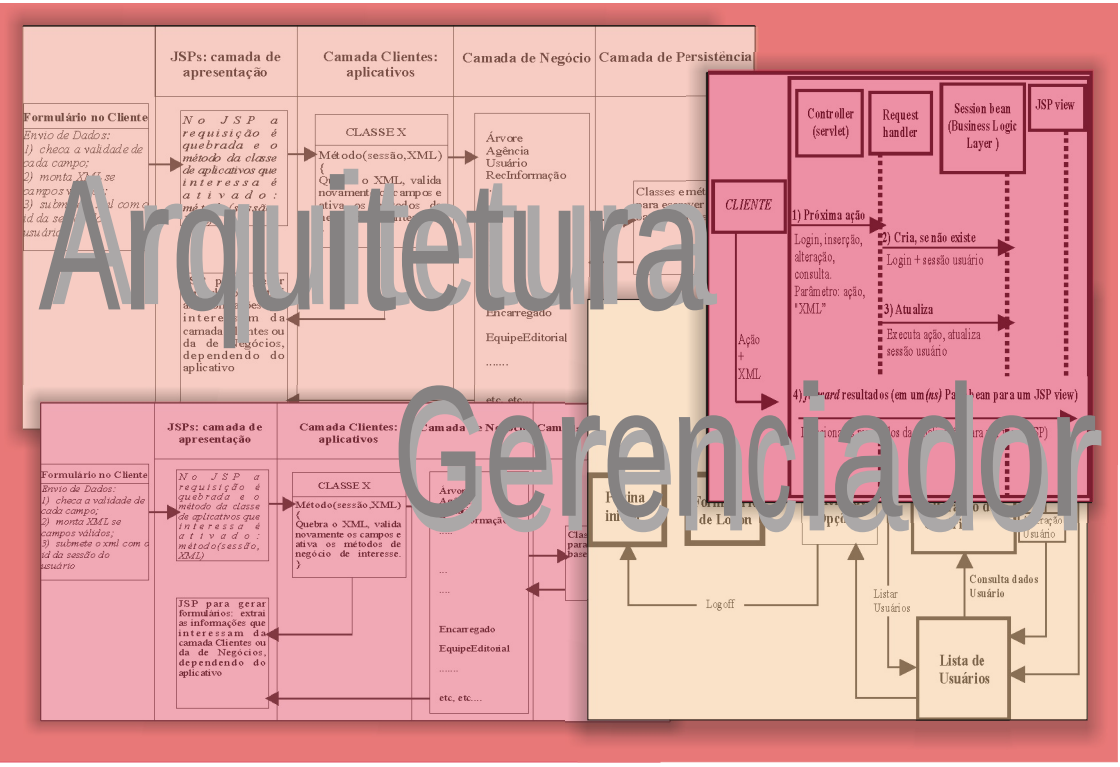
Documentos

Outubro, 2003 32

ISSN 1677-9274

Estudos para Melhoria da Arquitetura do Gerenciador de Conteúdos da Agência de Informação Embrapa



Embrapa

ISSN 1677-9274
Outubro, 2003

Empresa Brasileira de Pesquisa Agropecuária
Embrapa Informática Agropecuária
Ministério da Agricultura, Pecuária e Abastecimento

Documentos 32

Estudos para Melhoria da Arquitetura do Gerenciador de Conteúdos da Agência de Informação Embrapa

Maria Fernanda Moura
Sérgio Aparecido Braga da Cruz

Campinas, SP
2003

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
URL: http://www.cnptia.embrapa.br
e-mail: sac@cnptia.embrapa.br

Supervisor editorial: *Ivanilde Dispato*
Normalização bibliográfica: *Marcia Izabel Fugisawa Souza*
Editoração eletrônica: *Área de Comunicação e Negócios (ACN)*

1ª. edição on-line - 2003

Moura, Maria Fernanda.
Estudos para melhoria da arquitetura do gerenciador de conteúdos da Agência de Informação Embrapa / Maria Fernanda Moura, Sérgio Aparecido Braga da Cruz. — Campinas : Embrapa Informática Agropecuária, 2003.
24 p. : il. — (Documentos / Embrapa Informática Agropecuária ; 32)

ISSN 1677-9274

1. Ferramenta Agência. 2. Agência de Informação Embrapa. 3. Gerenciador de conteúdos. I. Cruz, Sérgio Aparecido Braga da. II. Título. III. Série.

CDD - 004.678 21st ed.

Autores

Maria Fernanda Moura
M.Sc. em Engenharia Elétrica, Pesquisadora da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo
13083-970 - Campinas, SP.
Telefone (19) 3789-5799
e-mail: fernanda@cnptia.embrapa.br

Sérgio Aparecido Braga da Cruz
M.Sc. em Engenharia Elétrica, Pesquisador da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo
13083-970 - Campinas, SP.
Telefone (19) 3789-5731
e-mail: sergio@cnptia.embrapa.br

Apresentação

Este trabalho sintetiza um estudo sobre tecnologias de informação, visando analisar seus potenciais para uso em projetos de pesquisa e desenvolvimento da Embrapa Informática Agropecuária. Esses estudos poupam, por muitas vezes, o trabalho de busca e análise das mesmas tecnologias no contexto de outros projetos, pois fornecem parâmetros básicos para nortear a decisão de utilizar/reestudar ou não tais tecnologias.

