Comunicado Técnico 62

Novembro, 2004
Campinas, SP

ISSN 1677-8464

Uma Análise Comparativa das Soluções Tecnológicas Utilizadas nas Apresentações de Dados da Agência de Informação Embrapa

Maria Fernanda Moura[1]
Adriana Delfino dos Santos[2]
Carla Geovana do Nascimento Macário[3]
Sérgio Aparecido Braga da Cruz[4]

A Agência de Informação Embrapa (Evangelista et al., 2003), ou simplesmente Agência, corresponde a um conjunto de várias Agências de Produto, que organizam informação técnica relevante para o agronegócio, especializada por produto, estruturada sob a ótica da cadeia produtiva, disponibilizada na internet para atender a perfis diversificados de consumidores de informação, tais como: produtores rurais, extensionistas, pesquisadores, técnicos, professores, estudantes, etc.

Todo o conjunto de informações da Agência é organizado e armazenado em meio eletrônico. Assim, foi criado um repositório de conteúdos de informações, categorizados segundo os diferentes ambientes de consumo da informação, e também criou-se um *sítio* onde são disponibilizadas as informações das diversas agências de produtos. Para manter os dados desse *sítio* íntegros e constantemente atualizados, foi construído um sistema automatizado (denominado, a partir deste ponto, por Gerenciador de Conteúdos), baseado na arquitetura cliente servidor e acessado através da intranet da Embrapa, para inserção, alteração e exclusão de dados das diversas agências de produtos, permitindo a troca e o reuso dos mesmos.

O Gerenciador de Conteúdos e as ferramentas de suporte à busca e navegação no *sítio* da Agência foram construídos com base no paradigma de desenvolvimento de software em camadas denominado Model View Control - MVC (Subrahmanyam et al., 2001; Bushman et al., 1996). A idéia central desse modelo é separar a manipulação dos dados, que corresponde à camada *model* - ou modelo, da forma de apresentação dos mesmos, que é a interface com o usuário: camada *view*, colocando-se o controle entre eles (camada *control*). A camada de controle promove a interação entre as duas outras, isto é, reage às "entradas" fornecidas pelo usuário do sistema.

É na camada de apresentação de dados - *view*, que ocorrem as trocas de informação com o sistema e a formatação de dados a serem apresentados; nela entra o trabalho de um *web designer*. O *web designer* é o profissional que deve se incumbir do *look and feel* do sistema, isto é, da aparência do sistema e de como o usuário interage com essa apresentação visual. Em geral, o *web designer* possui um bom conhecimento do uso de ferramentas de construção de páginas *web*, manipulação de figuras, HTML - HyperText Markup Language, e javascript - linguagem de programação utilizada, em geral, para checar e controlar a troca de dados entre formulários em páginas *web.* Em uma situação ideal, o

[1] M.Sc. em Engenharia Elétrica, Pesquisadora da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo - 13083-970 - Campinas, SP. (e-mail: fernanda@cnptia.embrapa.br)
[2] M.Sc. em Engenharia Elétrica, Pesquisadora da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo - 13083-970 - Campinas, SP. (e-mail: adriana@cnptia.embrapa.br)
[3] M.Sc. em Engenharia Elétrica, Pesquisadora da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo - 13083-970 - Campinas, SP. (e-mail: carla@cnptia.embrapa.br)
[4] M.Sc. em Engenharia Elétrica, Pesquisador da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo - 13083-970 - Campinas, SP. (e-mail: sergio@cnptia.embrapa.br)

Embrapa

trabalho do *web designer* deve ser restrito à definição dos conteúdos de informação a serem enviados ao sistema, formas de interação e apresentação de conteúdos ao usuário. Mudanças no *look and feel* do sistema, não devem provocar trabalho extra nas demais camadas. E, o trabalho dos programadores deve se restringir ao desenvolvimento das camadas do sistema responsáveis por obter os conteúdos enviados, processá-los e devolver os conteúdos resultantes do processamento às partes do sistema que vão apresentá-los ao usuário – camada de apresentação (*view*). Não deve ser exigido do *web designer* o domínio de como processar os conteúdos, e sim de como trocá-los com o usuário da maneira mais agradável possível.

O desenvolvimento da camada *view* da Agência, na sua versão 1.0, empregou várias soluções tecnológicas e vários perfis profissionais: um *web designer* com pouca formação em desenvolvimento de software, um segundo *web designer* com formação em publicidade e propaganda e, programadores trabalhando como *web designers*. Para padronizar a interface com o usuário, atingir um *look and feel* mais agradável e adequá-la aos diversos perfis de usuários, foi contratado um *web designer* que trabalhou junto a um profissional de editoração de imagens e especialistas do domínio – o que resultou na aparência da versão 1.1 da Agência. O trabalho de adaptação do novo desenho às trocas de dados do sistema foi realizado pela equipe de desenvolvimento. Durante esse trabalho puderam ser observadas as vantagens e desvantagens de ter utilizado cada uma das soluções tecnológicas, do ponto de vista do esforço empregado nas mudanças, dado o tempo disponível para a atividade e conhecimento das tecnologias pela equipe.

