Aplicações Corporativas

Aplicações Web Corporativas em Java EE

O **jCompany Full-Stack Framework - Community & International Version**, chamado resumidamente de **jCompany Free** neste artigo, é uma versão gratuita e de código aberto (licenciada sob GPLv3) da parte framework do **jCompany Developer Suite 5.1**, suíte completa de desenvolvimento mantida pela Powerlogic SA. O jCompany Free se encontra atualmente em versão beta 0.9 e prestes a ter sua primeira versão estável liberada.

A licença do jCompany Developer Suite é comercial – mas preserva alto índice de liberdade no acesso e repasse aos códigos fontes para os clientes, provendo três das quatro liberdades previstas pela FSF. Apesar de adequada para grandes empresas que produzem portfólios de aplicações corporativas em larga escala, seu aspecto comercial a torna menos acessível a pequenas e médias empresas, estudantes ou autônomos.

Foi pensando em democratizar este acesso, facilitar a formação de mão-de-obra e também em contribuir com a exportação de serviços de software do país – já que o jCompany Free traz os fontes do framework traduzidos do português para o inglês que a Powerlogic lançou o seu framework no modelo tradicional de "dual licence" (ver **Figura 1**).

Resumo DevMan

De que se trata o artigo:

Apresentação do jCompany Full-Stack Framework. O jCompany Free, como é chamado, faz a integração em nível de "Caso de Uso", reutilizando integradores de mais baixo nível para prover e gerenciar um ambiente com alto índice de reuso, típico do mundo Java EE Open-Source.

Para que serve:

Desenvolver aplicações corporativas Java EE 5 para Web com apoio do jCompany Free GPLv3, framework integrador que especializa 20 frameworks líderes de mercado, definindo uma arquitetura de base reutilizável que provê alta qualidade e produtividade.

Em que situação o tema é útil:

É muito complicado para as empresas controlarem a arquitetura de seus vários sistemas sem possuírem uma definição arquitetural coorporativa. Muitas tentam desenvolver sua própria plataforma de desenvolvimento e não obtém o sucesso desejado por essa tarefa estar fora do seu core-business.

Aplicações Corporativas com jCompany Free:

Se você trabalha em desenvolvimento corporativo envolvendo aplicações em grande quantidade e com largo volume de documentos (agregações de classes a serem persistidas), montar uma arquitetura de base Open-Source do zero é um grande risco. Construir padrões de alto nível, que exijam pouca programação, e evoluir dezenas de produtos de base internamente são em grande parte atividades fora do core-business da maior parte das organizações, mesmo com integradores de mais baixo nível, tais como Spring e JBoss Seam. Este trabalho de base com desvio de foco costuma culminar em soluções de arquitetura "artesanais", com variabilidade indesejável entre diversas equipes, isso quando possuem uma gerência de configuração! Pergunte-se: qual a versão e release atual da arquitetura (Open-Source) do sistema A ou B? Se você não tiver um único número para oferecer, possivelmente enfrentará dificuldades no futuro.

Nota do Devman

Veja o que você vai aprender adicionalmente neste artigo:

- O conceito de Dual Licence;
- O que é Full-Stack;
- A diferença de DTOs para objetos que possuem comportamento;
- A definição de DDD;
- A conceituação de WYSIWYG.

Como ilustra a **Figura 1**, a suíte comercial ataca a questão da produtividade também por outras dimensões além do framework integrador disponibilizado no jCompany Free. Porém, o jCompany Full Stack Framework, área em destaque na **Figura 1**, agrega grande valor ao desenvolvimento Java EE, visto que engloba a generalização radical de código MVC2 monótono (*Boiler-Plate*). As demais dimensões são:

- **jCompany IDE**: plugins Eclipse geradores de artefatos "não-Java" como formulários Web, configurações XML, properties, anotações (metadados), etc. (Importante: o jCompany é eminentemente uma solução OO. Este módulo não traz geradores de códigos Java, mas somente de artefatos não generalizáveis, como JSPs e descritores);

com jCompany Free

com jCompany Free

Desenvolva uma aplicação corporativa Java EE 5 para Web definindo uma arquitetura de base reutilizável que provê alta qualidade e produtividade

Paulo Alvim e Rogério Baldini

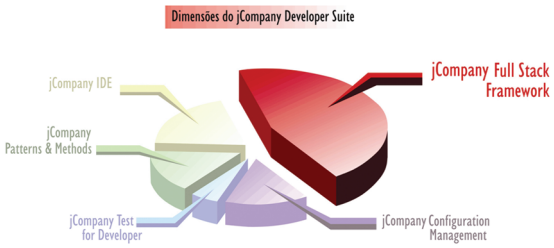

Figura 1. Dimensões do jCompany Developer Suite com segmento do jCompany Free em destaque.

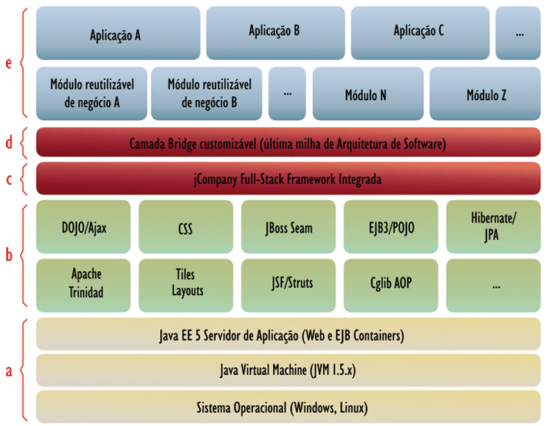

Figura 2. Arquitetura preconizada pelo jCompany Free.

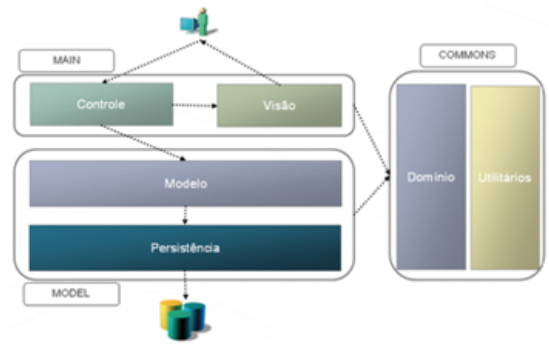

Figura 3. Arquitetura MVC2 do jCompany Free.

• **jCompany Test For Developer**: framework integrador de "última milha" para Testes de Unidade;

• **jCompany Patterns & Methods**: documentos de "padrões e métodos" para modelagem e especificação (Modelo de Classes, Casos de Uso, Estados e Interfaces com o Usuário);

• **jCompany Configuration Management**: gerência de configuração que embala, pré-configura e evolui mais de 40 produtos Open-Source distintos em uma única versão, sendo mais de 20 somente na parte framework.

