UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

FACULDADES LUIZ MENEGHEL

DANIEL DA SILVA AVANZI

DESENVOLVIMENTO DE COMPONENTES: UMA APLICAÇÃO AO CMS JOOMLA

Bandeirantes

2008

DANIEL DA SILVA AVANZI

DESENVOLVIMENTO DE COMPONENTES: UMA APLICAÇÃO AO CMS JOOMLA

Trabalho de conclusão de curso submetido às Faculdades Luiz Meneghel da Universidade Estadual do Norte do Paraná, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. MSc. Roberto Vedoato

Bandeirantes

2008

DANIEL DA SILVA AVANZI

DESENVOLVIMENTO DE COMPONENTES: UMA APLICAÇÃO AO CMS JOOMLA

Trabalho de Conclusão de Curso submetido às Faculdades Luiz Meneghel da Universidade Estadual do Norte do Paraná, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

COMISSÃO EXAMINADORA

_____

Prof. MSc. Glauco Carlos Silva
Faculdades Luiz Meneghel

_____

Prof. MSc. André Luis Andrade Menolli
Faculdades Luiz Meneghel

_____

Prof. MSc. Roberto Vedoato
Faculdades Luiz Meneghel

Bandeirantes, __ de_____ de 2008

A Deus que esteve em todos os momentos dessa importante etapa da vida.

AGRADECIMENTOS

Ao Prof. MSc. Roberto Vedoato, pela dedicação em todas as etapas do trabalho.

A meus pais, pela motivação e confiança.

A minha irmã, pela força.

Aos amigos, pela força e vibração durante a jornada.

Aos professores, colegas e funcionários da instituição, por serem essenciais para a conclusão do curso.

A todos que colaboraram com a realização deste trabalho, seja direta ou indiretamente.

"ENTREGA O TEU CAMINHO AO SENHOR;
CONFIA NELE, E ELE TUDO FARÁ".

Salmos 37:5

RESUMO

Devido à grande demanda de sistemas computacionais, novas técnicas vêm surgindo para tornar o processo de desenvolvimento mais ágil. No campo de portais e aplicações web, Sistemas Gerenciadores de Conteúdo aparecem como uma valiosa alternativa que não apenas facilita o processo de desenvolvimento, mas também traz um novo conceito de gerenciamento de conteúdo dinâmico. Esses sistemas são definidos como uma página pré-programada com diversas ferramentas administrativas, com funcionalidades baseadas em componentes. Dentre vários Sistemas Gerenciadores de Conteúdo de desenvolvimento, o Joomla merece destaque por ser atualmente o mais usado no mundo e também por ser de código aberto. Entretanto, pode haver situações nas quais os componentes prontos não atendem adequadamente todas as exigências dos clientes, havendo necessidade do desenvolvimento de funções específicas. Nessas situações, o Joomla tem como ponto negativo a escassez de documentação que descreva tal processo. O presente trabalho visa sanar essa necessidade através da documentação das principais características do desenvolvimento de componentes para o Joomla. Nele, é apresentada toda a estrutura de seu funcionamento, mostrando as divisões entre a área do usuário e administrativa. Também é abordada a arquitetura de desenvolvimento do componente, que separa a parte lógica da saída de informação, possibilitando ainda mais a reutilização de código. Funções básicas de segurança, fundamentais para dificultar a ação de invasores, são consideradas as principais classes do desenvolvimento de componentes para o Joomla e são descritas de forma objetiva para que possa ser facilmente empregada. Por fim, uma aplicação prática é realizada nas Faculdades Luiz Meneghel com o objetivo de suprir uma necessidade funcional do *website* da instituição. Neste desenvolvimento todos os procedimentos são documentados visando colaborar com a comunidade Joomla do Brasil.

Palavras-chave: sistema gerenciador de conteúdo; desenvolvimento de componentes; joomla.