O objetivo deste documento é mostrar a experiência de utilização de algumas tecnologias na camada *view* da Agência, cobrindo suas principais características, e, comparando algumas vantagens e desvantagens de cada uma, do ponto de vista do *web designer*. A troca da interface gráfica da Agência, na passagem da versão 1.0 para 1.1, é utilizada como exemplo do esforço de mudança, relativo às alterações ocorridas no sistema para refletir seu novo *look and feel*. E, ainda, as soluções são comentadas, com as respectivas observações de onde as utilizações das tecnologias escolhidas poderiam ser trocadas.

Soluções Tecnológicas usadas na Camada *View* da Agência

A partir da Fig. 1 é possível ter uma idéia geral de onde cada tecnologia foi utilizada e do que ocorre em cada camada – *View*, *Control* e *Model* – tanto na parte do ferramental que fica disponível na intranet quanto da parte da extranet. Os nomes das tecnologias estão representados na cor vermelha, e as setas vermelhas significam que a tecnologia auxilia a geração do **dado** ou do aplicativo (na sua camada de desenvolvimento).

A informação apresentada no Gerenciador de Conteúdos e no *sítio* da Agência pode ser dividida em conteúdo estático e dinâmico. Por conteúdo dinâmico deve-se entender toda troca de informação através de formulários de entrada, relatórios de resultados, expressões de **busca** no *sítio* e **cesta de documentos**, cujo funcionamento é similar ao de um *bookmark* – criado no navegador[5] para guardar endereços e títulos de *sítios* de interesse do usuário. O conteúdo estático deve ser visto como os hipertextos do *sítio*, que são gerados a cada nova publicação da Agência – isto é, após atualizações da base de dados; na figura correspondem às **páginas estáticas de conteúdos**. Além desses conteúdos especialmente criados para a Agência, também são disponibilizados hipertextos com as apresentações dos metadados – **páginas com metadados dos recursos eletrônicos**.

Os **recursos eletrônicos** são informações auxiliares/complementares referenciadas no *sítio*, nos conteúdos das páginas estáticas, e podem ser: publicações eletrônicas na *web*, outros *sítios*, imagens diversas, etc. A base de dados de recursos eletrônicos abriga os recursos que foram autorizados, por seus autores ou proprietários de direitos autorais, a serem armazenados na Agência; isso ocorre através de uma operação de ***Upload*** de arquivo para o repositório. A base de metadados fica sob um sistema gerenciador de banco de dados (**SGBD**) **ORACLE.** Após a publicação das páginas estáticas de conteúdos e das páginas de metadados, é gerado o **arquivo de índices para a Busca** pela ferramenta **SWISH-e** (Cruz, 2003); pois esse índices têm como base essas duas fontes de dados. A linguagem de programação utilizada nos aplicativos, nas camadas *Control* e *Model*, foi **JAVA**, salvo no aplicativo de busca em que utilizou-se PHP também nessas camadas.

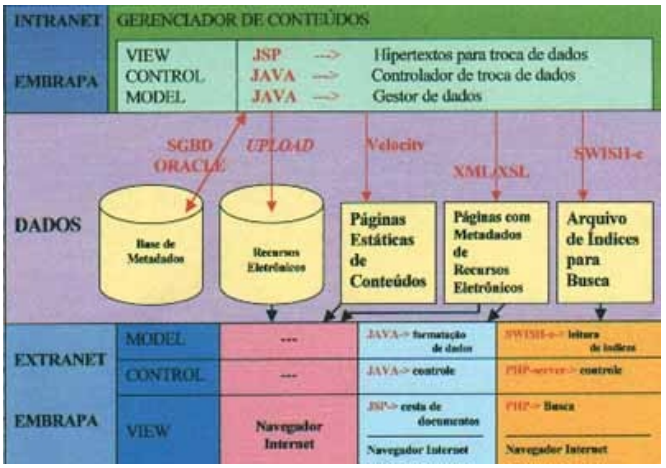

FIG. 1. Visão geral das diferentes tecnologias empregadas na Agência.

Os próximos subitens tratam especificamente das soluções tecnológicas utilizadas na camada *View*, para gerar conteúdo dinâmico e conteúdo estático – que são: JSP, PHP, XML/XSL e Velocity. Deve-se lembrar, que o ideal de cada solução apresentada, é poder ser utilizada por um *web designer*, minimizando a necessidade de conhecimento das particularidades de linguagens de programação ou minimizando a interferência do programador com a solução. Para um melhor resultado didático e permitir uma melhor comparação, os exemplos mostrados nos próximos subitens são fictícios, e não os reais implementados na Agência; repetindo-se o mesmo exemplo para cada solução tecnológica.

[5] Que pode ser o Internet Explorer, Mozilla, Opera, Netscape, etc.