☑ *O conhecimento do que está fora da versão GPLv3 é importante para quem inicia no jCompany Free, pois grande parte da documentação disponível considera as demais dimensões da versão comercial (especialmente do jCompany IDE). O livro "Tirando o Máximo do Java EE 5 Open-Source com jCompany Developer Suite", por exemplo, disponível em eBook no site da Powerlogic (www.powerlogic.com.br), pode ser útil, uma vez que o desenvolvedor reconheça esta distinção e busque formas alternativas de substituir as dimensões não incluídas.*

Arquitetura do jCompany Free

O jCompany Full-Stack Framework é considerado um **framework integrador de penúltima milha** devido ao seu posicionamento na pilha da arquitetura de software corporativa recomendada, ficando acima de um grande número de frameworks "de base" reutilizados (inclusive integradores de mais "baixo nível", como o JBoss Seam) e abaixo de uma camada chamada Bridge, para contextualizações corporativas, espaço destinado à empresa cliente. Veja o esboço mais macro da arquitetura na **Figura 2**.

Perceba que este esquema introduz dois aspectos sutis, mas bastante significativos, que maximizam o reuso e a padronização em uma arquitetura de software com base em produtos Open-Source, quando se fala em aplicações comerciais que manipulam formulários em larga escala:

• **Uma camada "c" reutilizada**, que permite o repasse da responsabilidade pela prospecção, integração, evolução, homologação e manutenção conjunta dos mais de 20 frameworks de base para a comunidade do jCompany Free. Esta é a "**penúltima milha da arquitetura**";

• **Uma camada "d" de responsabilidade da empresa**, que isola as implementações de negócio (representadas em "e") da arquitetura de base do jCompany Free e viabiliza que a empresa evolua ou altere qualquer padrão integrado de alto nível, generalizado

Nota do Devman

Dual Licence: O modelo Dual Licence representa a flexibilidade do licenciamento do software para se adaptar ao tipo de negócio do usuário. Se você está desenvolvendo e distribuindo aplicações opensource, existe a opção do framework GPLv3. Se você está desenvolvendo aplicações coorporativas com objetivos comerciais e não distribui o código fonte, deverá se encaixar na licença comercial.

Nota do Devman

Full-Stack: Em inglês, a expressão Full-Stack (framework de "pilha cheia") é utilizada alternativamente à classificação "integrador de penúltima milha", enfatizando o largo espectro de padronização e generalização do framework, por todas as camadas de uma arquitetura MVC2, incluindo camada ortogonal de Domínio e subdivisão da camada Modelo em Fachada, Serviço e Persistência.

Java64.indb 59 28/11/2008 10:34:11

na base. Esta é a "**última milha da arquitetura**", ou seja, depois dela começamos a pensar em casos de uso e regras de negócio.

Deste modo, viabilizamos o reuso com maior abstração: o arquiteto parte de um patamar bem mais alto do que partiria de um Spring ou JBoss Seam, por exemplo, reutilizando **uma arquitetura de base opinativa**, que define padrões em nível de caso de uso, incluindo formulários completos, seqüência de controle MVC generalizada e padrões de programação para extensão. Quando preciso, o arquiteto pode evoluir as definições de arquitetura para uma empresa ou para um portfólio específico de projetos por exceção, implementando variações a estes padrões na camada Bridge.

Por sua vez, os desenvolvedores trabalham de forma inteiramente desacoplada do jCompany Free (nenhum "*import*"!), trabalhando sobre uma **arquitetura contextualizada**, que reduz sensivelmente a quantidade de códigos Java e a variabilidade indesejável[1] na camada de negócio, como veremos. Quando preciso, ele também pode usar todos os recursos que os frameworks de base disponibilizam, interceptando as diversas partes do código através de "*Template Methods*" ou sobreescrevendo alguma funcionalidade do framework inteiramente.

Sob outro ângulo, a arquitetura MVC2[2] implementada pelo jCompany Free tem a organização ilustrada pela **Figura 3**.

Ela mostra as camadas típicas de um desenvolvimento MVC2, que o leitor já deverá conhecer. Para nosso perfil alvo de aplicações, a camada mais nobre costuma ser a de Domínio, que conterá classes em sua maioria persistentes e que representam entidades do negócio.

No jCompany Free a camada de Domínio é ortogonal, isto é, pode ser acessada por todas as outras camadas da arquitetura, e suas classes seguem o padrão "Entity Beans" da JPA, conforme definido no Java EE 5. POJOs são visíveis por todas as camadas do MVC2, evitando-se DTOs ou hierarquias paralelas e redundantes de dados. Por sua vez, estas entidades não dependem de outras camadas, sendo as mais estáveis da arquitetura. Em um próximo artigo desta série, veremos ainda que estas classes irão conter regras encapsuladas – e não somente dados, como os antigos Value Objects (VOs) dos Blue Prints J2EE, viabilizando uma programação Domain-Driven Design (DDD) conciliada com abordagens distribuídas (ex.: SOA).

Se nos aprofundarmos na arquitetura interna de cada camada, veremos que o jCompany Free ainda categoriza classes, em cada camada, definindo claramente seus papéis na arquitetura. Para tanto, utiliza sufixos padrões (customizáveis em nível de aplicação ou global) como **Action, Controller, Facade, Service, AS (Application Service), DAO, Entity e Helper**. Cada uma destas categorias já costuma conter código generalizado e evoluído por cinco anos para contemplar uma série de situações típicas (Manutenções de ciclo de vida de agregações de objetos e diversas variações relacionadas, em sua maior parte).

Cada classe genérica, em cada camada, permite programação via DP Template Method (com herança) ou DP Strategy (via Interface), ambas para codificação "por exceção". Não temos espaço no primeiro artigo para esta discussão, o que fica então para uma continuação.

Camada	Framework de Base
Visão	JSF 1.2 com Apache Trinidad, JSP 2.1, JSTL 1.2, Tiles, DOJO, Dynamic Drive HV Menu
Controle	Servlet 2.5, JBoss Seam, Eclipse BIRT, JBoss Client JMS, Yarfraw RSS
Modelo	EJB 3.0, CGLib
Persistência	JPA 1.0, Hibernate
Comuns	Apache Commons, Log4j, Hibernate Validator
Ferramentas	Maven, Ant (somente geração inicial de código)
Total	+ 20 (considerando-se dependências transitivas)

Tabela 1. Arquitetura MVC2 do jCompany Free.

Nota do Devman

DTO: DTO (do inglês Data Transfer Objects) é um Design Pattern usado para transferir dados entre subsistemas de uma aplicação. São comumente usados para conexão entre camadas da aplicação. Os DTOs não possuem qualquer comportamento. São construídos somente para transferência e recuperação de suas próprias informações.