ABSTRACT

Due to the great demand of computational systems, new techniques are appearing to make the process of development more agile. In the field of doorways and web applications Content Management Systems appear like a valuable alternative that not only make the process of development easier, but also it brings a new concept of management of dynamic content. These systems are defined as a preprogrammed page with several administrative tools, having his functions based on components. Among several Content Management Systems, the Joomla deserves distinction because it's the most used in the world and also because of it's open source. However, there might occur situations in which the ready components do not take adequately care of all requirements determined by clients, creating the necessity of the development of specific functions. In these situations, the Joomla takes as a negative point the shortage of documentation that describes such a process. The present work aims to cure this necessity through the documentation of the primary characteristics of the development of components for the Joomla. In it, the whole structure of his functioning is presented, showing the divisions between the area of the user and the administrative's. It is also aboards the architecture of development of the component, which separates the logical part of the of information's exit, making possible the reuse of the code. Basic functions of security, basic to make difficult the invaders action, they are considered as a principal class of the development of components for the healthy Joomla, and are described in the objective form so that it can be easily applied. Finally, a practical application is carried out in Faculdades Luiz Meneghel with the objective to provide a functional necessity of the institution's website. In this development all the proceedings are documented aiming to collaborate with the community Joomla of Brazil.

Key-words: content management system; development of components; joomla.

LISTA DE QUADROS

LISTA DE FIGURAS

LISTA DE SIGLAS

HTML	Hyper Text Markup Language
ASP	Active Server Pages
JSP	JavaServer Page
PHP	Hipertext Preprocessor
CMS	Content Management System
FALM	Faculdades Luiz Meneghel
WWW	World Wide Web
HTTP	Hiper Text Transfer Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol
URL	Uniform Resource Locator
FTP	File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
POP	Post Office Protocol
XML	eXtensible Markup Language
COTS	Commercial Off-the-shelf
ESBC	Engenharia de Software Baseada em Componentes
DBC	Desenvolvimento Baseado em Componentes
SQL	Structured Query Language
UI	User Interface

SUMÁRIO

1 INTRODUÇÃO

A *Internet* tem se tornado um dos principais meios de comunicação em todo o mundo. Este meio vem evoluindo a passos largos, das iniciais páginas estáticas nas quais usuários tinham pouca interação e navegavam basicamente através de *hyperlinks* às atuais páginas dinâmicas, que oferecem uma gama de funcionalidades e diferentes formas de interação. Portanto, a internet deixou de ser apenas um repositório de hiperdocumentos para tornar-se também um repositório das mais diversas aplicações (ROLIM, 2006).

Rolim (2006) diz que esta evolução é sentida tanto na perspectiva dos usuários finais quanto na perspectiva dos desenvolvedores que possuem um novo e amplo conjunto de tecnologias para auxiliá-los na tarefa de desenvolvimento de *websites*.

A princípio, o método de criação de *websites* era basicamente através do HTML (*Hyper Text Markup Language*), e todo o conteúdo a ser apresentado era definido linha a linha. Esse estilo de desenvolvimento que inicialmente era adequado a pequenos *websites* com conteúdos estáticos tem se tornado impraticável para maiores *websites* com conteúdos dinâmicos, que apresentam constantes atualizações e que oferecem diferentes funcionalidades (serviços de emails, buscas, compras e outros).

Segundo Santos (2007), algumas linguagens são específicas para o desenvolvimento *Web*, com maior destaque para o PHP (*Hipertext Preprocessor*), o ASP (*Active Server Pages*) e o JSP (*JavaServer Page*). São linguagens que têm suas funções processadas do lado do servidor, enviando apenas o resultado final para o usuário (navegador). Assim, possibilitam que os *websites* funcionem como verdadeiras aplicações, podendo acessar banco de dados, manipularem arquivos textos e outras funcionalidades.

Com o intuito de facilitar o desenvolvimento, sobretudo com relação a implementação de rotinas comuns a vários sistemas *Web*, tem-se os *frameworks* de desenvolvimento *Web* (como *Google Web Toolkit, Struts,VRaptor e CakePHP*) que livra o desenvolvedor de alguns aspectos de baixo nível de implementação (SANTOS, 2007).

Segundo Costa (2007), existem ainda os Sistemas de Gerenciamento de Conteúdo, do inglês *Content Management System* (CMS) que eliminam quase que completamente a programação, exigindo o mínimo ou nenhum conhecimento de baixo nível no desenvolvimento *Web*. Funciona como um sistema gerenciador de sites contendo ferramentas para criação e gerenciamento de conteúdo em tempo real, facilitando o desenvolvimento, administração, distribuição, publicação e disponibilidade de informação.