JSP - JAVA Server Pages

JSP é uma ferramenta para gerar páginas *web* que combina HTML e *scripting tags*, com o objetivo de facilitar a criação e manutenção dessas páginas (Subrahmanyam et al., 2001). *Scripting tags* são marcadores especiais, acrescentados ao código HTML, que serão interpretados por um programa específico no servidor. Como ilustrado na Fig. 2, um programa JSP é um formulário HTML com *scripting tags*, que envia dados via um protocolo (http/https) para a máquina servidora. No servidor há um *container*, um "programa especial", que na arquitetura J2EE, gerencia, em tempo de execução, os componentes da aplicação desenvolvidos de acordo com as especificações dessa arquitetura (Subrahmanyam et al., 2001); nesse caso, um *web container* que hospeda e gerencia JAVA *servlets* e páginas JSP, denominado Tomcat (The Apache Software Foundation, 2004a). O programa JSP, no *container*, é transformado em um novo formulário HTML, a partir das especificações do original, e um JAVA *servlet*, que é o programa habilitado a interpretar e responder uma requisição http/https. Os dados resultantes da execução do *servlet* são colocados nas posições indicadas pelas *scripting tags* no formulário HTML resultante.

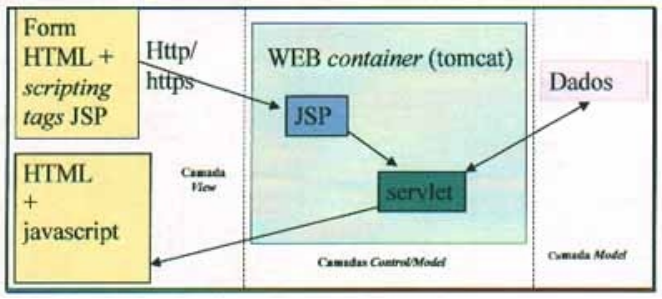

FIG. 2. Funcionamento de um programa JSP.

Para exemplificar a construção de um JSP, considere um aplicativo bancário, onde o cliente informa seu número de conta, número da agência, nome e valor do débito que deseja fazer, para o qual seja utilizado um JSP na apresentação do resultado dessa operação, denominado "saldo.jsp". A aparência de "saldo.jsp" poderia ser como a ilustrada na Fig. 3. Deve-se notar, na figura, que é necessário especificar a classe JAVA utilizada (na *scripting tag "usebean"*, o objeto "debito", que é da classe "OperacaoConta") e as classes utilizadas para chegar aos resultados (no caso, especificadas na *scripting tag "import"* – incompleta no exemplo), também é necessário saber como especificar os campos que se deseja (propriedades da classe OperacaoConta nas *scripting tags* do tipo *getProperty*). Logo, é necessário um certo grau de conhecimento de JAVA para utilizar as *scripting tags*, além de saber quais os dados que estarão disponíveis para o formulário, que são as propriedades especificadas, propriamente ditas. Note que o novo saldo não foi calculado no programa JSP, ele já estava disponível em uma das propriedades do objeto "debito", ou seja, ele foi calculado no servidor por um trecho de programa JAVA que não aparece no exemplo (porque está na camada *Model*[6]). Ou seja, sem a intervenção de um programador JAVA ou um conhecimento razoável de JAVA, não é possível escrever um JSP que funcione adequadamente.

```
<%@ import .... classes de interesse/>
<jsp: usebean name="debito" type="caminho/OperacaoConta"/>
<html> <title>.. <body>
<h1> Resultado da Operação Débito</h1>
 <li> Nome Cliente: <jsp:getProperty name="debito" property="nome"/> </li>
<li> Conta: <jsp:getProperty name="debito" property="conta"/> </li>
<li> Agência: <jsp:getProperty name="debito" property="agencia"/> </li>
<li> Saldo Anterior: <jsp:getProperty name="debito" property="saldoAnterior"/> </li>
<li> Valor Débito: <jsp:getProperty name="debito" property="valorDebitado"/> </li>
<li> Saldo Atual: <jsp:getProperty name="debito" property="saldo"/> </li>
<mais formatações html...>
```

FIG. 3. Aparência de um programa JSP.

Na Agência usou-se JSP para o desenvolvimento dos aplicativos que compõem o Gerenciador de Conteúdos, ou seja, a atualização da base de dados com os conteúdos da Agência e os metadados de recursos referenciados. Esses aplicativos são, basicamente, formulários que permitem inserção, alteração e exclusão de dados. Também foi utilizado JSP na construção do aplicativo Cesta de Documentos, do *sítio* da Agência, que tem um comportamento semelhante a um *bookmark* expandido, pois pode guardar os endereços eletrônicos de páginas de interesse do usuário mais seus metadados e pode ser baixado na máquina do cliente como um arquivo texto – só não é guardado diretamente no *browser*, para isso o usuário deve usar o *bookmark* do *browser* de navegação que estiver sendo utilizado.

PHP

PHP é um *script* de propósito genérico, que pode ser utilizado sozinho na máquina servidora, mas também pode ser embebido em código HTML, especialmente para criar páginas *web* dinâmicas; ou seja, utiliza-se *scripting tags* PHP em documentos HTML para executar funções específicas. Uma grande vantagem do PHP é sua compatibilidade com alguns sistemas gerenciadores de bancos de dados de domínio público, o que promove a facilidade de construir aplicações *web* que possam utilizar esses bancos.