A Agência de Informação Embrapa é um portal web cujo objetivo é agrupar os dados sobre diversas Agências de Produto, que por sua vez são organizadas sob a ótica da cadeia produtiva de cada qual. A concepção deste portal deve criar condições para transformar a informação gerada pela pesquisa em capital intelectual, oferecer mecanismos que aumentam a qualidade na recuperação de informação e prover ferramentas que facilitam o trabalho colaborativo entre os especialistas na construção dos conteúdos do portal, dado que a empresa possui centros de pesquisa em todo o território nacional. Participam deste projeto três centros de pesquisa da empresa: Embrapa Informação Tecnológica, Embrapa Gado de Corte e Embrapa Informática Agropecuária. O Gerenciador de Conteúdos da Agência de Informação Embrapa é o sistema responsável pela manutenção e atualização dos dados desse portal. Atualmente, o gerenciador se encontra em uso por onze Agências de Produto. Desde seu lançamento, uma lista de novos requisitos, sejam funcionais ou de melhorias de performance, tem sido identificada e analisada. Para satisfazer a estes novos requisitos, um conjunto de tecnologias e ferramentas foram analisadas, visando a sua incorporação ao sistema e uma nova arquitetura foi proposta.

Este trabalho apresenta a nova arquitetura do Gerenciador de Conteúdos, as tecnologias e ferramentas analisadas e sua utilização na arquitetura proposta.

José Gilberto Jardine
Chefe-Geral

Sumário

Estudos para Melhoria da Arquitetura do Gerenciador de Conteúdos da Agência de Informação Embrapa

Maria Fernanda Moura
Sérgio Aparecido Braga da Cruz

Problema Abordado

A Agência de Informação Embrapa 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 do agronegócio e disponibilizada via 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 é organizado e armazenado em meio eletrônico. Para isso, foi criado um repositório de conteúdos de informação, categorizado segundo os diferentes ambientes de consumo da informação, e um *site,* onde são disponibilizadas as informações das diversas agências de produtos. Para manter os dados desse *site* íntegros e constantemente atualizados, fez-se necessário construir um sistema automatizado, baseado na arquitetura cliente/servidor e acessado através da Intranet da empresa, para inserção, alteração, exclusão e publicação de dados das diversas agências de produtos, permitindo a troca e o reuso desses dados. A integração das ferramentas de software utilizadas na Agência de Informação Embrapa compõe seu gerenciador de conteúdos (Moura et al., 2003).

A Fig. 1 apresenta o esquema de arquitetura de software da versão atual do Gerenciador de Conteúdos. O modelo adotado segue a orientação de divisão em camadas mais as idéias do padrão de desenvolvimento Model-View-Controller - MVC (Subrahmanyam et al., 2001; Bushmann et al., 1996), no qual separaram-se os dados (modelo, *model*) da forma de apresentação dos mesmos (interface com o usuário, *view*), colocando-se o controle entre eles (*controller*). Esse modelo facilita a estruturação de aplicações que podem ser decompostas em grupos de subtarefas; cada grupo está em um nível de abstração particular (Buschmann et al., 1996), de modo que a comunicação entre os objetos seja sempre de uma camada para outra. Desta forma, como delimitadas na Fig.1 tem-se as três principais camadas:

- Dados (*model*): composta pelo sistema gerenciador de bancos de dados (SGBD), o driver JDBC (*JAVA Database Connectivity*) e uma API (*Application Program Interface*) para encapsular as operações do banco de dados específico versus o padrão de dados do sistema de nome API-BD;

- Controle (*controller*): formado pelas classes das camadas denominadas Aplicação e Básicas. Na camada Aplicação tem-se todas as operações que o sistema precisa realizar com os dados; e, na camada Básicas tem-se as classes de manipulação de cada elemento do domínio do sistema;

- Apresentação (*view*): corresponde a parte do sistema executada (ou interpretada) no cliente (navegador, formulários HTML com ou sem *Javascripts* e *applets* JAVA) e a parte do sistema composta pelos programas JSPs. *JAVA Server Pages* - JSP, é uma extensão da tecnologia de *servlets*, no entanto, enquanto os *servlets* são programas JAVA™ os JSPs são textos, divididos em duas partes: conteúdo estático (HTML - *HyperText Markup Language*) e conteúdo dinâmico (*tags* JSP e/ou *scripts* JAVA™). *Applets* são programas escritos em JAVA™ que podem[1] ser incluídos em uma página HTML, de forma similar à inclusão de uma imagem; o *applet* é executado na máquina cliente através de uma máquina virtual JAVA associada ao navegador (Sun Microsystems, 2002a).

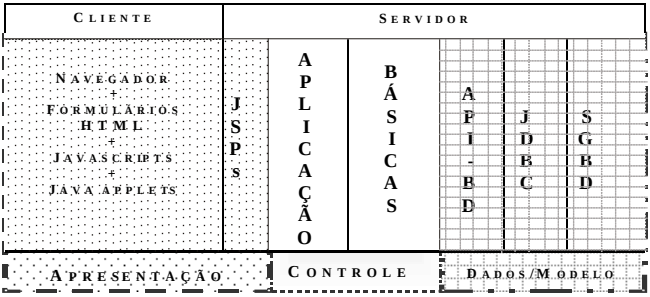


Fig. 1. Esquema da arquitetura de software da primeira versão.
Fonte: Moura et al. (2003).

Esse modelo facilita o reuso dos componentes criados, especialmente aqueles da camada do domínio e permite trocar camadas inteiras sem efeitos colaterais nas demais partes do sistema, o que facilita a adoção de padrões (camadas inteiras podem ser trocadas para aderir a novos padrões) e mantém as dependências entre camadas, ou dependências locais, sob controle (Buschmann et al., 1996). As tecnologias e padrões adotadas no modelo seguiram as recomendações de desenvolvimento da plataforma JAVA 2 Enterprise Edition - J2EE (Subrahmanyam et al., 2001).

[1] Veja item JSP Beans deste documento.

Para a nova versão do Gerenciador de Conteúdos foi necessário incorporar algumas evoluções, tanto para melhorar a implementação de algumas camadas quanto para satisfazer a novos requisitos de software. Assim, os estudos apresentados neste documento visam viabilizar a satisfação desses requisitos através da utilização de técnicas e ferramentas que melhorem as implementações das camadas do software. A nova arquitetura é apresentada com as soluções técnicas a serem implementadas na nova versão. Finalmente, este trabalho pretende, também, ser uma contribuição a outros projetos de software que necessitem do estudo de técnicas semelhantes e uma referência a um caso de mudança de arquitetura de software do modelo JSP cêntrico (Modelo 1) para arquitetura com *servlet ou* JSP controlador (Modelo 2) - (Sun Microsystems, 2003a).

