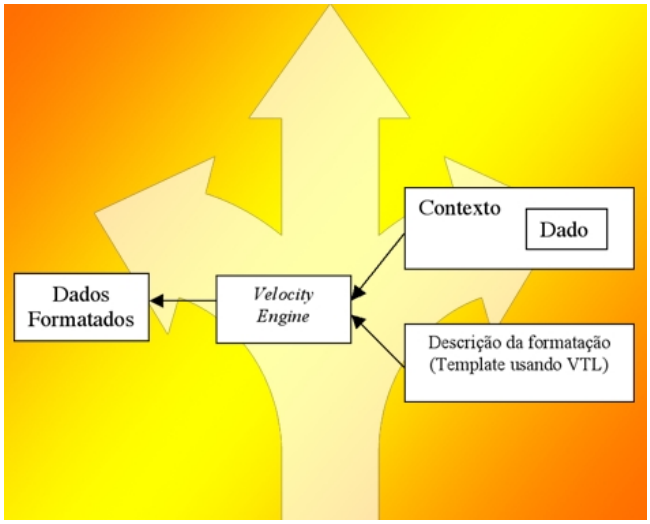
Ministério da Agricultura, Pecuária e Abastecimento

Comunicado Técnico 20

Outubro, 2002
Campinas, SP

ISSN 1677-8464



Formatação de Dados Usando a Ferramenta Velocity

Sergio Aparecido Braga da Cruz[1]
Maria Fernanda Moura[2]

Novas técnicas de desenvolvimento de software têm preconizado a divisão de sistemas em camadas especializadas que promovem uma maior flexibilidade na evolução, facilidade de manutenção e modularidade dos sistemas. Uma das camadas sempre presentes em todos os sistemas corresponde àquela relacionada a troca de informações, as quais têm como finalidade implementar a comunicação dos sistemas com um usuário final ou com outros sistemas. Para que esta comunicação possa ser eficiente, as informações trocadas precisam seguir um formato adequado. A implementação da camada de apresentação dos sistemas contam atualmente com ferramentas que facilitam o seu desenvolvimento. Nesta linha, surgiram as tecnologias XML - eXtensible Markup Language (Harold, 1999), JSP Java Server Pages (Juric et al., 2001), PHP (Fischer, 2000) e *Velocity* (Apache Software Foundation, 2002b).

A tecnologia XML tenta ser uma resposta genérica para o problema de formatação de dados. JSP e PHP são soluções para formatação voltadas principalmente para a implementação de aplicações no ambiente Web.

A *Velocity* surgiu da necessidade de uma solução mais simples e eficiente, utilizando a linguagem Java (Sun Mycrosystems, 2002), para o desenvolvimento de aplicações para o ambiente WWW (World Wide Web) (Krol, 1993) ou mais simplesmente *Web*. Apesar de sua simplicidade, pode ser utilizada nas mais diversas aplicações onde é necessária a formatação de dados textuais.

Neste documento é descrito o funcionamento da *Velocity* e um caso de uso dentro do projeto Agência (Embrapa, 1998).

Origem e características

A ferramenta *Velocity* é umas das soluções *open source* criadas sob o Projeto Jakarta (Apache Software Foundation, 2002a), o qual tem como objetivo criar e manter soluções sem custo na plataforma Java para o público em geral. Sua concepção inicial enfatiza a organização de sistemas interativos utilizando a arquitetura MVC (Model-View-Control) (Buschmann et al., 1996), os quais definem as seguintes camadas para um sistema:

- *Model* – relacionado ao processamento e geração de dados pelo sistema
- *View* – relacionado a apresentação dos dados
- *Control* – relacionada a lógica de controle da camada de processamento (*Model*)

A atuação básica da ferramenta *Velocity* consiste na formatação textual de dados, ou seja, na camada de

[1] *Mestre em Engenharia Elétrica, Pesquisador da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo – 13083-970 – Campinas, SP. (e-mail: sergio@cnptia.embrapa.br)*

[2] *Mestre em Engenharia Elétrica, Pesquisadora da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo – 13083-970 – Campinas, SP. (e-mail: fernanda@cnptia.embrapa.br)*

apresentação de dados dos sistemas (*View*). O exemplo mais comum deste tipo de aplicação ocorre na geração dinâmica de páginas na *Web*. Devido a versatilidade de seu projeto, a ferramenta *Velocity* pode também ser utilizada nas mais variadas aplicações, tais como, geração de código fonte SQL (*Structured Query Language*), *PostScript*, XML, relatórios, etc. Pode ser utilizada sozinha ou integrada com outras ferramentas de um sistema.