A Fig. 4 ilustra o funcionamento de um programa PHP, que é um formulário HTML com as *scripting tags* PHP. Na máquina servidora, o formulário HTML resultante é gerado à medida que o *script* é interpretado; de modo que os dados sejam lidos e colocados nos seus respectivos lugares junto ao HTML, quando ocorre leitura de dados.

FIG. 4. Funcionamento de um programa PHP.

Supondo que os dados do cliente do banco, do exemplo anterior, estejam em uma base de dados MySQL (MySQL AB, 2004) e que um programa PHP fará o cálculo do débito e mostrará os resultados, a aparência desse programa poderia

[6] Relembrando, o foco deste trabalho é o que ocorre na camada *View*.

ser como a ilustrada na Fig. 5. Deve-se notar que todas as operações são programadas no código, desde a especificação de onde está a base de dados, senhas de acesso, especificação de expressão de busca na base de dados (comando "SELECT"), gravação do novo saldo (comando "UPDATE"), etc. O exemplo poderia ser melhorado, a fim de separar as funcionalidades programadas, isto é, colocando-as em funções que seriam parametrizadas e tratadas nesse código – o que seria feito por um programador; mesmo assim, sempre sobraria alguma complexidade relativa a detalhes de programação no código, ou seja, o *web designer* não o trabalharia sozinho; nem tampouco a separação *Model-Control-View* estaria bem clara em um aplicativo escrito dessa forma.

Na Agência usou-se PHP para implementar o aplicativo Busca do *site*, isto é, a montagem das expressões de busca e a formatação de seus resultados, dado que a busca efetivamente é realizada pela ferramenta *swish-e* (Cruz, 2003). Esse aplicativo PHP foi desenvolvido de modo que a ferramenta de busca possa ser trocada a qualquer momento, sendo totalmente parametrizado – e dividido em várias funções utilizadas ao longo do código, mescladas no HTML de apresentação (formatação de dados); o quê acaba por lhe incutir uma certa complexidade lógica.

```
<html> <title>.. <body>
<h1> Resultado da Operação Débito</h1>
<% // início do php
$db = mysql_connect("máquina","usuário", "senha");
mysql_select_db("nomeBase",$db);
$result0 = mysql_query("SELECT Agencia Conta NomeCliente Saldo FROM ClientesBanco
        WHERE Agencia=\"$agencia\" and conta=\"$conta\" ",$db)
        or die ("Agência ou conta inválida");
$SaldoAnterior = $result(3);
$Saldo = $result(3) - $debito;
// grava novo saldo
mysql_query("UPDATE ClientesBanco SET Saldo = \"$Saldo\"
        WHERE Agencia=\"$agencia\" and conta=\"$conta\" ",$db);
// imprime no HTML
printf("<li> Nome Cliente:",$result(2)); printf(" </li>");
printf("<li> Conta: ", $result(1)); printf(" </li>");
printf("<li> Agência: ", $result(0)); printf(" </li>");
printf("<li> Saldo Anterior: ", $SaldoAnterior); printf(" </li>");
printf("<li> Valor Débito: ", $debito); printf(" </li>");
printf("<li> Saldo Atual: ", $Saldo); printf(" </li>");
// fim do php
%>
  <mais formatações html...>
```

FIG. 5. Aparência de um Programa PHP.

XML - eXtensible Markup Language

XML é uma linguagem de marcação de documentos, de propósito geral, derivada de SGML - **S**tandard **G**eneralized **M**arkup **L**anguage. A SGML é a que fornece as regras genéricas para organizar e marcar elementos de um documento, desenvolvida e padronizada pela ISO - International Organization for Standards em 1986, que é a geratriz do HTML, XML e outras.

A XML permite a definição de *tags* próprias e é muito utilizada em trocas de dados não-formatados (World Wide Web Consortium, 2004a). Por exemplo, se esses dados vão ser colocados em um formulário HTML (como nos casos dos itens anteriores), eles precisam ser retrabalhados; pois, um dos objetivos de exportar dados em XML foi separar o "*design* gráfico" (*look and feel* da apresentação) dos dados propriamente ditos. Para formatar dados representados em XML é necessário passá-los por um processador XSL - eXtensible Stylesheet Language. XSL é uma linguagem para expressar folhas de estilo, que especificam como uma classe de documentos XML deve ser apresentada, descrevendo as transformações para apresentação de cada instância da classe (World Wide Web Consortium, 2004b) – as folhas de estilo contêm as regras de transformação, daí a necessidade de um processador para essas regras.

A Fig. 6 mostra como os dados relativos ao débito do cliente, gerados por uma classe JAVA ou outro programa qualquer, podem ser representados em XML (caixa verde). Note que esse resultado em XML é um texto não-formatado. Para formatá-lo como uma página HTML, precisa-se definir uma especificação em XSL da forma como o HTML resultante deve ser mostrado. Recebendo como entrada o XML e a especificação em XSL, a aplicação, escrita em JAVA ou outra linguagem, dispara um processador de XSL. A princípio a especificação XSL deveria ser escrita por um *web designer*, dado que apenas ela precisa ser mudada para que se obtenha uma apresentação diferente do mesmo conjunto de dados; porém sua sintaxe não é muito simples.