Novos Requisitos Incorporados

A maioria dos novos requisitos, a serem incorporados na nova versão do Gerenciador de Conteúdos, são discutidos em Moura et al. (2002). A principal mudança na arquitetura refere-se à troca de dados utilizando-se *EXtensible Markup Language* - XML (World Wide Web Consortium, 2003). Na versão anterior, o uso de XML foi descartado para que se pudesse utilizar *JSP Beans* (ver item *JSP Beans*); porém, para acompanhar a tendência crescente por este tipo de solução e para ter maior independência em relação às soluções de implementação de tratamento dos dados, optou-se por reprojetar as camadas de interface utilizando-se esta técnica.

Outra mudança refere-se ao modelo de dados, pois este deve suportar os novos requisitos de controle de fluxo de trabalho por perfil de usuário e um modelo de unificação dos dados que se referem a recursos de informação de qualquer natureza. Os recursos de informação na Agência são formados pelos metadados de todo tipo de documento disponível no sistema, desde publicações quaisquer (em formato doc, PDF, rtf, etc.), imagens, sons, referências a outras instituições (externas à Embrapa), livros, fitas, anotações, etc., e, também, os conteúdos da árvore do conhecimento de cada Agência de Produto, que são considerados publicações eletrônicas internas à Embrapa. Na versão atual do software, respeitando a arquitetura da Fig. 1, na camada Básicas há classes diferentes para cada tipo de recurso de informação e conseqüentemente também diferentes tabelas de dados - dado que nessa versão cada tabela de dados originou uma classe (Moura et al., 2002). Dessa forma, com a unificação proposta, toda essa camada precisou ser modificada.

Além disso, a camada de interface com o banco de dados (API-BD, na Fig. 1) também foi reprojetada e reimplementada, de modo a refletir o modelo de classes

do sistema e facilitar a troca do sistema gerenciador de banco de dados, quando necessário. Deve-se esclarecer aqui que a troca do sistema gerenciador do banco de dados, quando necessário. Deve-se esclarecer aqui que a troca do sistema gerenciador do banco de dados é mais ou menos complicada na camada API-BD (Fig. 1), devido à forma como foram implementadas as montagens das expressões de consulta - que dependem do formato aceito pelo banco de dados. Essas expressões de consulta foram montadas dentro dos métodos de interface com o banco, assim, para trocar o banco, todos esses métodos precisam ser reimplementados.

Outra mudança, que naturalmente ocorreria de uma versão para outra, principalmente após os beta-testes implantados, seria relativa à interface gráfica disponível para os usuários. As mudanças da interface gráfica refletem-se na troca das camadas dos processos do Navegador, na Fig. 1; isto é, nos HTMLs, *JavaScripts e Applets* utilizados e, obviamente, nas trocas de dados com os JSPs, especialmente porque na nova versão isto ocorrerá via XML.

Assim, de forma estilizada, pode-se observar as principais mudanças da arquitetura na Fig. 2, onde os aplicativos da nova versão dos aplicativos de manutenção de dados do Agência são ativados de acordo com o perfil do usuário; assim, cada usuário pode executar aplicativos específicos, dependendo do conjunto de ações a ele atribuídas. As seguintes restrições devem ser observadas:

- no primeiro *login* de usuário válido, será criada uma sessão para o usuário e todo novo acesso será conferido com os dados dessa sessão - um objeto da classe Sessão (Sun Microsystems, 2003b);
- todos os formulários que enviarem dados ao sistema o farão através de um XML, cujas DTDs (Document Type Definition - DTD) serão especificadas para cada formulário (W3Schools, 2003);
- na camada de JSPs praticamente não ocorre processamento, apenas a ativação de algum método de classe da camada de aplicativos que tratará o XML recebido;
- os dados trocados entre processos cliente e servidor são duplamente validados: através de *javascripts* no cliente e, no servidor, após a análise sintática do XML. Com isto, espera-se garantir a confiabilidade dos dados e evitar que formulários que ficam na memória cachê de uma máquina cliente sejam reenviados ao servidor;
- no caso de controle de processamento em lote, o controle do lote será feito na sessão do usuário. Os processamentos em lote ocorrem quando há escolha de alterações de vários registros de dados, por exemplo, alterar metadados de vários recursos de informação;
- a camada de negócios encapsula a lógica de todos os componentes básicos do sistema e, funcionalmente, é muito semelhante à antiga camada Básicas;

- a camada de persistência é dividida entre classes que representam a de Negócios e classes que trabalham com um sistema gerenciador de banco de dados (SGBD) específico. Assim, tudo o que for específico de um único SGBD fica separado da correspondente classe e tabelas de dados (Cruz et al., 2003).

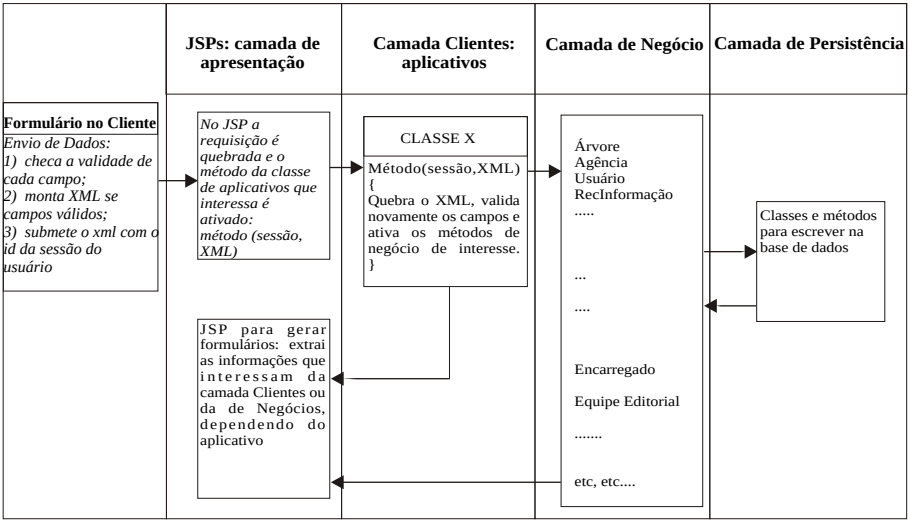


Fig. 2. Arquitetura da nova versão.