Arquitetura

A ferramenta *Velocity* esta organizada como ilustrado na Fig. 1. Os dados formatados na forma textual são gerados a partir dos dados brutos e de uma descrição da formatação. Os dados brutos devem estar encapsulados em classes Java e serem acessíveis por meio de uma interface pública (métodos públicos). A ferramenta *Velocity* infere, em tempo de execução, esta interface pública e através dela pode realizar a recuperação dos dados armazenados nas classes.

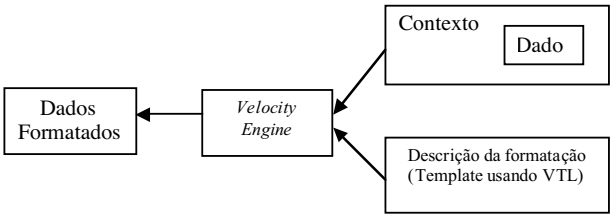


Fig. 1. Arquitetura da ferramenta *Velocity*.

A descrição da formatação deve ser realizada por meio da linguagem VTL (*Velocity Template Language*). A VTL é uma linguagem simples, contando com poucos comandos e que permite a definição de variáveis, controle de fluxo de execução, inclusão de *templates* e definição de macros. Esta simplicidade facilita a aprendizagem e utilização da VTL pelo *designer* de páginas, ao mesmo tempo em que o isola da parte de tratamento dos dados, os quais são manipulados pela aplicação principal em Java e implementados pelo programador da aplicação. Desta maneira a *Velocity* facilita o trabalho colaborativo e independente de programadores e *designers*.

O mapeamento dos dados da aplicação em Java a serem utilizados no *template* é realizado por meio de um contexto. O contexto é implementado pela classe *org.apache.velocity.VelocityContext* do pacote da *Velocity*, e a sua função é nomear os objetos da aplicação para que eles possam ser referenciados no *template*. Os nomes têm a forma de uma string Java (Fig. 2).

Contexto	
Nome	Objeto da aplicação
"cesta"	cesta 2
"cliente"	cliente1
"produto"	prod1

Fig. 2. Exemplo de contexto.

Com o mapeamento definido, o interpretador da *Velocity* poderá interpretar o *template*, e nesta interpretação, utilizará os nomes definidos no contexto para identificar e acessar dados dos objetos. No *template* os nomes definidos no contexto são expressos como referências. A referência é identificada por uma sintaxe especial da VTL e através dela os dados dos objetos Java podem ser acessados. A Fig. 3 ilustra o mapeamento entre dados na aplicação e suas referências no *template* em VTL.

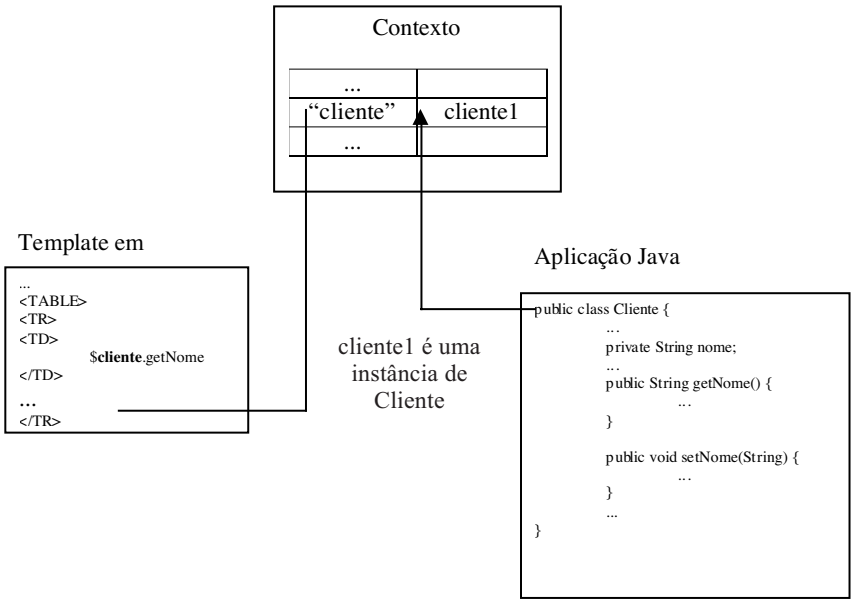


Fig. 3. Esquema de mapeamento de dados.