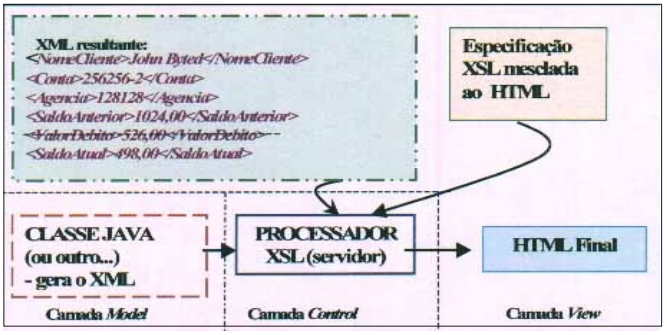

FIG. 6. Funcionamento de XML com XSL.

Para o exemplo dos dados relativos ao débito, uma formatação correspondente àquelas que foram dadas por PHP e JSP aos dados, seria a ilustrada na Fig. 7. Nesse exemplo, em particular, a formatação XSL é bastante simples, referenciando diretamente os campos do XML nas linhas em que eles devem ser escritos. Em geral, em uma aplicação real, no mínimo, seriam estabelecidos *templates* para cada campo, a fim de colocá-los em um laço de execução; e, todos esses detalhes de definição de *templates* e laços de execução não são exatamente simples, exigem um conhecimento razoável dos recursos de XSL.

Na Agência utilizou-se XML para gerar os dados relativos aos metadados dos recursos eletrônicos referenciados e/ou apresentados no *site*. Recursos eletrônicos são as próprias páginas *web* do *sítio* da Agência, outros *sites* e publicações eletrônicas citadas ou referenciadas na Agência, imagens ilustrativas, bases de dados, filmes, fitas de vídeo, etc. Essa solução foi adotada pois pretendia-se para exportar esses dados para outros sistemas; ou seja, as trocas de dados entre a Agência e outros sistemas deveria ocorrer via arquivos de dados especificados em XML. Dessa forma, existe uma especificação XSL, não-trivial, recheada de definições de *templates*, para formatar esses dados para o *sítio* da Agência.

```
<?xml version="1.0" encoding="ISO8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<!--
  É preciso especificar o padrão de codificação de caracteres do arquivo gerado,
    caso contrário será gerado um arquivo de saída contendo caracteres UNICODE
    (mais de um byte).
-->
<xsl:output encoding="ISO8859-1"/>
<xsl:template match="/">
  <HTML>
    <HEAD>
        <TITLE> Transação: <xsl:value-of select="@TRANSACAO"/> </TITLE>
      <LINK>
          <xsl:attribute name="href">../../AG01/estilos.css</xsl:attribute>
          <xsl:attribute name="type">text/css</xsl:attribute>
          <xsl:attribute name="rel">stylesheet</xsl:attribute>
      </LINK>
      <META>
          <xsl:attribute name="name">GENERATOR</xsl:attribute>
          <xsl:attribute name="content">GestorConteudoAgencia 1.1 Copyright 2003
            by Embrapa Informatica Agropecuaria (www.cnptia.embrapa.br)
            </xsl:attribute>
      </META>
      </HEAD>
      <BODY text="#000000" leftMargin="0" topMargin="0" marginheight="0"
                                        marginwidth="0" bgcolor="#FFFFFF">
        <h1> Resultado da Operação Débito</h1>
          <li> Nome Cliente: @NomeCliente </li>
          <li>  Conta: @Conta </li>
          <li>  Agência: @Agencia </li>
          <li>  Saldo Anterior: @SaldoAnterior </li>
          <li>  Valor Débito: @ValorDebito </li>
          <li>  Saldo Atual: @SaldoAtual </li>
          (mais formatações html...)
      </BODY>
  </HTML>
</xsl:template>
```

FIG. 7. Especificação do estilo em XSL.

Velocity

Velocity é uma ferramenta, escrita em JAVA, para gerar informação formatada a partir de um *template* (The Apache Software Foundation, 2004b). Ela possui uma poderosa linguagem de construção de *templates* para referenciar objetos definidos em código JAVA, com sintaxe simples. Ela tem sido muito utilizada em desenvolvimento de aplicações para *web*, pois é perfeitamente adaptável ao modelo MVC, mantendo o foco do *web designer* no *look and feel* e do programador na aplicação.

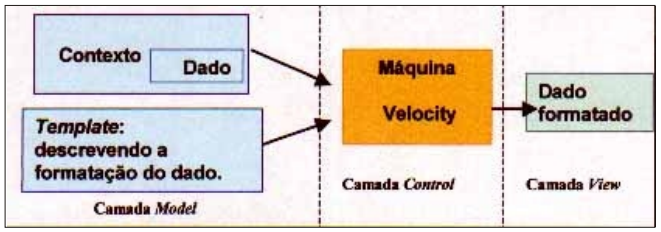

FIG. 8. Funcionamento da Velocity.