Nota do Devman

DDD: DDD (do inglês Domain-Driven Design) é uma abordagem para o design de software em que designs de domínio complexo deveriam ser baseados em modelos e que, para a maioria dos projetos de software, o foco principal deveria ser o domínio e as lógicas de domínio em contrário a uma tecnologia particular usada para implementar o sistema.

Frameworks "de base" reutilizados

Os principais frameworks de base integrados, evoluídos e homologados pelo jCompany Free estão relacionados na **Tabela 1**, segmentados por camada MVC2.

A relação não inclui dependências transitivas. No site do produto em *http://jcompany.sourceforge.net* encontram-se explicações mais detalhadas sobre cada matéria-prima Open-Source reutilizada, com perguntas chave como "para que serve?", "por que foi selecionada?" e "como o jCompany Free agrega valor à sua função básica?".

Arquitetura de desenvolvimento

Do ponto de vista do desenvolvimento, o jCompany Free traz um "projeto-modelo" (*template*) para a criação de projetos Eclipse que agrupa as camadas MVC2 em três projetos distintos, já pré-configurados com dependências de *classpath* para Eclipse, representados no diagrama da **Figura 3** pelas seções "**MAIN", "MODEL" e "COMMONS"**. Pode-se utilizar o framework em qualquer outra IDE Java, alterando-se o "projeto-modelo" para outras IDEs como NetBeans, por exemplo, muito embora adotar a convenção do jCompany Free deva ser a primeira opção.

1 Para simplificação, no tutorial deste artigo não utilizaremos uma camada Bridge configurada, mas iremos discuti-la em artigo subsequente.

2 Muitas vezes chamada MVC2-P nas documentações do jCompany, para enfatizar a separação da camada de Persistência com DP DAO.

Java64.indb 60 28/11/2008 10:34:12

Instalando e configurando o ambiente de desenvolvimento (IDE)

Como o jCompany Free não traz a dimensão jCompany IDE do jCompany Developer Suite, precisaremos montar todo o ambiente de desenvolvimento. Uma boa dica é buscar uma configuração similar àquela integrada no jCompany IDE, por já estar bem homologada com o framework. Esta configuração utiliza o Eclipse 3.4 com WTP, plugins como o Apache Derby, o Hibernate Console, o JInto e o Red Hat Studio.

Neste artigo, iremos requerer apenas alguns dos citados. Procure, no site do produto, informações mais detalhadas sobre a instalação da JVM, do Eclipse e de cada plugin. Resumidamente, algumas dicas:

• **JVM**: Sugerimos a instalação do ambiente Java na versão 1.5.x da Sun ou superior. **IDE**: A versão mais indicada do Eclipse é o pacote "Eclipse IDE for Java EE Developers" atualmente com codinome Ganymede.

Para melhor administrar as variáveis de ambiente, sugerimos criar um arquivo "*C:\jcompany\eclipse\startjCompany.bat*" para iniciar o Eclipse. Adicione as variáveis da seguinte forma:

```
set CATALINA_HOME= C:\jcompany\apache-tomcat
set JAVA_HOME=C C:\jcompany\java
set PATH=%JAVA_HOME%/bin;%PATH%
start "Eclipse 3.4" /MIN "eclipse.exe" -clean
-data workspace -nl pt_BR -vmargs -Xmx512m
-Xms256m -XX:MaxPermSize=128M
```

O uso do Eclipse é recomendado porque o *template* para criação de projetos do jCompany Free traz projetos Java com algumas configurações em arquivos ".classpath"e ".project", específicos para este IDE. Com um pequeno esforço, no entanto, um desenvolvedor NetBeans experiente deverá ser capaz de alterar este *template* para o seu IDE;

• **App Server/Web Contêiner**: O Apache Tomcat 6.x pode ser utilizado na maior parte do tempo. O JBoss AS 4.2.x e o Glassfish também são homologados. Em tese, aplicações jCompany Free devem rodar em qualquer Application Server Java EE 5 Compliant. Não recomendamos o uso de EJB3 como *default*, já que esta tecnologia pode ser introduzida "a posteriori", se for necessário. O jCompany oferece IoC, DI e AOP (transações na fachada) para POJOs, funcionais em Tomcat e compatíveis com EJB 3.0, simplificando uma eventual portabilidade para um contêiner EJB3.

☑ ***Importante:*** *o arquivo web.xml do "projeto-modelo" vem pré-configurado com algumas roles padrões. Por este motivo, é recomendável que se defina no servidor de aplicação estas roles no Tomcat. Partindo-se da hipótese de que a instalação está toda sendo feita abaixo de "c:\jcompany" e que o Tomcat está em "apache-tomcat", as roles devem ser criadas no arquivo c:\jcompany\apache-tomcat\conf\tomcat-users.xml. Exemplo:*

```
<role rolename="Administrador"/>
<role rolename="Membros"/>
<role rolename="AreaTecnica"/>
<user username="admin" password="senha"
roles="tomcat,Membros,AreaTecnica,manager,
admin,Administrador "/>
```

• **SGBD**: O *template* de projeto traz um arquivo de configuração de contexto para Tomcat com a definição de um pool de conexões para Apache Derby, que é liberado de forma apropriada para o Tomcat. Pode-se definir um pool JDBC para outro SGBD ou utilizar este SGBD para desenvolvimento. Informe-se sobre o Derby em *http://db.apache.org/derby/integrate/derby_plugin.html* e em *http://db.apache.org/derby/integrate/plugin_howto.html*. Não se esqueça de incluir o JAR que contém o driver JDBC para a conexão com o Apache Derby (ou outro SGBD) no diretório *\lib* do Tomcat;

• **Maven**: É uma ferramenta obrigatória para construção e liberação de aplicações jCompany Free. Baixe o arquivo zip do Maven e descompacte-o no diretório desejado. Utilize a versão 2.0.9. Mais informações podem ser encontradas em *http://maven.apache.org/*, ou no artigo publicado na Edição 62.

Após a instalação, algumas configurações do Maven são necessárias:

• A primeira delas é informar o local do repositório de artefatos. Para isso, basta editar o arquivo *c:\jcompany\apache-maven\conf\settings.xml* e retirar comentário da tag **<localrepository>**, deixando-a como no exemplo abaixo:

```
<localRepository>C:/jcompany/repositorio</
localRepository>
```

• Novos repositórios precisam ser definidos na configuração do Maven, visto que o repositório central não possui todas as dependências utilizadas pelo jCompany Free. Esses repositórios são:

```
http://repo1.maven.org/maven2
http://repository.atlassian.com/maven2/
http://update2.powerlogic.com.br/jcompany/
maven
```

A inclusão destes novos repositórios também deve ser feita no arquivo *c:\jcompany\apache-maven\conf\settings.xml*. Um exemplo completo deste arquivo poderá ser encontrada em *jcompany_utils/maven/settingsexample.xml*, diretório que será instalado com o jCompany Free, no próximo tópico.