Funcionamento e esquema de uso

Durante uma execução normal da ferramenta *Velocity*, são realizados os seguintes passos:

- A *Velocity* é inicializada, carregando uma configuração que definirá algumas características de seu funcionamento.
- Um esquema de mapeamento entre referências no arquivo de formatação em VTL e as classes Java é especificado (criação do contexto).
- A *Velocity* carrega template especificado.
- O *template* é analisado e armazenado em memória na forma de uma estrutura de dados conhecida como AST (*Abstract Syntax Tree*).
- A representação AST é então utilizada sempre que for necessária a formatação de dados, realizando a substituição adequada onde necessário das referências pelos valores obtidos dos objetos Java, de acordo com as regras de mapeamento definidas pelo contexto.

Em aplicações *Web* um esforço inicial deve ser realizado na definição das informações necessárias para apresentação final das páginas geradas dinamicamente através da *Velocity*. Neste ponto um padrão deve ser definido para nomenclatura de referências, e deste modo tanto o *designer* de páginas quanto os programadores poderão trabalhar de maneira independente e em paralelo. Este trabalho então pode ser dividido em duas tarefas principais, as quais correspondem a criação do *template* usando a VTL e a construção da aplicação Java com a chamada dos métodos adequados da API (*Application Program Interface*) da ferramenta *Velocity* para sua execução.

Linguagem VTL

A linguagem para descrição de *templates* VTL possui poucos recursos, sendo simples e fácil de assimilar. Todos os elementos da linguagem estão descritos na Tabela 1, onde, na coluna sintaxe, elementos entre colchetes indicam elementos opcionais. A linguagem VTL funciona embutida no arquivo texto e o seu princípio de funcionamento está baseado na inclusão de diretivas e variáveis num arquivo texto qualquer formando um modelo ou *template*.

As variáveis contêm os dados que deverão ser apresentados em tempo de execução. Variáveis, propriedades e métodos estão intimamente ligados ao acesso a objetos Java através do mapeamento definido pelo contexto. Uma variável pode ser definida no *template* através da diretiva *#set* ou pode estar associada ao contexto. O uso de variáveis que não tenham sido definidas no *template* implica na apresentação da expressão invocando o valor da variável diretamente no documento sendo gerado. Este comportamento pode ser contornado utilizando a notação silenciosa para acesso a variável, e, neste caso, nada será apresentado. A notação formal deve ser utilizada quando o uso da notação normal implicar em ambigüidade na identificação da variável a ser utilizada.

Para controle de fluxo de execução a linguagem VTL conta com as diretivas *#if*, *#foreach* e *#stop*. O comando *#if* tem funcionamento análogo ao de outras linguagens de programação. Sendo que sua condição é considerada verdadeira se for igual ao valor booleano *true* ou um valor não booleano diferente de *null*. O comando *#foreach* deve ser utilizado para execução de iteração na VTL. Todos os elementos necessários para iteração estão definidos como argumentos do *#foreach* e desta forma não existe possibilidade de um *loop* infinito como poderia ocorrer se existisse um comando do tipo *while*. O comando *#stop* finaliza a interpretação de *templates* e é muito utilizado em depurações.

Com a finalidade de organização e reuso de *templates* existem diretivas para inclusão *#include* e *#parse*. A diretiva *#include* inclui um outro arquivo texto sem interpretá-lo como um arquivo em VTL. O diretiva *#parse* inclui um arquivo interpretando-o como um arquivo VTL.

Existe também a diretiva *#macro* que permite a definição de novas diretivas semelhantes a subrotinas. Isto pode ser útil para efeito de organização de código, eliminação de repetições de trechos num mesmo *template* ou montagem de uma biblioteca de macros contendo operações repetitivas.