Com a utilização de um CMS, o proprietário e usuários autorizados têm total autonomia sobre o conteúdo apresentando, dispensando assistência de terceiros, já que atualizam conteúdos sem a necessidade de alterar códigos. Os conhecimentos para sua utilização não vão muito além das habilidades necessárias para um editor de textos. Outro fator de suma importância é o tempo de desenvolvimento, que para *websites* complexos*,* pode ser consideravelmente menor quando comparado com outros métodos existentes, reduzindo assim custos (COSTA, 2007).

Um dos CMS em maior expansão é o Joomla. Possui uma comunidade bastante ativa, devido principalmente por ser de código aberto (*open source*). É um gerenciador que contém diversas extensões, com funcionalidades para diversos propósitos. A grande vantagem de ser *open source* é que os componentes existentes podem ser alterados e componentes novos podem ser implementados (LEMOS, 2008).

O foco deste trabalho é o desenvolvimento de componentes para o CMS Joomla, com o intuito de estudar e desenvolver novas funcionalidades que possam ser de grande utilidade dentro dos projetos das Faculdades Luiz Meneghel (FALM) e também contribuir para a comunidade Joomla.

No próximo capítulo são apresentados aspectos de problemas que podem ser solucionados com a realização deste trabalho, no capítulo 3 é exposta a justificativa do desenvolvimento do mesmo, seguida da descrição dos objetivos a serem atingidos no capítulo 4. A fundamentação teórica do tema escolhido é apresentada no capítulo 5. No capítulo 6 é mostrado os fatores relevantes no desenvolvimento de um componente para o Joomla. Uma aplicação prática do tema estudado é exposta no capítulo 7. E por fim, as considerações finais são descritas no capitulo 8.

2 PROBLEMATIZAÇÃO

É merecedor de grande destaque os aspectos de desenvolvimento *Web*, pois o produto final gerado (sites) é indispensável e de extrema importância para os mais diversos setores. Tanto para sites praticamente estáticos, com poucas funcionalidades, quanto para sites mais complexos o desenvolvimento *web* pode apresentar problemas de produtividade.

O CMS Joomla vem como um abordagem para aumentar a produtividade no desenvolvimento *web* e abstrair o desenvolvedor de aspectos de implementação.

Entretanto, em situações nas quais o CMS Joomla ou suas extensões não supram as necessidades do desenvolvedor, é possível a expansão do gerenciador através da criação de novos componentes (LEBLANC, 2007).

3 JUSTIFICATIVAS

Em alguns casos, para suprir com eficiência os requisitos do *website,* é indispensável a criação de componentes. Apesar da grande comunidade Joomla, grande parte de seus usuários limita-se a utilizar os componentes já existentes. Esse fato ocorre principalmente devido a dificuldade de encontrar uma documentação adequada que descreva o processo de desenvolvimento de componentes.

Este trabalho visa descrever e aplicar em exemplos práticos o processo de desenvolvimento de componentes para o Joomla, contribuindo com a comunidade do CMS de desenvolvimento mais utilizado no mundo.

4 OBJETIVOS

4.1 Objetivo Geral

Este trabalho tem como objetivo geral desenvolver componentes para o gerenciador de conteúdos Joomla para serem aplicados no *website* das Faculdades Luiz Meneghel, gerando uma documentação com a descrição de tal processo.

4.2 Objetivos Específicos

- Pesquisar tecnologias usadas no desenvolvimento *Web* com maior ênfase em CMS;
- Utilizar os conhecimentos adquiridos em lógica de programação através de implementações realizadas principalmente em PHP;
- Colaborar com as comunidades do CMS de código aberto Joomla, disponibilizando informações de extrema relevância no desenvolvimento de componentes para o mesmo;
- Criar um tutorial de desenvolvimento de um componente para o Joomla;
- Aplicar os conceitos adquiridos através da criação de funcionalidades úteis e específicas para auxílio do desenvolvimento do *website* da FALM.

5 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a evolução da *Web*, destacando as principais técnicas de desenvolvimento utilizadas no momento. Com maior ênfase em Sistemas Gerenciadores de Conteúdo.

5.1 Histórico do desenvolvimento *Web*

A *internet* tornou-se o principal meio de comunicação em todo o mundo, sendo assim alvo de grande investimento, tanto em relação ao estudo quanto ao desenvolvimento de suas funcionalidades.