☑ ***Importante:*** *Estando em uma rede cuja navegação se faça através de um Proxy, configure as informações do proxy no mesmo arquivo de configuração acima, de modo que o Maven possa baixar "on-line" as dependências gerenciadas pelo jCompany Free.*

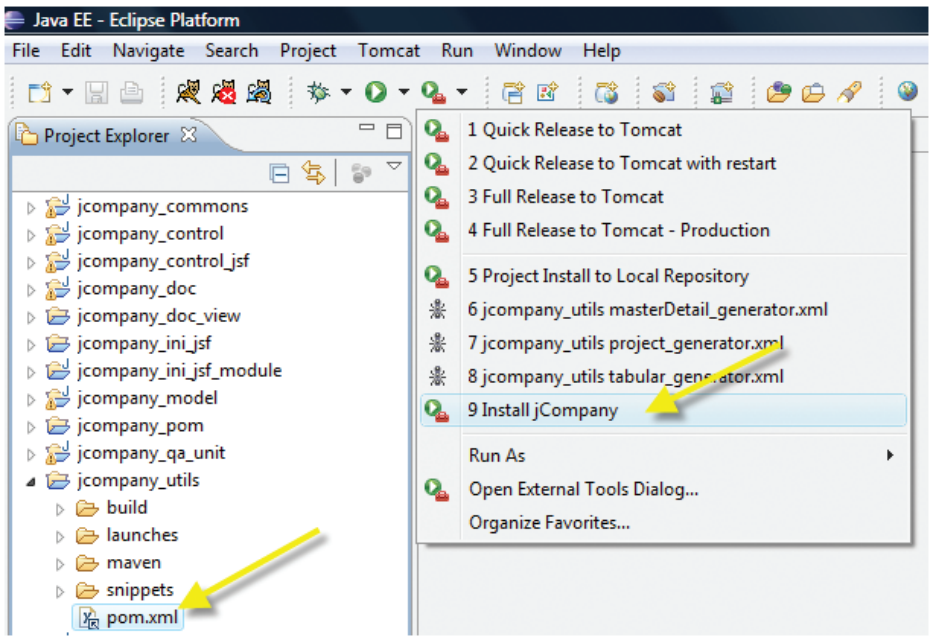

Figura 4. Execução de rotinas Maven para construção do jCompany Free.

Java64.indb 61 28/11/2008 10:34:13

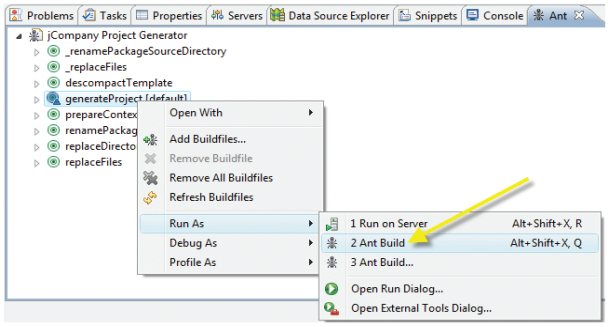

Figura 5. Criando um novo projeto a partir de arquivo Ant.

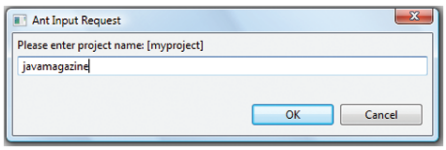

Figura 6. Última janela de diálogo Ant com nome da aplicação.

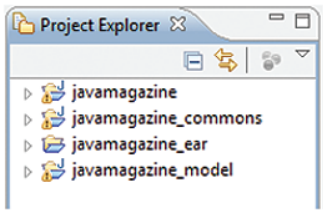

Figura 7. Projetos de desenvolvimento gerados.

• Para a integração das rotinas de construção e liberação do Maven com o Eclipse, é necessária a configuração de duas variáveis na IDE, que indicam onde estão instalados o Maven e o seu repositório, respectivamente. No Eclipse, abra o menu *Window>Preferences>Java>Build Path>Classpath Variable* e adicione a variável:

```
M2_REPO=C:\jcompany\repositorio
```

Em seguida, a partir do menu *Window>Preferences>Run>Debug>String Substitution*, adicione a variável MVN:

```
MVN=C:\jcompany\apache-maven\bin\mvn.bat
```

Com o ambiente de IDE configurado, podemos partir para a instalação do jCompany Free em si.

Instalando e configurando o jCompany Free

O jCompany Free é distribuído através de um único arquivo compactado chamado *jcompany-community.zip*, disponível para download a partir de *http://jcompany.sourceforge.net*. Este arquivo contém todo o seu código-fonte e documentação disponível, além de rotinas Maven preparadas para compilar e montar arquivos de distribuição JAR (Java) e WAR (recursos), e baixar todas as dependências de forma automatizada.

Após baixar o arquivo, descompacte-o no diretório *c:\jcompany\framework*. Diversos projetos configurados no padrão Eclipse serão expandidos, contendo os fontes do jCompany Free segmentados para a arquitetura MVC2. Importe todos estes projetos no Eclipse.

Em um primeiro momento, alguns dos projetos apresentarão problemas de compilação, o que será corrigido quando fizermos a primeira construção via Maven, o que provocará o *download* automático de todas as dependências que faltam no *classpath* pré-configurado no Eclipse.

Para facilitar o processo de construção do jCompany Free, configure outra variável no Eclipse. Abra o menu *Window>Preferences>General>Workspace>Linked Resources* e adicione a seguinte entrada:

```
JCOMPANY_POM=C:\jcompany\jcompany\pom.xml
```

Com isso, o ambiente está completamente configurado para gerarmos os binários. Clique sobre o arquivo *pom.xml* abaixo do projeto "jcompany_utils" e execute a tarefa "Install jCompany" a partir de *Run>External Tools>Open External Tool Dialog>Program*, conforme a **Figura 4.**

Se tudo foi configurado corretamente até aqui (não somente a compilação), o empacotamento e a distribuição do jCompany Free terá sido realizada, como também o **download de todas as suas dependências dos sites remotos, seguida da instalação das mesmas no repositório Maven local**. Esse processo pode demorar alguns minutos, visto que serão baixados aproximadamente **50 MB** em "matéria-prima Open-Source", os frameworks de base reutilizados no jCompany Free.

Após a finalização da tarefa de instalação, caso os projetos no *workspace* do Eclipse ainda estejam com problemas de compilação, selecione todos os projetos e execute a tarefa de limpeza em *Project>Clean*.