Duas diretivas (*##* e *#** <comentário> **#*) definem comentário, os quais não são interpretados pela *Velocity* e também não são enviados para o resultado sendo gerado.

Tabela 1. Elementos da linguagem VTL.

	Sintaxe	*Exemplos*
Identificador	Qualquer seqüência de caracteres alfabéticos, números, - (hífen) ou _ (sobrescrito), começando por um caracter	var-Sa_9 a123 abx
Definição de variáveis	Identificador precedido por um $ (cifrão). Uma exclamação (!) pode seguir o cifrão indicando notação silenciosa. O identificador pode estar entre chaves indicando notação formal	$var1-Sa_9 (notação formal) $!var1-Sa_9 (notação silenciosa) ${var-Sa_9} (notação formal)
Propriedades	Seqüência de identificador, ponto (.), identificador precedido por um cifrão ($). Uma exclamação (!) pode seguir o cifrão indicando notação silenciosa. A seqüência de identificador, ponto (.), identificador pode estar entre chaves indicando notação formal	$customer.Address (notação regular) ${purchase.Total} (notação formal)
Métodos	Seqüência de identificador, ponto (.), identificador e abre e fecha parênteses. Entre os parênteses podem existir uma lista de argumentos separados por vírgula. Cada argumento pode ser ou uma variável ou uma string. Toda esta seqüência de termos deve ser precedida por um cifrão ($). Chaves podem ser usada para representar a notação formal.	$customer.getAddress() (notação regular) ${purchase.getTotal()} (notação formal) $page.setTitle("My Home Page") (notação regular com parâmetros)
Diretivas		
Atribuição	#set($ref = [", ']arg[", '])	#set($monkey = "bill") #set($monkey.Friend = "monica") #set($monkey.Blame = $whitehouse.Leak) #set($monkey.Plan = $spindoctor.weave($web)) #set($monkey.Number = 123) #set($monkey.Numbers = [1..3]) #set($monkey.Say = ["Not", $my, "fault"])
Controle de fluxo de execução	#if (condição) ... [#elseif (condição) ...]* [#else ...] #end	Igual a: #if($foo == $bar) Maior que: #if($foo > 42) Menor que: #if($foo < 42) Maior ou igual a: #if($foo >= 42) Menor ou igual a: #if($foo <= 42) Igual a: #if($foo == 42) Igual a: #if($foo == "bar") Não booleano : #if(!$foo)
	#foreach($ref in arg) ... #end	<table> #foreach($produto in $produtoList) <tr><td>$velocityCount</ td><td>$produtoNome</td></tr> #end </table>
	#stop – Stops the template engine	#stop
Inclusão de arquivo	#include(arg[, arg2, ... argn])	#include("email.txt", "teste.txt") #include($var1, $var2)
	#parse(arg)	#parse("tabela.vm") #parse($var1)
Definição de macros	#macro(nomedamacro $arg1[, $arg2, $arg3, ... $argn]) < código em VTL ... > #end	#macro(linhas $cor $lista) #foreach($item in $lista) <tr><td bgcolor=$cor>$item</td></tr> #end #end
Invocação de macros definidas	#nomedamacro($arg1 $arg2)	#set ($c = "red") #set ($produtos = ["abacaxi","banana","caju","damasco"]) #linhas($c $produtos)
Comentário	## This is a comment. #* This is a multiline comment. This is the second line *#	

Aplicação Java

O acesso a todas as funcionalidades da ferramenta *Velocity* deve ser realizado através da API definida pelas classes *org.apache.velocity.app.Velocity*, e/ou *org.apache.velocity.app.VelocityEngine*. A classe *org.apache.velocity.app.Velocity* possibilita o uso da *Velocity* seguindo o modelo *Singleton* (Gamma et al.,1995), onde uma única instância do objeto *Velocity* é compartilhada por toda a aplicação Java. Este modelo é adequado quando se requer uma configuração centralizada e compartilhamento de recursos (sistema de *log*, *templates*, etc.). Um esquema básico de código Java para uso da *Velocity* neste modelo seria:

```
import org.apache.velocity.app.Velocity;
import org.apache.velocity.Template;
...
//  Configura a Velocity
Velocity.setProperty(Velocity.RUNTIME_LOG_LOGSYSTEM,
this);

//  Inicializa
Velocity.init();
...
// Acesso ao template
Template t = Velocity.getTemplate("template.vm");
...
```