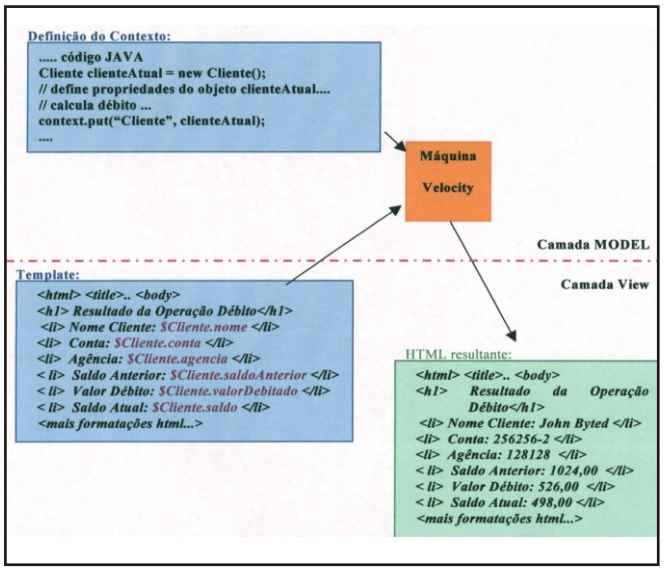

FIG. 9. Exemplo de *template* Velocity e resultado.

A máquina da Velocity recebe como entrada o *template* especificado e um objeto que define o contexto em que os dados devem ser utilizados; a definição do contexto (implementada pelo programador) faz o mapeamento entre as especificações colocadas no *template* (realizadas pelo *web designer*) e a geração dos dados produzindo o dado formatado, como ilustrado na Fig. 8. O dado formatado pode ser um arquivo texto qualquer: HTML, JSP, PHP, XML, XSL, um *script* qualquer, relatórios diversos, etc.; dado que o *template* é genérico. Para o exemplo dos dados relativos ao débito do cliente, ocorreria o ilustrado na Fig. 9. Note que o problema de recuperar os dados do cliente, calcular o débito, atualizar a base de dados, etc., foi deixado no código JAVA (faz parte da camada *Model*), cabendo ao *web designer* apenas a formatação dos dados no *template* (camada *View*); faz-se necessário apenas que programador e *web designer* definam o nome do objeto tratado e suas propriedades.

Na Agência foi construído um *template* Velocity para os hipertextos com os conteúdos apresentados no *site*. Como a variação desses conteúdos não é tão freqüente, elas são publicadas na *web* como páginas estáticas que são atualizadas sempre que necessário – ou seja, sua publicação é refeita quando ocorrem alterações (atualização de dados das Agências).

Situação de Troca da Interface da Agência

A evolução da versão 1.0 da Agência para a 1.1 foi caracterizada por uma pequena evolução do modelo de dados, alguma mudança na interface dos aplicativos da intranet, Gerenciador de Conteúdos, e especialmente pela troca total do *look and feel* do *sítio* da extranet. Para projetar essa nova interface houve a contratação de um *web designer* que trabalhou junto a um profissional de editoração de imagens e especialistas do domínio. No entanto, o trabalho de adaptação da nova formatação aos dados gerados pelo sistema foi realizado pela equipe de desenvolvimento do sistema. A Tabela 1 mostra o resumo das mudanças no código, pessoas envolvidas e tempo gasto nas atividades.

Tabela 1. Esforço de mudança de interface da Agência de Informação - Versão 1.1.

Mudança	*Módulo do Sistema*	*Número de técnicos*	*Perfil Técnico*	*Tempo*
Páginas estáticas de conteúdo	Template Velocity	1	Técnico sem conhecimento	1 semana
Apicativo busca	2 programas PHP	1	O próprio desenvolvedor dos programas	1 semana
Páginas com metadados dos recursos	Especificação XSL	1	O próprio técnico que especificou o XSL original	1 semana
Cesta de documentos	Programa JSP	1	Técnico com pouco conhecimento de JSP	1 semana

Devido às características das formas de definição dos módulos de interface e às dificuldades de cada módulo, além do fato de ter-se pouco tempo para efetivar as mudanças, que correspondia a uma única semana, a equipe foi dividida de modo a otimizar as curvas de aprendizado dos seus elementos; por isso, os técnicos com mais conhecimento nos módulos foram alocados nas tarefas. Inicialmente, tentou-se utilizar o mesmo técnico que trabalhou no *template* da Velocity para efetuar as alterações dos programas que implementam o aplicativo de busca em PHP. Um dos

programas PHP é relativamente complexo, desdobrando-se em várias funções parametrizadas, pois foi construído para que pudesse ser reconfigurado, porém não foi devidamente documentado. Desta forma, alocou-se o próprio desenvolvedor dos programas PHP para efetivar as alterações; evitando-se gastar tempo em estudo de código. Para a especificação XSL ocorreu o mesmo, pois esta exigia um bom conhecimento de XSL e da lógica de construção dos *templates* utilizados. No entanto, o código JSP e o *template* Velocity apresentaram-se bem mais simples, um pouco de conhecimento de JSP e apenas o entendimento dos campos no *template* Velocity foram suficientes; dado que tanto um como outro estavam apenas embutidos em uns formulários HTMLs – que eram os que efetivamente precisaram ser alterados.