Para finalizarmos a configuração, podemos customizar a visualização e edição de JSPs através de *Snnipets* para facilitar o aprendizado das tags JSF especializadas do Apache Trinidad pelo jCompany Free. No Eclipse acesse *Window>Show View>Other...>General>Snippets* e, na visão de "Snippets", clique com o botão direito do mouse e depois em *Customize*. Faça então a importação do arquivo que se encontra em *c:\jcompany\framework\jcompany_utils\snippets/snippets.xml*.

Neste ponto devemos ter o ambiente de desenvolvimento instalado e configurado para o início do desenvolvimento.

Todas estas instalações e configurações do Eclipse, plugins diversos, App Servers, Maven e frameworks de base, são realizadas em um único passo de instalação pelo módulo jCompany IDE, disponível somente na versão comercial – mas para um desenvolvedor da comunidade de software livre, todos estes são produtos líderes em sua categoria e bem conhecidos.

Criando um primeiro projeto

A distribuição atual do jCompany Free utiliza scripts Ant para a geração de novos projetos de desenvolvimento e artefatos necessários para a realizações de vários Casos de Uso Padrões[3].

Estes casos de uso normalmente envolvem manutenções do ciclo de vida de agregações de objetos, mesmo complexas, tais como "Manter Classe" (para classes com poucos objetos), "Manter Agregação Simples" (CRUD Simples, envolvendo uma única classe e vários componentes), "Manter Agregação Mestre/Detalhe" (CRUD para composições envolvendo

3 A versão comercial possui plugins específicos do Eclipse que aprimoram estas atividades e roteiros de Cheat-Sheets que funcionam como orquestradores, substituídos no jCompany Free por uma versão básica em Ant.

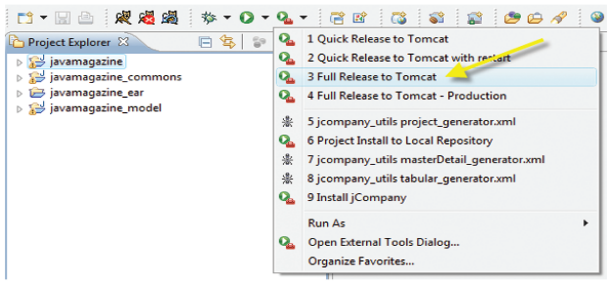

Figura 8. Diversas tarefas Maven customizadas do jCompany Free.

Java64.indb 62 28/11/2008 10:34:13

uma classe raiz, vários componentes e composições). Existem muitos outros na versão comercial, que com o tempo serão portados para o jCompany Free.

Para criar um novo projeto, execute o script Ant *jcompany_utils/build/project_generator.xml*. A **Figura 5** captura o momento do disparo para a criação do projeto.

O diálogo do "jCompany Project Generator" (Ant Input Request) irá solicitar quatro informações:

1. O endereço do arquivo de *template* para criação de projetos (basicamente, um ZIP que pode ser expandido e customizado). Informe o caminho deste arquivo, que deverá estar na pasta *.../repositorio/org/jcompany/ini/jcompany_ini_jsf/*, com seu devido versionamento.

Ex.: *c:/jcompany/repositorio/jcompany/ini/jcompany_ini_jsf/0.9/jcompany_ini_jsf-0-.9.zip*;

2. O diretório raiz onde a aplicação será gerada. Ex.: *c:/jcompany/eclipse/workspace*;
3. O nome da empresa. Ex.: "powerlogic";
4. O nome da aplicação. Ex.: "javamagazine". A **Figura 6** ilustra o diálogo da entrada de dados;

Após a geração, os quatro projetos criados devem ser importados para o Eclipse, usando *File>Import>General>Existing Projects into Workspace*. A **Figura 7** mostra o resultado da importação no Eclipse. Três deles seguem a organização MVC2 ilustrada na **Figura 3** e o quarto (com sufixo EAR), somente é utilizado como auxiliar na criação de arquivos EAR via Maven, quando utilizando JBoss AS ou Glassfish, por exemplo. Perceba que os projetos seguem uma estrutura de pacotes bem padronizada, trazendo ainda diversas classes vazias, que definem "espaços padrões da arquitetura" para conter implementações típicas.

Com o projeto criado, poderemos executar nossa primeira rotina Maven para compilação, construção e liberação de arquivo WAR para o Tomcat. Através dos diálogos de *External Tools* do Eclipse, verifique que várias rotinas do jCompany Free já estão disponíveis. Para a primeira liberação (*deploy*), selecione o projeto principal (clique sobre ele) e execute a tarefa "Full Release to Tomcat". A **Figura 8** representa a interface de chamada para a tarefa Maven.

Essa tarefa fará a liberação da aplicação no diretório *\webapps* do Apache Tomcat, configurado anteriormente. Quando a tarefa estiver completa, inicialize o servidor Tomcat e entre no Navegador com a url *http://localhost:8080/javamagazine*.

Listagem 1. Delegação necessária na entidade para montar link de edição.

```
public String getEditionLinkPlc() {
  return PlcEntityHelper.getInstance().getEditionLinkPlc(this);
}
```

Listagem 2. Relação de Entidades para o Hibernate.

```
<mapping class="com.javamagazine.app.entity.Funcionario"/>
<mapping class="com.javamagazine.app.entity.Dependente"/>
```

Listagem 3. Trecho de código com validações invariantes do funcionário como anotações

```
@Column(name="CPF", length=11, nullable=true)
@PlcValidateCpf
private String cpf;

@Past
@Column (name = "DATA_NASCIMENTO", nullable=false, length=11)
@Temporal(TemporalType.DATE)
private Date dataNascimento;
```

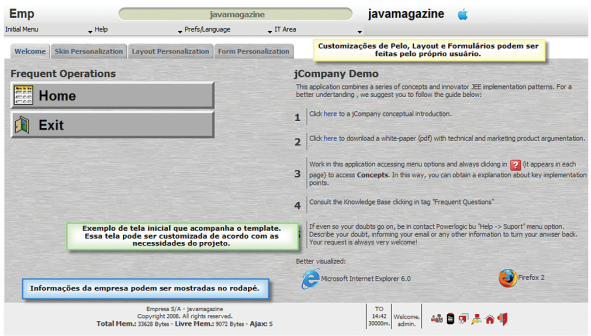

Figura 9. Página principal contendo web-design básico.

A primeira página exibida pela aplicação é a de autenticação (login). Informe um usuário que possua a *role* "Membros", conforme vimos anteriormente. Por exemplo, "admin" e "senha".