A classe *org.apache.velocity.app.VelocityEngine* possibilita a criação de várias instâncias independentes da *Velocity*, cada uma com a sua configuração. Um esquema básico de código Java para uso da Velocity como instância independente seria:

```
import org.apache.velocity.app.VelocityEngine;
import org.apache.velocity.Template;
...

// Cria uma nova instância do engine
VelocityEngine ve = new VelocityEngine();

// Configura instância local
ve.setProperty(
VelocityEngine.RUNTIME_LOG_LOGSYSTEM, this);

// inicializa
ve.init();
...
Template t = ve.getTemplate("teplate.vm");
...
```

Para a interpretação propriamente dita com a substituição adequada das referências no *template* por valores de objetos Java deve ser criado um contexto e invocada a interpretação, como esquema-tizado a seguir:

```
VelocityContext contexto = new VelocityContext();

contexto.put("nome", "Velocity");
contexto.put("projeto", "Jakarta");

/* Trata o template */
StringWriter strw = ne
w StringWriter();

Velocity.mergeTemplate("testtemplate.vm", contex
to, strw );
```

O resultado da interpretação de um *template* é armazenado num objeto do tipo Writer, a partir do qual poderá ser enviado para um arquivo, apresentado ao usuário como documento formatado, armazenado em base de dados, etc.

Existem 4 maneiras de invocar a interpretação do *template*:

- *evaluate(Context context, Writer out, String logTag, String instring)*

  evaluate(Context context, Writer writer, String logTag, InputStream instream)

  Permite o tratamento *templates* armazenados em *String´s* ou *InputStream´s* do Java.

- *invokeVelocimacro(String vmName, String namespace, String params[], Context context, Writer writer)*

  Permite chamada de macros definidas em templates *Velocity*.

- *mergeTemplate(String templateName, Context context, Writer writer)*

  Maneira indicada e mais eficiente para execução de *templates* armazenados em arquivos.

Configuração da Velocity

Existe uma série de parâmetros de configuração da *Velocity* que podem ser utilizados, os quais determinam o comportamento da *Velocity* durante o tratamento de *templates*. Todos os parâmetros têm um valor *default*, que pode ser substituído pelo valor mais adequado pelo usuário da classe. Estes parâmetros somente se

tornam efetivo após a execução do método *init()*. A definição dos parâmetros de configuração para a *Velocity* pode ser realizada utilizando um dos seguintes esquemas:

- Chamando o método *init(String nomedearquivo)* ou *init(Properties propriedades)*, passando um arquivo de configuração ou objeto *Properties* respectivamente. O arquivo de configuração segue a sintaxe de leitura da classe *Properties*. Os valores não definidos no arquivo receberão valores *default*.
- Usar métodos *setProperty(String parametro, Object valor)* para definir valores de parâmetros individualmente e posteriormente chamar *init()*.

Exemplo:

```
VelocityEngine ve = new VelocityEngine();
Properties p = new Properties();
p.setProperty(VelocityEngine.FILE_RESOURCE_LOADER_PATH,
"/homedir/dirtemplates");
ve.init(p);
```

No exemplo apresentado é definido um novo local a partir do qual os *templates* podem ser carregados.

Exemplos de parâmetros disponíveis na ferramenta *Velocity*.

Constante Java	Parâmetro de configuração	Descrição
FILE_RESOURCE_LOADER_PATH	File.resource.loader.path	Indica o path para recursos (templates), o valor default é o diretório corrente
RESOURCE_LOADER	resource.loader	declara um carregador de recurso, default é file
FILE_RESOURCE_LOADER_CACHE	file.resource.loader.cache	Indica se será usado recurso de cache, valor default é false
OUTPUT_ENCODING	output.encoding	Indica o esquema de codificação de caracteres dos dados de saída
INPUT_ENCODING	input.encoding	Indica o esquema de codificação de caracteres dos dados de entrada
COUNTER_NAME	directive.foreach.counter.name	Indica o nome da variável automaticamente criada para informar o número de interações da diretiva #foreach. Default = velocityCount
COUNTER_INITIAL_VALUE	directive.foreach.counter.initial.value	Indica o valor inicial da variável associada ao contador da diretiva #foreach. Default = 1