Resumo Comparativo e Discussão

Ferramentas como JSP e PHP apresentam funcionalidades semelhantes, dado que as duas podem ser utilizadas para alimentar formulários HTML com conteúdos dinâmicos, provendo uma forma de conexão direta com um programa servidor. Optar pela escolha de uma ou outra vai depender da arquitetura de desenvolvimento de software, tamanho/complexidade da aplicação, grau de reuso desejado, conhecimento das ferramentas e tempo para execução do projeto. Por exemplo, na Tabela 2 são colocados alguns exemplos de escolhas e possibilidades de troca das escolhas – dentre o conjunto de soluções tecnológicas apresentado neste trabalho.

Ferramentas como XML/XSL e Velocity possuem uma ampla aplicação, dado que trabalham sobre *templates* genéricos que podem gerar qualquer arquivo texto: outro XML, HTML, SQLs, programas em linguagens de programação quaisquer, relatórios variados, etc.

O XML tem sido mais utilizado em trocas de dados entre sistemas, dada a sua flexibilidade de especificação de conteúdo, e é o padrão recomendado pela W3C (World Wide Web Consortium, 2004c); além disso é muito utilizado para especificar arquivos de configuração. Por exemplo, os arquivos de configuração das ferramentas do Projeto Jakarta (Tomcat, Struts, Ant, etc.) são todos especificados em XML. A dificuldade de uso do XML para camadas de apresentação de dados advém da dificuldade de especificar também o XSL que é o estilo de apresentação. No entanto, as duas linguagens de marcação, XML e XSL, foram criadas inicialmente com o objetivo de separar o conteúdo de dados do estilo de apresentação, ou seja, o propósito posterior, de troca de dados, é que se sobrepôs ao original.

A ferramenta Velocity, por outro lado, foi criada e é utilizada especialmente para implementar de forma adequada a camada *View* do padrão MVC. A formatação dos dados é dada por uma sintaxe bastante trivial e a sua integração aos programas JAVA, do ponto de vista do desenvolvedor, é bastante simples também; ambos os lados, programador e *web designer*, possuem facilidades de uso que não requerem grandes conhecimentos do funcionamento da ferramenta. A Tabela 3 mostra alguns exemplos de aplicação da Velocity e XML. Ainda, na Tabela 3 é citada a situação em que a XML foi utilizada em troca de dados na Agência, que é a especificação da árvore hiperbólica – componente gráfico de interface utilizado na navegação sobre a estrutura hierárquica da Agência. Essa é uma especificação textual, que pode ser lida e compreendida por um ser humano, além de ser mais facilmente armazenável em um banco de dados; e, embora seja um caso a parte da discussão aqui colocada, é também bastante interessante e sem alternativas de troca neste momento.

Tabela 2. Exemplos de uso de JSP e PHP.

Aplicativo	*Descrição*	*Solução*	*Motivo*	*Possível Troca*
Tabulação da Análise de Aspectos Comportamentais (uso interno da Embrapa Informática Agropecuária)	Questionário aplicado aos empregados sobre avaliação de aspectos comportamentais deles e de seus colegas	PHP com base de dados MySQL	Curto tempo para desenvolvê-lo; facilidade de comunicação e uso de MySQL sob PHP	JSP; porém aumentando a complexidade de tratamento do aplicativo e base de dados; implicando em maior tempo de desenvolvimento
Gerenciador de Conteúdos da Agência	Atualização de dados sobre a Agência de Informação Embrapa e publicação dos mesmos	Servlet (JAVA) controladora e JSP na camada *view*	Escolha de arquitetura a ser utilizada (J2EE), projeto razoavelmente grande, segurança de dados	PHP, porém a arquitetura teria que ser revista
Busca do Sítio Agência de Informação	Aplicativo que realiza a busca no *site* , com chamadas à ferramenta swish-e	PHP e swish-e (CRUZ, 2003)	Facilidade de integração com a swish-e, que deve ser executada como um *script* na máquina servidora (UNIX like)	JSP, complicando-se a forma de integração com a swish-e
Cesta do Sítio Agência de Informação	Aplicativo que funciona como um *bookmark* do sítio, auxiliando o usuário a guardar os endereços e metadados dos documentos que mais lhe interessam	JSP com definição de sessões para usuário	Facilidade de criar e gerenciar sessões, que são estruturas de dados associadas a um usuário com tempo de vida determinado	PHP, estudando-se uma forma de gerenciar de modo adequado as informações de um usuário e garantindo a segurança das mesmas

Tabela 3. Exemplos de uso de XML, XML/XSL e templates Velocity.