Ssas alterações visam garantir que os novos requisitos estejam incorporados à nova versão e que o padrão de desenvolvimento Model-View-Controller - MVC (Bushmann et al., 1996) continua sendo seguido. Além disso, para também garantir que as soluções de software que suportam esse fluxo de controle entre as camadas sejam de fácil evolução e eficientemente utilizadas, realizou-se um estudo sobre as soluções disponíveis, que serão apresentadas no próximo item.

Soluções Estudadas

Dado que os novos requisitos incorporados à atual versão do software somados a alguns problemas de implementação demandaram tamanha alteração na arquitetura da versão original, decidiu-se analisar outras soluções, ainda dentro da arquitetura J2EE, que permitissem melhorar especialmente a forma de implementação.

Assim, foram estudados alguns componentes, soluções e ferramentas que pudessem ser incorporadas ao desenvolvimento da nova versão, apresentados nos próximos subitens: *JSP beans*, *Tag extensions*, *Tag libraries*, *EJBs*, modelos de desenvolvimento para arquiteturas *multi-tier* e ferramentas de apoio à construção desse tipo de software.

JSP Beans

JAVA *beans* são componentes (classes) reusáveis de código JAVA, que respeitam a um determinado padrão de interface, de modo a poderem ser reconhecidos por métodos de introspecção e manipulados por outras aplicações, sem que seja necessário um pré-conhecimento dos mesmos; um modelo bastante utilizado para *beans* é o *JAVA Beans*™ (Sun Microsystems, 2002b). Os JSP *beans,* são JAVA *beans* utilizados com as *tags* JSP para importar e exportar dados entre formulários e *beans* (instâncias de classe implementadas no padrão de um *bean)*. Os *JSP beans* podem ser:

- page - *Beans* visíveis para uma única página JSP, durante a vida da requisição corrente (*current request*), em geral variáveis locais do método *service*;
- request - *Beans* visíveis em uma única página JSP, bem como para qualquer página ou *servlet* que inclua esta página ou faça requisições para esta página (geralmente: *request attributes*);
- session - *Beans* visíveis para todas as páginas JSP e *servlets* que participam de uma certa sessão de usuário, entre um ou vários *requests* (atributos de sessão);
- application - *Beans* visíveis para todos os JSPs e *servlets* que fazem parte de uma aplicação web (atributos de contexto do *servlet*).

As *JSP Tags*, ou diretivas JSP, embutem código, de modo que não seja necessário escrever extensos *scriplets* nos JSPs. *Scriptlets* são trechos de código JAVA que aparecem num código JSP. Por exemplo, a partir da classe Usuário, com atributos *código*, *senha*, *nome* e *papel* (seu perfil no sistema) implementada no padrão *JAVA beans*; supondo-se que todos esses atributos serão passados via um formulário de dados para o JSP utilizando-se o método *Post*, para serem atribuídos a uma instância dessa classe. Ao escrever o código para atribuir os dados ao *bean* (instância dessa classe) pode-se utilizar *scriplets*, como no exemplo 1 ou *JSP tags*, como no exemplo 2.

Exemplo 1 - Utilizando *scriplets:*

```
< %@page language = "java" import = "java.util.*,Usuario.*" % >
< %
// início do scriplet
        novo_usuário = new Usuário();
        novo_usuario.getCodigo(request.getParameterValues("Codigo"));
        novo_usuario.getSenha(request.getParameterValues("Senha"));
        novo_usuario.getNome(request.getParameterValues("Nome"));
        novo_usuario.getPapel(request.getParameterValues("Papel"));
% >
```

Exemplo 2 - Utilizando *JSP tags*:

```
<jsp: usebean id = "novo_usuário" scope = "page"
class = "Usuario.class" >
<jsp: setProperty name = "novo_usuário" property = "*" >
</jsp: usebean >
```

Comparando-se os dois exemplos é fácil observar como o uso das *tags* facilita a implementação dos *JSPs,* livrando-os da presença de código JAVA, o que, certamente, é mais apropriado para repassá-los a um *web designer* [2]. Porém, quando os parâmetros carregam listas de valores, não é tão simples evitar o uso de *scriptlets*. Suponha que um mesmo usuário possa ter vários papéis no sistema, e, conseqüentemente, que todos esses papéis lhe sejam atribuídos no formulário de entrada de dados; então o código no JSP, mesmo com o uso das *tags* precisaria ser semelhante a:

```
<jsp: usebean id = "novo_usuário" scope = "page"
class = "Usuario.class" >
<jsp: setProperty name = "novo_usuário" property = "*" >
</jsp: usebean >
< %
// mais scriplet para pegar os parâmetros da lista de papéis
if (request.getParameter("Papeis") != null) {
        String [] papeis = request.getParameterValues("Papeis");
        int tamPapeis = papeis.length;
        for (int i = 0; i < tamPapeis; i + +)
                Novo_usuário.atribuiPapel(papeis[i]);
}
% >
```

Isso pode ocorrer em vários outros casos em que as *JSP tags* não cubram todas as necessidades do código. Em alguns casos, definir extensões para as *tags* pode resolver esses problemas, como pode ser visto no próximo item. Apesar de ser sempre possível utilizar *scriplets*, o importante é utilizar-se deles o menos possível a fim de manter o JSP o mais próximo possível de HTML puro - para que ele possa ser repassado a um *web designer* sempre que alguma alteração de "interface gráfica" se faça necessária.