Em 1969, um projeto desenvolvido pelo Departamento de Defesa dos E.U.A. criou a primeira rede de computadores. O trabalho ficou conhecido como ARPAnet e se tratava de uma rede experimental militar com o objetivo de estabelecer uma comunicação entre os computadores. Posteriormente, o acesso à rede começou a ser difundido chegando atualmente há uma série de redes interligadas que formam a *Internet*. A partir de 1983 o crescimento foi surpreendente, estimando-se que a cada momento metade dos conectados a *Internet* obtiveram seu acesso no último ano (SUYAMA, 2007).

Geralmente, o acesso a informações ocorre através do serviço WWW (*World Wide Web* – ou apenas *Web*), que surgiu em 1989 como um integrador de informações consistentes em diferentes plataformas. O hipertexto é a forma padrão das informações contidas na *Web*, permitindo a interligação de vários documentos de diferentes servidores. Sua codificação é feita através do HTML que possui marcas que são interpretadas pelos clientes WWW em diferentes plataformas. A transferência de informações é rápida e objetiva, sendo realizada através do protocolo HTTP (*Hiper Text Transfer Protocol*), que funciona acima dos protocolos TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*) que estabelecem a conexão entre cliente e servidor (OTSUKA, 1997).

Segundo Santos (2007), a localização de recursos na internet é feita pelo HTTP através da utilização de URLs (*Uniform Resource Locator*). Os URLs comuns representam arquivos ou diretórios, mas podem também executar tarefas complexas de pesquisas em banco de dados e *internet*. Essas requisições são

realizadas do cliente para o servidor, podendo haver serviços simultâneos de comunicação quando as portas utilizadas forem diferentes. São exemplos de ocorrências como transferências de arquivos os FTPs (*File Transfer Protocol*) e de gerência de emails o SMTP (*Simple Mail Transfer Protocol*) e o POP (*Post Office Protocol*).

No início, toda a exibição das informações da *Web* ocorria de modo estático. Atualmente estudos estão centrados em diferentes formas de interação dos *websites* com os usuários, com o objetivo de desenvolver aplicações funcionando totalmente na *internet*. A técnica foi desenvolvida pela Netscape, que no início, foi muito criticada por criar soluções fora do padrão exigido, como o JavaScript (ZAKAS, 2006).

A *Web* 2.0 surge sobre perspectiva de interatividade e acessibilidade total. As aplicações *Web* sob esse novo paradigma podem estar aptas a serem executadas nos celulares, videogames e na tão esperada TV digital. O que garante o sucesso é o acesso de multiplataformas (equipamentos diferentes), que permite maior participação dos usuários, que lêem, criam e colaboram mais (ROLIM, 2006).

Para os desenvolvedores, o efeito também é positivo, pois a evolução da *Web* tem melhorado também a técnica de programação, que muitas vezes vem sendo realizada através de modificações de módulos, evitando a programação total de um projeto (ROLIM, 2006).

Os métodos de desenvolvimento de *websites* são muitos. Existe a criação através de pura programação, que descreve o site linha a linha, sendo necessário um grande domínio - por parte do desenvolvedor - das linguagens de desenvolvimento *Web* a serem utilizadas. No entanto, existem ferramentas de criação de páginas (Frontpage, Dreamweaver, entre outras) que facilitam essa exausta programação, que funcionam através da criação gráfica, gerando o código automaticamente da estrutura montada. Essas ferramentas são viáveis em alguns casos (*websites* simples), porém não eliminam totalmente a necessidade de ter conhecimento das tecnologias utilizadas (SANTOS, 2001).

Segundo Costa (2007), recentemente um novo tipo de desenvolvimento tem crescido de forma surpreendente, conhecido como *Content Management System*. Nesse caso, não é necessário o conhecimento de uma

linguagem de programação, e sim, o domínio da ferramenta.

5.2 Sistemas Gerenciadores de Conteúdo

A exigência do mercado faz com que novos métodos de desenvolvimento possam ser criados. Dentre essas exigências, está a grande demanda em *websites* complexos para ser entregues com urgência. Em muitas vezes ao se utilizar métodos convencionais, dificilmente seria possível cumprir com todos requisitos nos prazos estipulados. Outro problema, é que na maioria dos métodos de desenvolvimento, os *websites* são vistos como páginas ligadas ou relacionadas umas as outras, tornando todo meio de navegação estático. Isso quer dizer que a página não muda, a não ser que o desenvolvedor faça as alterações nas descrições de código-fonte (COSTA, 2007).