Além destes, existem um série de outros parâmetros que podem ser utilizados e que estão descritos no manual da ferramenta *Velocity.*

Trecho de código fonte Java exemplificando o uso da ferramenta Velocity

```
...
public void gerarPaginasHTMLArvore(String idArvoreCon) throws PubAgenciaException {

        PagEstat   pE          = new PagEstat();
        ParteNoh  pnoh         = new ParteNoh();
        String                 enderecoOrigem = "";
        ConfiguraPag           configuracaoPagina  = new ConfiguraPag();

        configuracaoPagina.setEnderecoMaisDetalhes("../../"+AgConfig.getCatalogoDir()+"/"+AgConfig.getRecEletronicoDir()+"/");
        configuracaoPagina.setEnderecoCesta(AgConfig.getUrlCesta()); // com .jsp no final
        configuracaoPagina.setEnderecoPaginas(AgConfig.getPaginasEstaticasRaiz());
        configuracaoPagina.setEnderecoPaginasArvore(AgConfig.getAgenciaCodebase()+"/"+idArvoreCon+"/arvore");

        configuracaoPagina.setEnderecoBuscaSimples("../busca.php3");
        configuracaoPagina.setEnderecoBuscaAvancada("../../busca/buscaavancada.php3?aliasag="+idArvoreCon); // soh desvio
        configuracaoPagina.setEnderecoApplet(idArvoreCon+".html");

        String endereco = "";
        String enderecoPaginas = AgConfig.getPaginasEstaticasRaiz();

        ...

        try {
                File fdir=new File(enderecoPaginas);
                if(!fdir.exists())
                {
                        if(!fdir.mkdirs()) {
                                throw new PubAgenciaException("Nao conseguiu criar diretório para gerar as páginas estáticas da
árvore");
                                }
                        }
                        // Usando a Velocity para gerar as páginas

                        VelocityEngine ve = new VelocityEngine();

                        Properties p = new Properties();

                        p.setProperty(VelocityEngine.FILE_RESOURCE_LOADER_PATH, AgConfig.getTemplatePath());
                        p.setProperty(VelocityEngine.RUNTIME_LOG,AgConfig.getVelocityLogFile());
                        ve.init(p);

                        //  pegando o template
                        Template t = ve.getTemplate(AgConfig.getTemplate());

                        while(pE.criaProximaPagina(idArvoreCon)) {
                                pnoh = pE.getPartePagina();

                                // criação do contexto

                                VelocityContext context = new VelocityContext();

                                // criação de dados e trasnferências dos mesmos para o contexto

                                context.put("ConfiguraPag", configuracaoPagina);
                                context.put("ParteNoh", pnoh);
                                context.put("Linha1Menu", pE.getLinha1Menu());
                                context.put("ColunasMenu", pE.getColunasMenu());

                                // utilizando o merge do contexto com um writer.

                                StringWriter writer = new StringWriter();
                                t.merge( context, writer );

                                // imprimindo resultado
                                // System.out.println( writer.toString() );

                                try {
                                        endereco = enderecoPaginas + (pE.getIdConteudo()).trim() + ".html";
                                        FileOutputStream os = new FileOutputStream(endereco);
                                        PrintStream ps = new PrintStream(os);
                                        DataOutputStream ods = new DataOutputStream(os);
                                        ods.flush();
                                        ods.writeBytes(writer.toString());
                                } catch (Exception e) {
                                        throw new PubAgenciaException ("Erro ao gerar HTML:" + e.toString());
                                }
                        }
                }
                catch (Exception e) {
                        throw new PubAgenciaException ("Problema ao montar páginas: "+e.getMessage());
                }
        }
```

