seja realmente promover o roteamento de documentos em uma empresa para revisão, ação ou aprovação. Mencionaria o Lotus Workflow, o Windows Workflow Foundation e o ProcessIt! como ferramentas interessantes, mas provavelmente esse assunto veio à tona como tópico dessa disciplina por conta do enorme sucesso do Lotus Notes no seu auge (anos 90).

O Lotus Notes é uma aplicação colaborativa em *desktop*, do tipo *client-server* (que também pode ser chamada de aplicação *groupware*), hoje de posse da IBM. É uma espécie de "canivete suíço" dos *softwares*, possuindo, entre outras coisas, leitor de *e-mail*, mensagens instantâneas, *browser*, bloco de anotações, agenda, um ambiente de desenvolvimento *desktop* e *web* e uma plataforma de interação com aplicações colaborativas, sendo essas duas últimas características as principais responsáveis pela sua relação com o *workflow*. O Lotus Workflow é, obviamente, mais específico, mas pode trabalhar integrado ao Notes. O principal concorrente do Notes é (adivinhem...) da Microsoft, o Exchange, que também possibilita criar aplicações típicas de *workflow*.

[8] A WfMC é uma organização global, fundada em 1993, de usuários, desenvolvedores, consultores, analistas, universidades e grupos de pesquisa em *workflow* e BPM.