Millarch (2005) mostra que se tratando de uma empresa realmente ativa, onde constantemente novos clientes são inclusos, produtos novos tem de ser divulgados, há necessidade de mensagens para mídia e parceiros e é natural o desejo de uma publicação das informações em seu *website* para um fortalecimento da imagem institucional.

A maioria das empresas com esse porte tem o seu *website* informacional. Porém, muitas vezes, isso é feito através de um *Webmaster* da empresa – o mesmo funcionário que realiza serviços de manutenção de rede, software e hardware -, através de ferramentas de edição de páginas HTML. Essas ferramentas são ótimas para estruturar o *design,* porém, quando se trata para gerenciar novos conteúdos diversas limitações entram em questão, conforme descritas por Millarch (2005):

- A administração do site geralmente fica atrelada a um computador específico, onde o software está instalado e configurado (senhas de FTP, licença de uso do programa, etc).
- São comuns modificações acidentais nas páginas e em seus códigos, pois o conteúdo, neste caso, está misturado com a lógica de programação. Isto pode resultar em links quebrados,

problemas de formatação, alterações no *design* e até mesmo a indisponibilidade do *website* como um todo.

- Ao longo do tempo, as páginas criadas no *website* ficam “perdidas”. É difícil implementar um mecanismo de pesquisa que filtra e organiza as informações em seções e categorias para que possam ser facilmente encontradas. A arquitetura da informação é comprometida, bem como a experiência e satisfação do usuário final.

Quando há um contrato com uma agência de *design*, que efetua as manutenções e outras solicitações designadas pelo cliente, algumas desvantagens podem vir a ocorrer. Como o alto custo de manutenção, a falta de autonomia e muitas vezes o tempo levado para a publicação dos novos conteúdos (MILLARCH, 2005).

O método proposto para suprir esses problemas, recebe o nome de Sistema de Gerenciamento de Conteúdo, que funciona como um sistema gerenciador de sites, portais e intranets (MILLARCH, 2005).

Segundo Costa (2007), toda a organização do CMS é diferente dos outros métodos de desenvolvimento, pois quase todas as informações exibidas no site funcionam dinamicamente. Assim, o conteúdo pode sofrer alterações conforme as solicitações dos visitantes ou novas informações podem ser adicionadas por membros autorizados. Esse gerenciamento de atualizações não é possível em outros métodos de desenvolvimento encontrados, sendo que essas modificações seriam realizadas apenas por profissionais especializados.

O CMS é caracterizado como um *framework* de alto nível, ou seja, uma estrutura de site pré-programado, que possui recursos disponíveis de manutenção e administração. Assim, a criação, o armazenamento e a gerência ocorre de forma dinâmica, através da *internet* por uma interface interativa de usuário (COSTA, 2007).

Segundo Millarch (2005), a utilização de um CMS em uma empresa, permite total autonomia sobre o conteúdo que deseja ser divulgado dispensando a dependência de terceiros em manutenções rotineiras. Pois, cada membro da equipe poderá gerenciar seu próprio conteúdo, diminuindo os custos com recursos

humanos. Esse gerenciamento ocorre porque as habilidades necessárias para essa administração não passa dos conhecimentos básicos para um editor de texto.

O diferencial dos Sistemas de Gerenciamento de Conteúdo é certamente as modificações do conteúdo que ocorrem de forma rápida e segura de qualquer computador com acesso a *internet*, dispensando o acompanhamento de um especialista (MILLARCH, 2005).

Os CMS atualmente podem ser de código aberto ou proprietários. No entanto, em relação à gerência do conteúdo, os benefícios são os mesmos. Abaixo segue algumas vantagens descritas por Duarte e Lanna (2007):

- Modificações e adaptações na criação de portais podem ser feitas muito rapidamente;
- Filosofia de manutenção de conteúdo: Criadores Múltiplos de Conteúdo (*Multiple Content Creators*);
- Interoperabilidade através de vários menus;
- Curva de aprendizado: rápida e com pouco esforço;
- Não há necessidade de nenhum outro tipo de *software* ou *plug-in,* para criação e manutenção, basta um navegador;
- Funciona de qualquer computador com SO Windows, Linux ou qualquer outro com uma interface X-Windows;
- Páginas *Web* podem ser simples ou complexas.