Trecho de *template* em VTL relativo ao item anterior

```
<html>
<head>
<title>N&oacute; da &aacute;rvore</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<script language="JavaScript">
<!—
function MM_swapImgRestore() { //v3.0
  var i,x,a=document.MM_sr; for(i=0;a&&i<a.length&&(x=a[i])&&x.oSrc;i++) x.src=x.oSrc;
}
...
//—>
</script>
</head>

<body bgcolor="#FFFFFF" text="#000000"
...

<table width="501" border="0" cellpadding="0" cellspacing="0">
  <tr>
  <div align="left">
     <td width="501" height="83" valign="top">
          <font color="#333333">
                    <font face="Verdana" size="2">
                    <font color="#003366">
                    <b>Refer&ecirc;ncias:</b>
                    </font>
                    </font>
                    </font>
     </td>
  </div>
#foreach ($recurso in $ParteNoh.recursosReferenciados)
<table width="501" border="0" cellpadding="0" cellspacing="0">
  <tr>
            <td width="501" height="83" valign="top">
            <font color="#333333">
            <font face="Verdana" size="2">
            <font color="#003366">
                    #if ($recurso.ehEletronico && $recurso.ehCatalogado)
                    <a href="$recurso.referencia" target="_blank">$recurso.titulo</a>
      #else
                    $recurso.titulo
      #end
            $recurso.resumo
            </font></font></font>
            #if ($recurso.ehCatalogado)
                    <font color="#FF0000" face="Verdana" size="1">
                    <a href="$recurso.linkMaisDetalhes" target="_blank">Mais Detalhes</a>
                    </font>
            #elseif(!$recurso.ehEletronico)
                    <font color="#FF0000" face="Verdana" size="1">
                    <a href="$recurso.linkMaisDetalhes" target="_blank">Mais Detalhes</a>
                    </font>
                     #end
                     <font color="#FF0000" face="Verdana" size="2">
            <form action="$ConfiguraPag.enderecoCesta" method="post">
                    <input type="hidden" name ="id" value = "$recurso.idRecEletronico">
                    <input type="hidden" name = "descricao" value = "$recurso.descricao">
                    <input type="hidden" name = "contexto" value = "$ParteNoh.contexto" >
                    <input type="hidden" name = "acao" value = "adicionar">
                    <input type="hidden" name = "tipo" value = "$recurso.tipo">
                    <input type="hidden" name = "titulo" value = "$recurso.titulo">
                    #foreach ($identificador in $recurso.identificadores)
                                        <input type="hidden" name ="identificador" value = "$identificador.endereco">
                        #end
                        <input type="image" src  = "../imagens/carrin.gif">
            </form>
     </font>
     </td>
  </tr>
</table>
#end

</body>

</html>
```

Considerações finais

A ferramenta *Velocity* foi utilizada na fase de Publicação do Sistema Agência, onde um conjunto de dados obtidos por consulta a uma base de dados e estruturados na forma de uma árvore em XML é organizado e apresentado na forma de conjunto de páginas HTML *(HyperText Markup Language)*. Estas páginas estão ligadas entre si por intermédio de um conjunto de *hyperlinks* permitindo que possam ser "navegadas". Estas páginas formam um *site* sobre um determinado domínio que é um dos produtos gerados pelo Sistema Agência.

A abordagem inicial, na construção das páginas, não utilizava nenhuma ferramenta adicional além da linguagem Java. As expressões HTML eram armazenadas em *strings* Java, e fixas no código da aplicação. Isto implicava em uma série de problemas:

- dificuldade de transposição de *layouts* de páginas HTML para a aplicação. À medida que as páginas HTML se tornavam mais elaboradas e adequadas do ponto de vista de interface e navegabilidade, a complexidade das expressões HTML aumentava, e a incorporação destas expressões no código Java se tornava mais difícil;
- dificuldade de manutenção das páginas HTML. Uma vez que o código HTML estivesse transposto para a aplicação Java, se tornou difícil corrigir trechos HTML, dentro de um código Java;
- distância do *design* de páginas dos dados gerados pela aplicação. Devido a uma falta de interface computacional embutida no *layout* das páginas HTML, o *designer* das páginas gerava um texto HTML plano, sem nenhuma indicação de onde os trechos dinâmicos deveriam ser inseridos pelos programadores. Isto dificulta a transposição dos *layouts* para o código Java, pois o programador necessita conhecer as intenções do *designer* além de HTML. Este é um fator importante facilitando a ocorrência de erros na geração das páginas HTML;
- dificuldade de manutenção da aplicação. A mistura de trechos HTML e código Java diminui a legibilidade do código e conseqüentemente a sua manutenção, principalmente após o uso de rotinas em *JavaScript* na páginas HTML geradas.