Ao entrar na aplicação (ver **Figura 9**) experimente os diversos recursos pré-construídos no jCompany Free: personalizações multi-pele, multi-leiaute e de formulário, rodapés com efeito *fisheye* (para ações como "desconectar", "quem está on-line", "release notes", etc.), botões tipo quiosque como atalho para funções mais usadas, *tab-folder* DHTML e menus *pull-down* – tudo isso internacionalizado, sendo o padrão do jCompany Free a geração no idioma inglês[4].

Construindo um primeiro "Caso de Uso Padrão"

Iremos agora implementar um caso de uso de manutenção "Mestre/Detalhe" (Funcionário/Dependentes, Nota Fiscal/Itens, etc.), consistindo de um formulário de manutenção com operações de CRUD e afins (impressão, ajuda, clonagem, assistente, etc.) e também de outro formulário associado, para "pesquisa e seleção" de objetos para edição[5].

Passo 1: Criar as classes que representam Entidades de Domínio (Classes Persistentes)

A criação das classes Java que representam as Entidades de Domínio pode ser feita a partir do Eclipse ou via geração a partir do modelo UML em alguma ferramenta CASE, como o Enterprise Architect (EA). Um vídeo básico de explicação sobre sincronismo de classes Java com o EA pode ser visto em *http://www.powerlogic.com.br/mda/mda1.html*.

Para entidades, alguns padrões devem ser seguidos (todos são customizáveis, mas, para o propósito deste artigo, recomenda-se utilizá-los):

- Devem ser criadas no projeto "javamagazine_commons", dentro do pacote "com.javamagazine.app.entity";

4 Como dissemos na introdução, o jCompany Free possui a mesma base de código do framework do jCompany Developer Suite, porém todos os seus métodos, pacotes, mensagens e diretórios foram traduzidos do português para o inglês, para iniciativas de exportação de serviços. Com o tempo, os javadocs também serão traduzidos para o inglês.

5 Esta pesquisa traz facilidades além do escopo deste artigo, como operações generalizadas para paginação, exportação dos resultados pesquisados em diversos formatos (xls, csv, xml), visão em RDF/RSS, dentre outros - tudo com programação declarativa (algumas anotações somente).

Java64.indb 63 28/11/2008 10:34:14

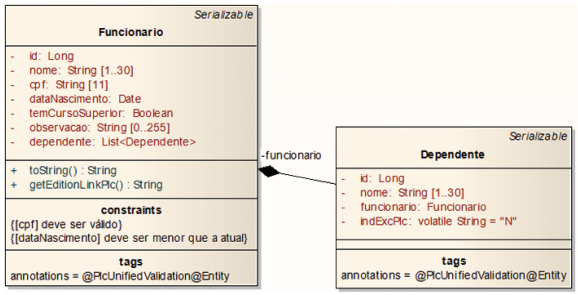

Figura 10. Modelo de Classes para o tutorial.

• Devem ser identificadas por estratégia Object-Id (propriedade com nome "id", tipo Long e algum esquema de geração automatizada);

• Devem implementar **java.io.Serizalizable**;

• Devem possuir a anotação **@PlcUnifiedValidation**, para que os componente JSF do jCompany herdem os tamanhos e nulidades das Entidades (evitando a necessidade de informá-los de forma redundante em páginas);

• Devem possuir um método **toString()** retornando um valor descritivo da entidade (Ex.: **return getNome()** para funcionário), pois o jCompany irá utilizá-lo como parte do título do formulário, para distinguir múltiplas instâncias no "favoritos" ou "histórico" do navegador;

• Entidades principais (raízes da agregação) devem implementar o método com a delegação da **Listagem 1**;

• Entidades de detalhe ("composições" no linguajar da UML) devem possuir a propriedade transiente **String indExcPlc**, para auxiliar na exclusão de objetos otimizada e em lote (recebe o termo "volatile" no modelo UML).

Se, por um lado, mesmo os padrões acima podem ser modificados, por outro existem diversas outras boas práticas padronizadas para representação de entidades no jCompany Free que não serão discutidas neste artigo por questão de espaço.

Em nosso tutorial, vamos usar o exemplo "Funcionario -> Dependente", cujo modelo de classes está representado na **Figura 10**. Todas as classes Java e artefatos utilizados a partir daqui estão disponíveis para *download* a partir do site da Java Magazine, assim como no site do jCompany Free.

Passo 2: Declarar o mapeamento Objeto-Relacional para as Entidades

Com as classes criadas, faça o mapeamento Objeto-Relacional segundo o padrão JPA, utilizando anotações (Java Annotations). Para tanto, você pode utilizar o plugin "Dali" ou qualquer outra ferramenta de sua preferência. O código das classes, já mapeado, pode ser baixado do site da Java Magazine e do site do jCompany Free.

Para finalizar os mapeamentos, como usaremos o Hibernate sobre o Tomcat, não se esqueça de relacionar as entidades no arquivo *hibernate.cfg.xml*, que se encontra disponível no projeto "javamagazine_model". A **Listagem 2** mostra o trecho de código que deve ser editado no Hibernate.

Passo 3: Declarar e/ou codificar as validações invariantes

Após o mapeamento, implemente as validações de entrada de dados de forma declarativa e encapsulada nas próprias entidades, através de anotações simples (somente as "invariantes", que não variam conforme nenhum caso de uso da aplicação). A **Listagem 3** mostra algumas das validações disponíveis.

Passo 4: Gerar os artefatos Java EE

Utilizaremos um novo script Ant para gerar uma primeira versão de artefatos Java EE necessários para uma aplicação Web, tais como JSP, chamada de Menu Tiles (XML) e metadados (anotações). Execute o arquivo *jcompany_util/build/masterDetail_generator.xml*, utilizando a visualização Ant do Eclipse, clicando com o botão direito do mouse *Run As>Ant Build*.

O Script "jCompany Master/Detail Generator" irá solicitar algumas informações:

1. O diretório raiz onde a aplicação foi gerada. Ex.: *c:/jcompany/workspace*;
2. O nome da empresa. Ex.: "powerlogic";
3. O nome da aplicação (o mesmo informado no script de geração de projetos). Ex.: "javamagazine";
4. URL base para o caso de uso. Trata-se de um texto identificador, todo em letras minúsculas, para ser utilizado como padrão nos nomes de diversos artefatos relacionados, bem como nas URLs das páginas de manutenção e seleção. Informe "funcionario" para o nosso caso;
5. Nome da classe raiz da agregação (mestre), que será utilizada no caso de uso. Ex: **Funcionario**. Importante: O nome não deverá conter o pacote;
6. Nome da classe de composição (detalhe) que será utilizada no caso de uso. Ex: **Dependente**. Importante: O nome não deverá conter o pacote.

A partir destas informações, a rotina Ant criará alguns artefatos e alterar outros, mas note que **nenhum código Java será gerado!** Gerar código Java repetitivo é uma simplificação que cobra seu preço. Em uma abordagem OO, a primeira consideração deveria ser a fatoração deste código repetitivo – e isso é feito à exaustão no jCompany Free.