Nos termos descrito acima, é notável o grande benefício gerado para os desenvolvedores e clientes através da utilização dos Sistemas de Gerenciadores de Conteúdos. De modo simplificado, pode-se dizer que para iniciar seu desenvolvimento, basta escolher a ferramenta que irá usar e começar a gerenciar.

> “O Sistema de Gerenciamento de Conteúdo é uma realidade no mercado, como alternativa não só para a criação, mas, também para manutenção de sites ou portais, simples ou complexos, que acabam por libertar organizações e profissionais liberais do domínio das empresas de *design,* que antes, por deterem o monopólio da informação, cobravam preços que extrapolavam a realidade de alguns pequenos negócios” (LEMOS, 2008).

5.3 JOOMLA

Joomla é uma palavra adaptada foneticamente da expressão "*Jumla"* na língua Swahili (muito utilizada na costa leste da África) para o inglês, e sua pronúncia é "*Djunla"*. Seu significado pode ser "tudo junto" ou "como um todo" (DUARTE e LANNA, 2007).

Dentre as ferramentas CMS, o Joomla recebe grande destaque devido sua grande ascensão, sendo atualmente o gerenciador mais usado em todo o mundo. Há diversas comunidades que colaboram com seu crescimento. A comparação com seu principal concorrente é mostrada nas figuras 1 e 2 (LEMOS, 2008).

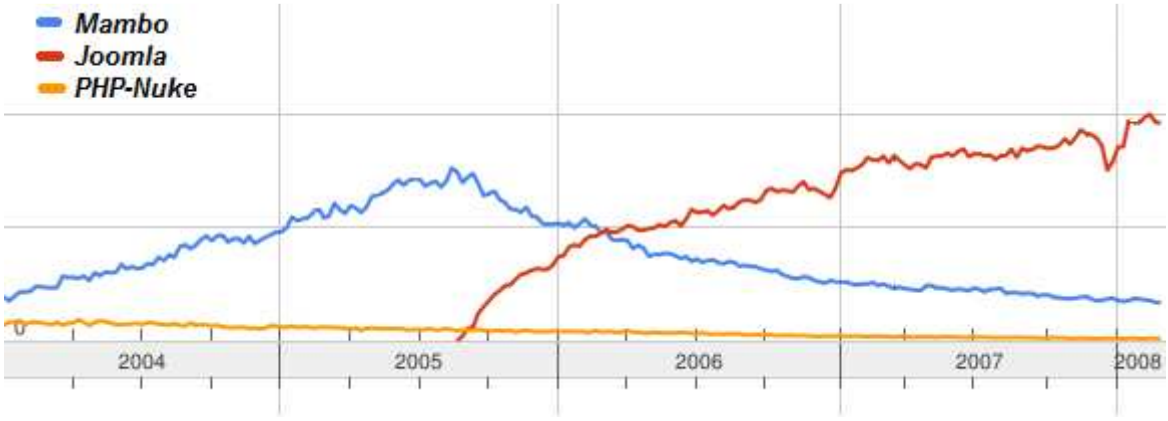

Figura 1: Comparação Mundial dos CMS (LEMOS, 2008).

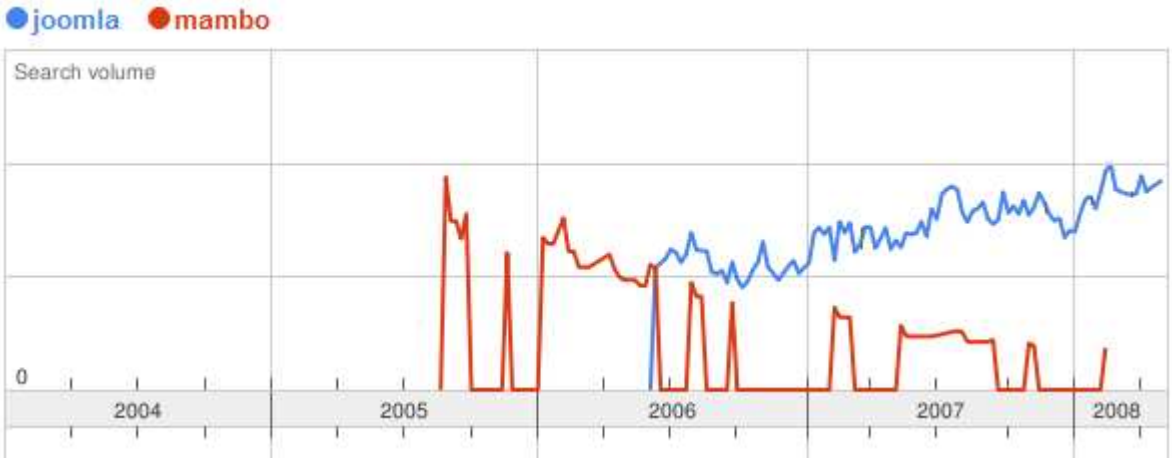

Figura 2: Comparação Brasileira dos CMS (LEMOS, 2008).