O uso da ferramenta *Velocity* na implementação desta funcionalidade da aplicação, resolveu estes problemas, através da separação do código Java do código HTML, agora armazenados em *templates*. A manutenção tanto do código Java quanto do código HTML ficou facilitada. A facilidade de entendimento da linguagem VTL possibilita que o *designer* possa manipular diretamente os *templates*, e neles fica claro onde serão inseridos os conteúdos dinâmicos, diferentemente de um arquivo HTML plano. Como vantagem da adoção da *Velocity*, surgiu a necessidade natural da criação de classes para dar suporte a criação do contexto. Esta camada induz a separação entre dados e apresentação na aplicação.

Diante destes resultados, dentro do projeto, estuda-se a possibilidade de substituição de outros trechos de código onde existe a necessidade de geração de páginas estáticas em formato HTML. Atualmente, em alguns trechos no sistema, a geração está sendo realizada utilizando-se a tecnologia XML. Dados são consultados na base de dados e transformados em XML, que por sua vez são transformados para HTML por meio da XSLT (*Extensible Stylesheet Language Transformations*). Apesar destas transformações já enfatizarem a separação dos dados de sua apresentação através do estruturação dos dados em XML, esta solução tem se mostrado lenta e complexa se comparada ao uso da *Velocity*. A linguagem utilizada pela XSLT , apesar de mais poderosa, é muito mais complexa que as diretivas da VTL. Dificilmente a XSLT poderia ser utilizada diretamente por um *designer* de páginas. A transformação XML em HTML por meio da XSLT é um processo lento. Além de facilitar a manutenção do *layout* de páginas HTML através do uso de *templates*, com a utilização da *Velocity* a geração do arquivo HTML diminuirá um passo, pois os dados não precisarão ser transformados em XML antes de serem formatados.

A ferramenta *Velocity* se mostrou adequada na solução do problema de formatação de dados textuais no Sistema Agência e implicou num mínimo impacto de implementação, uma vez que poucas linhas de código são necessárias para incorporá-lo no código já existente.

Referências bibliográficas

APACHE SOFTWARE FOUNDATION. The Apache Jakarta Project: The Jakarta site. Disponível em: <http://jakarta.apache.org>. Acesso em: 28 out. 2002a.

APACHE SOFTWARE FOUNDATION. The Apache Jakarta Project: Velocity. Disponível em: <http://jakar'ta.apache.org/velocity/index.html>. Acesso em: 28 out. 2002b.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. Pattern-oriented software architeture: a system of patterns. New York: John Wiley, 1996. 467 p. (Wiley Series in Software Design Patterns, 1).

EMBRAPA. Serviço de Produção de Informação. Projeto 1998-2002 Agencia de produtos e servicos de informacao. Brasilia, 1998. 89 p.

FISCHER, H. G. PHP: guia de consulta rápida. [S.l.]: Novatec, 2000. 128 p.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design patterns: elements of reusable object-oriented software. Reading: Addison Wesley, 1995. 395 p. (Addison-Wesley Professional Computing Series).

HAROLD, E. R. XML bible. Foster City: IDG Books Worldwide, 1999. 1015 p.

JURIC, M. B.; BASHA, S. J.; LEANDER, R.; NAGAPPAN, R. Professional J2EE EAI. Birmingham: Wrox, 2001. 928 p.

KROL, E. The whole internet: user's guide & catalog. Sebastopol: O'Reilly, 1993. 376 p.

SUN MICROSYSTEMS. The source for Java technology. Disponível em: <http://java.sun.com>. Acesso em: 28 out. 2002.

Comunicado Técnico, 20

Embrapa Informática Agropecuária
Área de Comunicação e Negócios (ACN)
Av. André Tosello, 209
Cidade Universitária - "Zeferino Vaz"
Barão Geraldo - Caixa Postal 6041
13083-970 - Campinas, SP
Telefone (19) 3789-5743 - Fax (19) 3289-9594
e-mail: sac@cnptia.embrapa.br

1ª edição
2002 - on-line

Expediente

Supervisor editorial: *Ivanilde Dispato*
Normalização bibliográfica: *Marcia Izabel* Fugisawa Souza
Capa: *Intermídia Publicações Científicas*
Editoração Eletrônica: *Intermídia Publicações Científicas*