Tag Extensions e Tag Libraries

Tag Extensions é o mecanismo pelo qual pode-se definir novas *tags* a serem usadas nos JSPs. É aconselhável que ao se definir novas *tags* eles sejam de fato reusáveis

[2] Deve-se lembrar que os JSPs visam justamente constituir camadas de manipulação de interface para serem tratados por *web designers*.

entre vários JSPs, pois, caso contrário, o esforço de construção pode não valer a pena. Assim, o ideal é que se construam *tags* que possam ser usadas por vários sistemas da mesma empresa, ou então que se tente utilizar as *Tag Libraries*.

Tag Libraries são *tag extensions* já padronizadas, distribuídas em pacotes, havendo várias opções de domínio público ou comerciais. Por exemplo, o pacote JSPTL - JSP™ Standard Tag Library (Apache Software Foundation, 2003b) corresponde a uma *tag library* padrão de JSP, que contém as *tags* para controle de fluxo - iterações e laços. Outros pacotes básicos, de domínio público, trabalham com *sqls*, *xml*, *xsl*, etc., e são facilmente encontradas em vários *sites*.

Resumidamente, para definir uma *tag extension* (Subrahmanyam et al., 2001) precisa-se: definir sua interface para o JSP (< novatag: acaoDela... >); descrevê-la em XML, de acordo com um padrão preestabelecido, em um arquivo de extensão *tld* sob o diretório WEB-INF/tlds; definir e implementar os métodos de *handler* da *tag* (classe Java que contém os métodos que implementam cada *tag*); descrever sua localização no arquivo web.xml; e, ativá-la no JSP. Os *handlers* implementam interfaces predefinidas e novos métodos, dado que existe uma classe *handler* e dela deve-se derivar o *handler* específico.

Arquitetura Ideal - Multi-tier

Quando se desenvolve um sistema de software para Internet utilizando J2EE, espera-se ter um sistema todo dividido em multicamadas, cuja forma e funcionalidades ideais são apresentadas, de forma resumida, na Fig. 3, onde:

- enterprise resources: podem ser ou não código java, dados, lógica de aplicação específica ou mesmo sistemas legados;
- entity ejbs: objetos "leves" (geralmente, classes de interface) para encapsular os da classe abaixo. eles correspondem às visões específicas dos dados. na arquitetura ejb - *enterprise java beans*, são os *entity beans*. ejb nada mais é que uma arquitetura para encapsular a lógica de negócios, para componentes altamente especializados; que correspondem a uma coleção de classes java e uma especificação xml formando uma única unidade. as classes seguem regras predefinidas, provendo *callbacks* específicos; e, ainda, são executados em um *container*, encapsulando operações de mais baixo nível (transações, segurança, escalabilidade, concorrência, comunicação, gerenciamento de recursos, persistência, manipulação de erros, etc.);
- business logic layer: as aplicações estão aqui com todo o seu fluxo de controle, que independe completamente da apresentação. na arquitetura ejb corresponde aos *session ejb*;

- A camada Swing + non UI: não é obrigatória, existe se for comum a várias diferentes implementações da GUI (*Guided User Interface*) para as mesmas aplicações;
- Web specific library classes: para executar *parsers* mais complexos nos *requests*, manusear *cookies*, encriptar dados, etc.;
- JSP resource: contém os *JSP beans* e implementações das *tag extensions*. Esta camada é a ponte entre a interface e os objetos do domínio.

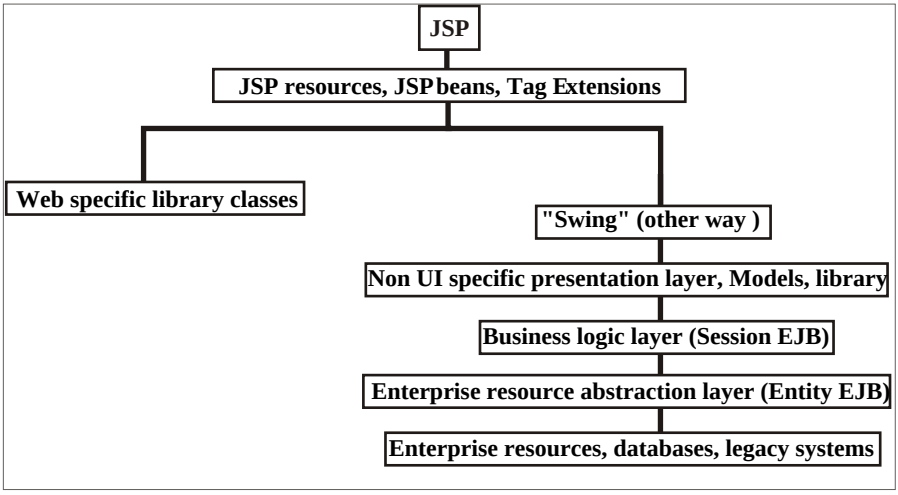


Fig. 3. Arquitetura multicamadas ideal.
Fonte: Allamaraju et al. (2001).

Front Controller Architecture

O modelo de desenvolvimento *Front Controller* (Subrahmanyam et al., 2001) prevê a distribuição do *workflow* e suas *views* a partir de uma *servlet* controladora (ou um JSP controlador, dependendo da opção de implementação utilizada), de modo que praticamente nada da funcionalidade da camada de negócios apareça nesses controladores. Esse tipo de modelo introduz um componente que intercepta a requisição do cliente e executa uma das seguintes ações:

- serviços como autenticação ou controle de acesso;
- escolha de uma *view* para manipular a requisição;[3]
- mudança de estado através de acesso a informações do processo do usuário.