Passo 5: Editar os metadados

Um arquivo de configuração (metadados), mostrado na **Listagem 4**, é definido com nome *package-info.java* no projeto "commons" e abaixo do pacote **org.jcompany.config.domain.app.funcionario**, que declara a Entidade Raiz do Caso de Uso Padrão. Esta declaração é essencial, pois a partir dela as rotinas de introspecção do jCompany Free conseguem obter inteligência para generalizar a manutenção.

Existem centenas de configurações de metadados que ativam comportamentos genéricos do framework, declaráveis como anotações em arquivos *package-info.java*, tanto na camada de controle quanto na de domínio (veremos algumas em outros artigos). Por exemplo, o jCompany Free produz aplicações internacionalizadas como padrão, e o inglês é o idioma padrão. Mas basta alterar o locale "en_US" nos metadados em escopo de aplicação, no arquivo *org.jcompany.config.app.package-info.java* da camada controle (projeto principal), para "pt_BR", para obter uma aplicação com idioma padrão em português.

Passo 6: Editar as JSPs

Além dos arquivos de metadados, algumas JSPs também são geradas na pasta *src/main/webapp/WEB-INF/jsps/funcionario*, contendo segmentos (*markups*) montados pelo Tiles[6] para os formulários de manutenção e seleção, além de indicações de leiaute.

Estas JSPs deverão ser editadas para conter campos correspondentes às entidades, o que pode ser feito com editores do WTP

6 Um suporte alternativo ao Apache Trinidad + Tiles está a caminho no ramo comercial, com as tecnologias RichFaces + Facelets.

Java64.indb 64 28/11/2008 10:34:14

Nota do Devman

WYSIWYG: WYSIWYG (do inglês What You See Is What You Get) é a capacidade de um programa de computador de permitir que um documento, enquanto manipulado na tela, tenha a mesma aparência de sua utilização. Esse termo é muito utilizado por editores visuais de HTML e JSPs, em que você pode editar o conteúdo dos arquivos e ao mesmo tempo visualizar seu formato final.

e a ajuda dos Snippets que importamos – ou com o editor do Red Hat Studio, que pode ser configurado para visualizar as tags do jCompany Free e apresentar um WYSIWYG básico (instruções mais específicas para este segundo caso podem ser encontradas no site do produto).

Veja na **Listagem 5** o exemplo final da JSP para a parte "mestre" do formulário de manutenção, contendo já os acréscimos de campos com formatos apropriados.

As JSPs do jCompany Free são ultra-simplificadas, com apenas uma dependência (inicialmente), das tags do jCompany Free, todas especializações do Apache Trinidad. As tags assumem *defaults* para inúmeros itens, tais como estilos CSS, tamanho e nulidade de campos, validação invariante, etc. Além disso, as JSPs já vêm recortadas apropriadamente (uma para cada seção distinta dos formulários) e com uma convenção de nomenclatura que dispensa a declaração de leiautes Tiles! Nem mesmo a declaração de formulário (**<f:form>**) é redundada, pois está fatorada para os leiautes ancestrais, juntamente com topo, rodapé, menu, declarações HTML de body, header, área de botões e mensagens, etc..

☑ *Lembre-se de adicionar valores para os novos rótulos criados no arquivo de mensagens "ApplicationResources_en_US.properties" (se desejar manter o I18n e usar as tags "plcf:title" para os rótulos).*

Passo 7: Editar a chamada de menu

A hierarquia do menu principal da aplicação é definida no padrão Tiles, no arquivo *app-tiles-menu1.xml*. O menu será então renderizado em diversos leiautes diferentes, de forma inclusive dinâmica, conforme a preferência de usuários. Não é necessário modificar a hierarquia para nosso exemplo.

Passo 8: Configurar o SGBD Apache Derby

Antes de prosseguir, precisamos disponibilizar um banco de dados. Supondo que o plugin do Derby esteja configurado, vamos adicionar a Nature do Derby para o projeto. Para tanto, clique com o botão direito do mouse no projeto, no item *Apache Derby>Add Apache Derby Nature* e, em seguida, na opção *Apache Derby>Start Derby Network Server*, para iniciar a instância do banco de dados.

Passo 9: Liberar as modificações para o Tomcat e atualizar do Esquema DDL

Neste ponto, podemos efetuar mais uma liberação para o Tomcat, mas desta vez ela não precisa ser completa (full). Vamos somente enviar os arquivos modificados e/ou criados para o Tomcat, ao utilizarmos agora uma liberação rápida. Selecione o projeto principal e execute a tarefa "Quick Release to Tomcat with restart". Terminada a tarefa, inicie o Tomcat ou, caso já esteja iniciado, acesse e entre na aplicação.

Antes de testar o caso de uso, precisaremos atualizar o esquema do banco para criar as tabelas relacionais mapeadas. Para ajudar neste processo, o jCompany Free traz um menu especial, que dispõe de utilitários disponíveis apenas para usuários com role "AreaTecnica" (conforme no *web.xml*). Um destes utilitários, especializado do Hibernate, nos permitirá obter a DDL a partir do mapeamento que fizemos. Acesse o menu *IT Area>DDL Scheme Generation* e clique no botão *Generate Scheme*. Depois de conferir a DDL gerada, clique no botão *Execute Scheme*. As mensagens "Success in generating DDL" e "Success in updating scheme" são esperadas, respectivamente.

Listagem 4. Definição da entidade "raiz" da agregação a ser mantida, bem como padrão de caso de uso adotado.

```
@PlcConfigGroupAggregation(
  entity = com.javamagazine.app.entity.Funcionario.class,
  configDefault = @PlcConfigDefault(logic = PlcConfigDefault.Logic.
  MASTER_DETAIL)
```