Aplicativo	*Descrição*	*Solução*	*Motivo*	*Possível Troca*
Publicação do Sítio Agência	Parte relativa aos conteúdos da Agência, um conjunto de hiperdocumentos estáticos	Template Velocity	Documentos estáticos de formatos repetitivos que são publicados apenas quando ocorrem atualizações	XML/XSL ou JSP, porém a complexidade seria bem maior nos dois casos
Página Mais Detalhes do Sítio Agência	Parte do Sítio relativa aos metadados dos documentos catalogados na Agência	XML/XSL	A idéia inicial era a troca de dados com outros sistemas, que acabou sendo descartada por hora	Template Velocity; facilitaria a manutenção, especialmente a troca de interface.
Especificação da Árvore Hiperbólica	A especificação de como deve ser apresentada a árvore do conhecimento da Agência, que é armazenada no banco de dados	Especificação XML	Os atributos da árvore são especificados em XML e lidos por uma ferramenta que os interpreta e transforma-os em uma representação gráfica	Sem alternativa, por hora

Essas observações e resultados têm como conseqüências imediatas o auxílio na definição das tecnologias que serão utilizadas na versão 2.0 das ferramentas de software da Agência de Informação Embrapa. Nessa versão, o uso de XML ficou restrito a processos de troca de dados. Na camada *View* expandiu-se o uso da Velocity e também de JSPs mais simples. Os JSPs utilizados possuem pouquíssimas *tags* e quase não há *scripts*, de modo que eles mais se assemelham a simples formulários HTML. Espera-se com isso facilitar a evolução de interface gráfica, podendo contar mais com a ajuda apenas de *web designers*, sem precisar dispor de pessoal de desenvolvimento nas futuras evoluções restritas a *look and feel*.

Conclusões

1. Do ponto de vista do uso do padrão de desenvolvimento MVC a melhor solução tecnológica utilizada, no desenvolvimento das ferramentas de software da Agência de Informação Embrapa, foi a Velocity.

2. Do ponto de vista do trabalho do *web designer*, também no mesmo contexto de desenvolvimento, a melhor solução tecnológica foi o uso da Velocity.

3. A solução tecnológica de JAVA Server Pages - JSP foi bastante satisfatória para os aplicativos da intranet.

4. O uso de XML, no contexto da Agência, teria sido mais útil apenas para trocas de dados.

Referências Bibliográficas

THE APACHE SOFTWARE FOUNDATION. **The Apache Jakarta Project**: The Jakarta *site* - Apache Jakart Tomcat. Disponível em: <http://jakarta.apache.org/tomcat/index.html>. Acesso em: 10 ago. 2004a.

THE APACHE SOFTWARE FOUNDATION. **The Apache Jakarta Project**: Velocity. Disponível em: <http://jakarta.apache.org/velocity/index.html>. Acesso em: 10 ago. 2004b.

BUSHMAN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; SATL, M. **Pattern-oriented software architeture**. West Sussex, UK: John Wiley, 1996. 468 p.

CRUZ, S. A. B. da. **Implantação de um serviço de busca em site da WWW**. Campinas: Embrapa Informática Agropecuária, 2003. 8 p. (Embrapa Informática Agropecuária. Comunicado Técnico, 50). Disponível em: <http://www.cnptia.embrapa.br/modules/tinycontent3/content/2003/comtec50.pdf>. Acesso em: 10 ago. 2004.

EVANGELISTA, S. R. M.; SOUZA, K. X. S. de; SOUZA, M. I. F.; BRAGA, S. A. C.; LEITE, M. A. de A.; SANTOS, A. D. dos; MOURA, M. F. Gerenciador de conteúdos da Agência Embrapa de Informação. In: INTERNATIONAL SYMPOSIUM ON KNOWLEDGE MANAGEMENT - ISKM=SIMPÓSIO INTERNACIONAL DE GESTÃO DO CONHECIMENTO, 2003, Curitiba. **Anais**... Curitiba: Pontifícia Universidade Católica do Paraná, 2003. p.1-12. Parte do CD-ROM.

MySQL AB. **MySQL**: the world's most popular open source database. Disponível em: <http://www.mysql.com/>. Acesso em: 10 ago. 2004.

SUBRAHMANYAM, A.; BUEST, C.; DAVIES, J.; JEWELL, T.; JOHNSON, R.; LONGSHAW, A.; NAGAPPAN, R.; SARANG, P. G.; TOUSSAINT, A.; TYAGI, S.; WATSON, G.; WILCOX, M.; WILLIAMSON, A.; O´CONNOR, D. **Professional Java server programming J2EE 1.3 edition**. Birmingham: Wrox Press, 2001. 1250 p.

WORLD WIDE WEB CONSORTIUM. **eXtensible Markup Language (XML)**. Disponível em: <http://www.w3.org/XML>. Acesso em: 10 ago. 2004a.

WORLD WIDE WEB CONSORTIUM. **The Extensible Stylesheet Language Family (XSL)**. Disponível em: <http://www.w3.org/Style/XSL/>. Acesso em: 10 ago. 2004b.

WORLD WIDE WEB CONSORTIUM. **W3C World Wide Web Consortium**. Disponível em: <http://www.w3.org/>. Acesso em: 10 ago. 2004c.

Comunicado Técnico, 62

Ministério da Agricultura, Pecuária e Abastecimento

Embrapa Informática Agropecuária
Área de Comunicação e Negócios (ACN)
Endereço: Caixa Postal 6041 - Barão Geraldo
13083-970 - Campinas, SP
Fone: (19) 3789-5743
Fax: (19) 3289-9594
e-mail: sac@cnptia.embrapa.com.br

1ª edição on-line - 2004