[3] Uma formatação de dados para visualização dos mesmos, de acordo com o padrão de desenvolvimento MVC.

Na Fig. 4 pode-se observar como as requisições do cliente provocam as ações da *servlet* controladora, passando-as para um *handler* específico de seu tipo, que por sua vez escolhe o *bean* que deve tratá-la na camada de negócios e escolhe uma *view* para os resultados. Esse modelo com a mistura de uma *servlet* controladora e visões criadas com JSP é também conhecido como Modelo 2 (Seshadri, 2003), em oposição ao Modelo 1 - que era o modelo JSP cêntrico. A *servlet* utilizada nesse modelo controla as ações distribuindo o fluxo de controle da aplicação web e os JSPs, por sua vez, são focados exclusivamente nas tarefas de escrever código HTML.

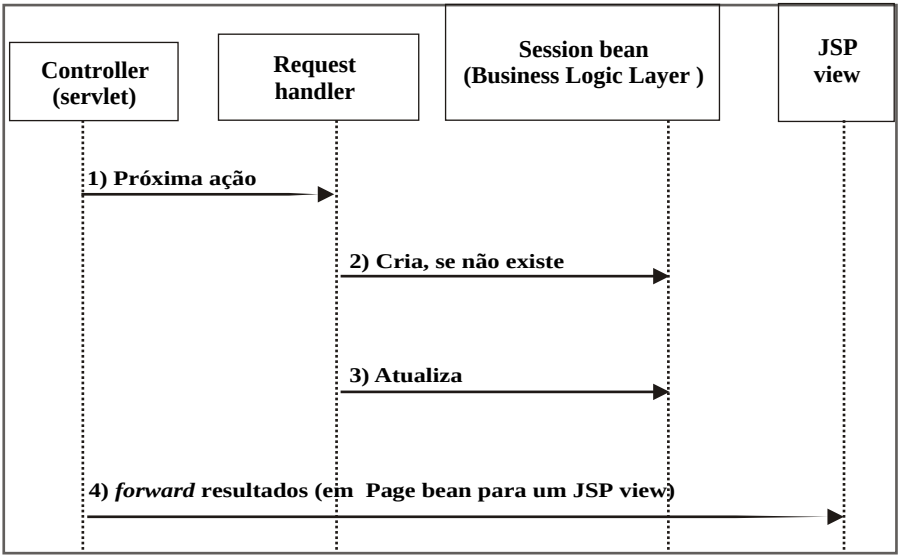


Fig. 4. *Front Controller Architecture.*
Fonte: Subrahmanyam et al. (2001), adaptada pelos autores.

Deve-se notar que na versão atual do Gerenciador de Conteúdos, ilustrada na Fig. 1, foi utilizado o Modelo 1; e, ainda, tanto o Modelo 1 como o Modelo 2 seguem o padrão de desenvolvimento MVC - Model-View-Controller (Bushmann et al., 1996). Essa mudança do Modelo 1 para o Modelo 2 na arquitetura do Gerenciador de Conteúdos ocorreu naturalmente devido à evolução dos requisitos, como apresentado nos itens anteriores. De acordo com as recomendações da JavaSun (Sun Microsystems, 2003a), "se uma aplicação web, originalmente desenvolvida sob o Modelo 1, usa muitos *scriplets*, *JSP tags* customizadas e muito código *JavaScript* para consistir e formatar as requisições ela deve ser reformatada para o Modelo 2"; e esse é justamente o caso do Gerenciador de Conteúdos, pois os JSPs originais já estavam muito sobrecarregados e com os novos requisitos os JSPs ficariam mais pesados.

Essa arquitetura adapta-se perfeitamente ao modelo de trocas de mensagens entre as camadas da Fig. 2, pois a camada cliente passa uma requisição de ação e um conjunto de dados no formato XML para o servidor[4]. Se a camada de apresentação, ilustrada na Fig. 2, for transformada em uma *servlet* controladora, basta enxergar os aplicativos como os tratadores (os métodos de *handler*) das requisições como ilustrado na Fig.5. De fato, esse tratamento resolve bem a questão de separação entre a camada de apresentação e a de negócios; de modo que, ao se utilizar XML nas trocas de dados e a informação sobre quais ações executar, tanto faz se o aplicativo é *web* ou *standalone*, bastando trocar as interfaces.

Ferramenta para Front Controller Architecture ou Modelo 2

A ferramenta Struts (Apache Software Foundation, 2003a), ou Struts Framework, segue o paradigma do modelo MVC, que ajusta-se perfeitamente ao Modelo 2 (ou *Front Controller*, como tratado no item anterior) e a sua arquitetura pode ser descrita a partir desse modelo de desenvolvimento. A Struts provê a camada de controle para uma aplicação web enquanto as demais camadas são desenvolvidas utilizando-se *JavaBeans* na camada do modelo e JSPs na camada de visão. Claramente essas camadas desenvolvidas podem incluir outros componentes, tais como EJBs junto ao modelo ou *templates* da Velocity (Cruz & Moura, 2002) na camada de visão.

Na Struts o principal componente da camada de controle é uma *servlet* da classe ActionServlet, configurada pela definição de um conjunto de ActionMappings. Um objeto da classe ActionMapping define um caminho mapeado contra uma URI (*Uniform Resource Identifier*) fornecida pelo *request* de entrada e normalmente especifica o nome completo da classe da ação (classe Action). Todas as ações são derivadas da biblioteca org.apache.struts.action.Action. As ações, assim como ilustrado na Fig. 5, podem encapsular a lógica de negócios, interpretar as saídas e despachar o controle para uma *view* apropriada (para criar a resposta apropriada). Pode-se também, configurar um conjunto de beans utilizados só para a entrada de dados a partir da classe ActionForm, a fim de manter-se estados e validações entre várias requisições (*requests*). Ainda, a Struts customiza algumas *tag libraries* fornecidas para suportar a camada View; algumas acessam a camada de controle, outras são genéricas para várias aplicações (pode-se utilizar outras *tag libraries*, como a JSPTL).