Listagem 5. /WEB-INF/jsps/funcionario/funcionarioMestre.jsp.

```
<plcf:table titleKey="masterman.title" onExit="comutaAba(1,'corpo:detail:0:
  name')">
   <plcf:line>
    <plcf:cell>
      <plcf:title titleKey="label.id"/>
      <plcf:oid id="master_id" helpKey="ajuda.id" autoRetrieve="S"/>
    </plcf:cell>
    <plcf:cell>
      <plcf:title titleKey="label.nome"/>
      <plcf:text id="nome" value="#{plcEntity.nome}" helpKey="ajuda.nome" />
    </plcf:cell>
   </plcf:line>
   <plcf:line>
    <plcf:cell>
      <plcf:title titleKey="label.dataNascimento"/>
      <plcf:date id="dataNascimento" value="#{plcEntity.dataNascimento}"
        helpKey="ajuda.dataNascimento" />
    </plcf:cell>
    <plcf:cell>
      <plcf:title titleKey="label.cpf"/>
      <plcf:text id="cpf" value="#{plcEntity.cpf}"  size="14"  helpKey="ajuda.
        cpf" onkeydown="mascararCnpjCpf(this, event, 0);"/>
    </plcf:cell>
   </plcf:line>
   <plcf:line>
    <plcf:cell>
      <plcf:title titleKey="label.temCursoSuperior"/>
      <plcf:checkBox readOnly="#{requestScope.visualizaCampoPlc=='t'}"
        id="temCursoSuperior" value="#{plcEntity.temCursoSuperior}"
        helpKey="ajuda.temCursoSuperior"/>
    </plcf:cell>
    <plcf:cell>
      <plcf:title titleKey="label.observacao"/>
      <plcf:area id="observacao" value="#{plcEntity.observacao}"
        helpKey="ajuda.observacao" columns="80" numberLines="3" />
    </plcf:cell>
   </plcf:line>
</plcf:table>
```


Java64.indb 65 28/11/2008 10:34:14

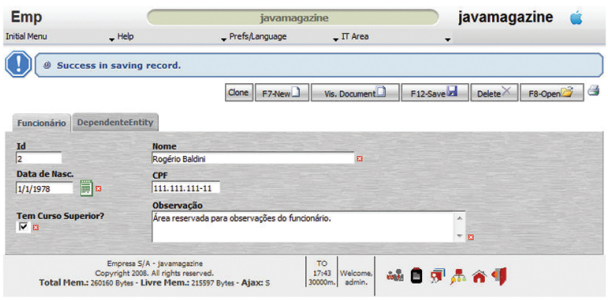

Figura 11. Formulário final, com opções funcionais e acabamento avançado.

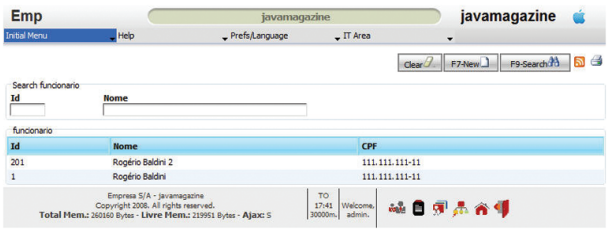

Figura 12. Tela de seleção, com opções de impressão, geração de RSS e link para edição dos registros.

Passo 10: Testar os dois formulários que compõem o novo caso de uso

Agora chame nosso formulário de manutenção a partir do item de menu "Initial Menu". Preencha-o com alguns dados (válidos e inválidos) e clique em "F12-Save". A **Figura 11** ilustra o formulário final de manutenção.

Perceba que os estilos CSS estão corretos e se comportam bem nas várias "peles"; marcas de obrigatoriedade aparecem somente nos campos "nullable=false"; os tamanhos dos campos e validações que declaramos (restrições invariantes) estão funcionais, etc. Grande parte da inteligência assumida é, na verdade, herdada do modelo de Domínio, sem que tenhamos que codificá-la de forma redundante nos formulários JSF (lembrando que a codificação por exceção é sempre possível).

Experimente clicar nos botões *Clone* para obter uma cópia nova de um registro (para criação rápida de similares) e em *Vis. Document* seguido de "Impressora -> Print" para apresentar dinamicamente um leiaute apropriado para impressão, variando o leiaute padrão de Tab-Folder DHTML, apropriado para entrada de dados.

Agora utilize "F8-Open" para comutar para o formulário de "pesquisa e seleção". A parte de argumentos permite a busca por iniciais do nome e, com configurações de metadados apenas (anotações), pode-se definir novos argumentos e operadores, opções para seleção paginada e outros, assuntos para o próximo artigo. A **Figura 12** ilustra o formulário final de seleção.

Conclusão

Apesar de um maior esforço inicial de configuração, advindo do alto índice de reuso e integração de dezenas de produtos, note que o jCompany Free produz resultados em alto estágio de finalização, com **qualidade de produção, diferentemente de frameworks de mais baixo nível.**

A partir, basicamente, de nosso modelo de domínio mapeado, formulários em JSP e algumas configurações, o jCompany Free foi capaz de prover leiautes ricos completos e de reusar 100% das programações MVC2 necessárias (não programamos nenhuma linha de Java procedimental). E este sistema pode ser embalado de forma distribuída com uma mera opção de configuração que altera seu empacotamento, enviando "um EAR com camada Modelo para lá e um WAR para cá".

Se você tem dezenas ou centenas de Casos de Uso para produzir, certamente perceberá uma diferença considerável nos ganhos em manutenção, se comparar a estratégia OO do jCompany Free com geradores de código que replicam classes por todas as camadas, com base em leiautes pobremente evoluídos ou fatorados.

Nos próximos dois artigos que finalizam a série, iremos discorrer sobre outros Casos de Uso Padrões e suas diversas variações (Inclusões e Extensões Padrões) e também – é claro – sobre como programar regras de negócio MVC2 "por exceção", customizar a arquitetura na camada Bridge ou estendê-la em plugins. Somente aí conheceremos, além da gratuidade (Free-cash), a liberdade total (Freedom) do jCompany Free.

http://jcompany.sourceforge.net
jCompany Free.

http://ant.apache.org
Apache Ant.

http://maven.apache.org
Apache Maven.

http://tiles.apache.org
Apache Tiles.

http://tomcat.apache.org
Apache Tomcat.

http://myfaces.apache.org/trinidad
Apache Trinidad.

http://www.eclipse.org/webtools/main.php
Eclipse WTP.

http://www.eclipse.org
Eclipse.

http://www.hibernate.org/412.html
Hibernate Validator.

http://www.hibernate.org
Hibernate.

Paulo Alvim
alvim@powerlogic.com.br
é Diretor de Tecnologia da Powerlogic SA, autor do livro "Tirando o Máximo do Java EE Open-Source 5 com jCompany Developer Suite" e codificador das primeiras versões do jCompany Developer Suite, mantendo coordenação ativa do projeto **Powerlogic jALM** da Powerlogic.

Rogério Baldini
rogerio.baldini@powerlogic.com.br
tem 10 anos de experiência em tecnologia Java e é atualmente o Product Owner da família jCompany do Powerlogic jALM, solução de Application Lifecycle Management (ALM) da Powerlogic, que integra o jCompany Developer Suite, jCompany QA Suite e jCompany Production Suite. É também o líder do projeto jCompany Free, atualmente catalogando interessados em contribuir com o projeto.

Tirando o Máximo de JAVA EE 5 Open-Source com jCompany Developer Suite, Paulo Alvim, 2008

Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/javamagazine/feedback

Java64.indb 66 28/11/2008 10:34:15