A *Struts* funciona basicamente da seguinte forma:

[4] Rever Fig. 2. Arquitetura da Nova Visão, no item Novos Requisitos Incorporados

- quando iniciada, o controlador analisa o arquivo struts-config.xml, utilizando-o para configurar e usar eficientemente outros objetos da camada de controle; juntos esses objetos formam a configuração da Struts, que, por sua vez, define os ActionMappings da aplicação;
- a *servlet* controladora consulta os ActionMappings para rotear os HTTP *requests*, que podem ser repassados para um JSP ou uma Action.

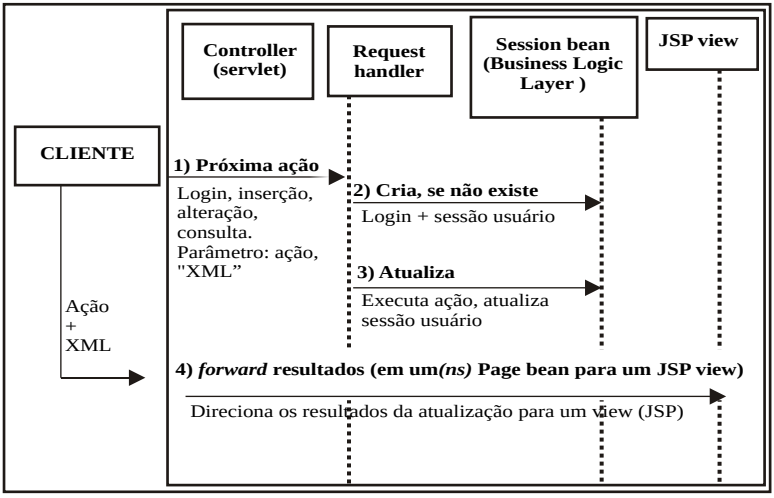


Fig. 5. Porque *Front Controller* se adequa a nossa arquitetura.
Fonte: Subrahmanyam et al. (2001), adaptada pelos autores.

Resultados Parciais

A partir da incorporação dos novos requisitos e da nova arquitetura, baseado no modelo 2, será implementada a nova versão do Gerenciador de Conteúdos da Agência de Informação Embrapa. Para isto, será utilizada como base a ferramenta Struts e as demais a ela agregadas, tais como os arquivos de configuração, *tag libraries*, JSP e *templates* da Velocity.

Para experimentar essas tecnologias, verificando sua adequação às necessidades do projeto, e também para obter um modelo de desenvolvimento adequado para o ambiente de implementação, foi desenvolvido um estudo de caso para o fluxo de controle das ações pertinentes ao cadastro de usuários de uma Agência de Produto. O estudo de caso segue o esquema de seqüência de páginas apresentado na Fig. 6, onde os retângulos representam páginas apresentadas ao usuário, as

setas indicam *hyperlinks* entre as páginas e setas rotuladas indicam processamento de requisição. A partir do formulário "página inicial" há um *hyperlink* para o formulário que checará o "*login*" do usuário e a partir desses dados, se o usuário do sistema for válido, apresentará uma lista de opções de ações para esse usuário. Na nova versão do gerenciador de conteúdos, as ações que a *servlet* controladora (neste caso implementada pela ferramenta *Struts*) apresentará ao usuário depende do perfil que foi cadastrado para o mesmo; por exemplo, se o usuário é o "administrador" da Agência de Informação Embrapa ou o "editor geral" de uma Agência de Produto ele terá acesso ao "cadastro de usuários".

Os dados que o usuário fornece ao sistema a partir dos vários formulários apresentados são repassados ao servidor em formato XML *(eXtensible Markup Language*). Como o objetivo do estudo de caso era apenas experimentar as tecnologias, os dados são preenchidos diretamente em XML. Após a submissão dos dados, o tratamento da requisição analisa o XML, através de um *parser* específico, construindo um novo objeto em memória. No caso de inserção ou alteração de usuário, esse objeto em memória é, então, persistido em banco de dados, utilizando-se as classes de persistência - esse procedimento exercita parcialmente todas as camadas da arquitetura ilustradas na Fig. 2.

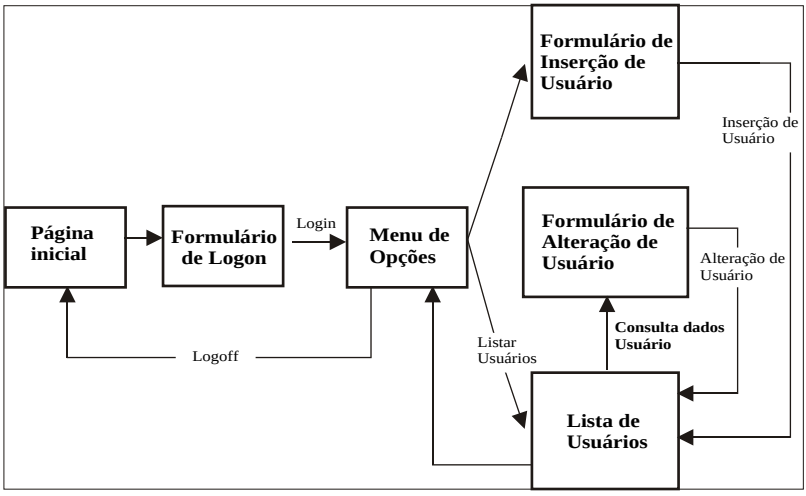


Fig. 6. Exemplo implementado.
Fonte: Cruz et al. (2003).

Se a opção do usuário for obter uma lista de usuários, a requisição tratada provocará uma consulta à base de dados e o envio dos resultados da consulta a uma página JSP. A opção pela implementação de *views* com JSP foi apenas para construir rapidamente esse exemplo, acredita-se também que *templates Velocity* devam ser avaliados aqui, devido à facilidade de alterá-los, como mencionado por Cruz & Moura (2002).

Desta forma, os resultados parciais, após o desenho da arquitetura, foram: a implementação do estudo de caso, com todas as camadas, porém apenas com os métodos necessários ao cadastro de usuários completamente implementados; e, a instalação e configuração da ferramenta Struts.

Com o desenvolvimento desse estudo de caso, foi possível verificar que o uso do Modelo 2 - na arquitetura proposta, bem como o uso das tecnologias adotadas, facilita a incorporação de novas funcionalidades e evolução das já existentes. Isto com menor impacto no código já implementado, dado que novas definições podem ser feitas quase que exclusivamente através de arquivos de configuração, pouca manutenção em código JAVA e algum, ou alguns, novos *templates* Velocity. E, ainda, facilita a distribuição de trabalho dentro da equipe de desenvolvimento, dado o baixo acoplamento entre as classes.

Conclusões

Com base nas tecnologias estudadas e no estudo de caso realizado, pôde-se concluir que a arquitetura proposta neste trabalho e sua implementação utilizando as tecnologias aqui apresentadas satisfazem as necessidades impostas pelos novos requisitos do gerenciador de conteúdos da Agência de Informação Embrapa. Além disso, a arquitetura e tecnologias para implementação aqui apresentadas garantem um grau de flexibilidade bastante elevado ao sistema, isto é:

- suas classes podem ser integradas a outros sistemas de informação da empresa;
- sua apresentação pode ser facilmente expandida para outros formatos, tais como WAP (*Wireless Application Protocol*) e para interface Palm;
- a arquitetura atende à necessidade imediata da empresa em fornecer a Agência também em CD, o que implica em modificar apenas a camada de apresentação.

Assim, a nova versão do gerenciador de conteúdos da Agência de Informação Embrapa deverá seguir a arquitetura proposta neste trabalho, com base no Modelo 2 e uso das tecnologias aqui apresentadas.

Referências Bibliográficas

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

APACHE SOFTWARE FOUNDATION. The Jakarta Project Struts. Disponível em: < http://jakarta.apache.org/struts/ > . Acesso em: 17 jul. 2003a.

APACHE SOFTWARE FOUNDATION. The JSP Standard Tag Library (JSTL): release early access 2 (EA2). Disponível em: < http://jakarta.apache.org/taglibs/doc/standard-doc/standard-ea2/ > . Acesso em: 9 set. 2003b.

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

CRUZ, S. A. B. da; MOURA, M. F. Formatação de dados usando a ferramenta Velocity. Campinas: Embrapa Informática Agropecuária, 2002. (Embrapa Informática Agropecuária. Comunicado Técnico).

CRUZ, S. A. B. da; MOURA, M. F.; SANTOS, A. D. dos; LEITE, M. A. de. A. arquitetura do gerenciador do conteúdo da Agência de Informação Embrapa. In: CONGRESSO BRASILEIRO DE COMPUTAÇÃO-CBComp, 3.; WORKSHOP DE INFORMÁTICA NA SAÚDE, 3.; WORKSHOP DE INFORMÁTICA APLICADA AO MEIO AMBIENTE, 1.; FÓRUM DE INFORMÁTICA APLICADA À PESSOAS PORTADORAS DE NECESSIDADES ESPECIAIS, 2., 2003, Itajaí. Anais... Itajaí: UNIVALI, 2003. p. 1068-1081. ref. SIN255.

MOURA, M. F.; LEITE, M. A. A.; EVANGELISTA, S. R. M.; SOUZA, K. X. S.; SANTOS, A. D. dos. Arquitetura de software para a manutenção de dados da Agência de Informação Embrapa. Campinas: Embrapa Informática Agropecuária, 2002. (Embrapa Informática Agropecuária. Boletim de Pesquisa e Desenvolvimento).

MOURA, M. F.; SANTOS, A. D. dos; LEITE, M. A. A.; CRUZ, S. A. B. da; SOUZA, M. I. F.; SOUZA, K. X. S.; EVANGELISTA, S. R. M. Gerenciador de conteúdos da Agência Embrapa de Informação. In: INTERNATIONAL SYMPOSIUM ON KNOWLEGDE MANAGEMENT = SIMPÓSIO INTERNACIONAL DE GESTÃO DO CONHECIMENTO ISKM 2003, 2003, Curitiba. Anais ... [Curitiba: PUCPR: CITS, 2003].

SESHADRI, G. Understanding JavaServer pages model 2 architecture: exploring the MVC design pattern. Disponível em: <http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html> . Acesso em: 17 jul. 2003.

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.

SUN MICROSYSTEMS. Applets. Disponível em: <http://java.sun.com/applets/>. Acesso em: 01 nov. 2002a.

SUN MICROSYSTEMS. Designing enterprise applications with the J2EE Platform, second edition. Disponível em: <http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html>. Acesso em: 10 set. 2003a.

SUN MICROSYSTEMS. Java 2 platform SE v1.3.1: interface java.beans DesignMode. Disponível em: <http://java.sun.com/j2se/1.3/docs/api/java/beans/DesignMode.html>. Acesso em: 02 nov. 2002b.

SUN MICROSYSTEMS. The Java Web Services Tutorial. Maintaining client state. Disponível em: <http://java.sun.com/webservices/docs/1.0/tutorial/doc/Servlets11.html#64744>. Acesso em: 17 jul. 2003b.

WORLD WIDE WEB CONSORTIUM. eXtensible Markup Language (XML). Disponível em: <http://www.w3.org/XML/>. Acesso em: 28 out. 2002a.

W3SCHOOLS. DTD tutorial. Disponível em: <http://www.w3schools.com/dtd/>. Acesso em: 9 set. 2003.

Informática Agropecuária

Ministério da Agricultura, Pecuária e Abastecimento

Governo Federal