O Joomla é um CMS *open source* desenvolvido a partir do Mambo, que por sua vez é mais antigo e também de grande sucesso, porém uma ferramenta proprietária. Joomla é o resultado da separação entre a empresa Miro, detentora dos

direitos sobre o Mambo, e seus desenvolvedores. A separação ocorreu por uma discordância entre ambos devido a uma atitude da empresa transferir o controle do Mambo para uma fundação, conhecida como *Mambo Foundation*, deteriorando a participação dos desenvolvedores (LEMOS, 2008).

Segundo Lemos (2008), devido ao grande sucesso do CMS criado, os desenvolvedores se preocuparam com a integridade do projeto e com o futuro de seus usuários, abandonando a empresa e lançando um novo gerenciador totalmente derivado do Mambo, porém com a vantagem de ser *open source*.

O fato de ser de código aberto contribuiu para que em menos de dois anos desde o lançamento da primeira versão - conhecida como Joomla 1.0 -, o Joomla tenha se tornado o sistema de gerenciamento com maior número de usuários e recursos disponíveis. Esse crescimento é devido também à diversidade de extensões extras, criadas não necessariamente pelos desenvolvedores do projeto, e sim por usuários que desenvolvem funções para suprir suas necessidades específicas (DUARTE e LANNA, 2007).

5.3.1 Tecnologias Utilizadas no Joomla

O Joomla é escrito em PHP, caracterizada como uma linguagem de programação interpretada, livre e muito utilizada para gerar conteúdo dinâmico na *Web*. Desde seu surgimento em 1994, o PHP está incluso nas mais diversas formas de desenvolvimento, inclusive nas técnicas assíncronas, recentemente estruturadas (NIEDERAUER, 2005).

Segundo Niederauer (2005), apesar de ser de fácil interpretação e utilização para usuários da área, o PHP é uma linguagem robusta que possibilita algumas funções de orientação a objeto, sendo então uma das mais utilizadas na *Web*.

Possui muitas características que apóiam seu uso, como o fato de ser gratuito e de código aberto. Outro detalhe é que pode ser misturado com o HTML, ou seja, pode escrever um trecho em PHP seguido por HTML e assim por diante. O que mais destaca em sua utilização *Web*, é o fato de ser baseado no servidor. Assim, quando é acessada uma página PHP através do navegador, todo o código é executado no servidor, exibindo somente o resultado final para o usuário.

Como a página exibida no navegador já é processada, o consumo de recursos do computador do cliente é menor (NIEDERAUER, 2005).

Segundo Niederauer (2005), quando se trata de banco de dados, o PHP também recebe seu destaque, possuindo *drivers* para os mais populares servidores de banco de dados: MySQL, PostgreSQL, SQLite, InterBase, Oracle, SQL Server e outros.

Dentre os bancos de dados citados, o Joomla utiliza o MySQL, que por sua vez, também é livre como o CMS em estudo e sua linguagem utilizada (PHP). Atualmente é o banco de dados de código aberto mais popular do mundo, com mais de 6 milhões de instalações entre *websites*, *datawarehouse*, aplicações comerciais e outras mais. Essa popularidade é devido a sua consistência, alta performance, confiabilidade e fácil usabilidade. É desenvolvido pela empresa MySQL AB, que tem como missão criar um banco de dados superior que esteja disponível a todos (MYSQLBRASIL, 2008).

5.3.2 Áreas do Joomla

Quando é feita a instalação do Joomla em um servidor *Web*, é criado automaticamente duas áreas distintas que separam a gerência da aplicação pública.

5.3.2.1 Área de Usuários (*Front-end*)

Segundo Gameleira (2006) *Front-end* é a área do *website* disponível para o público, caracterizado como um *website* normal da *Web*, com a vantagem de permitir que usuários autorizados façam alterações do seu conteúdo diretamente no *website*.

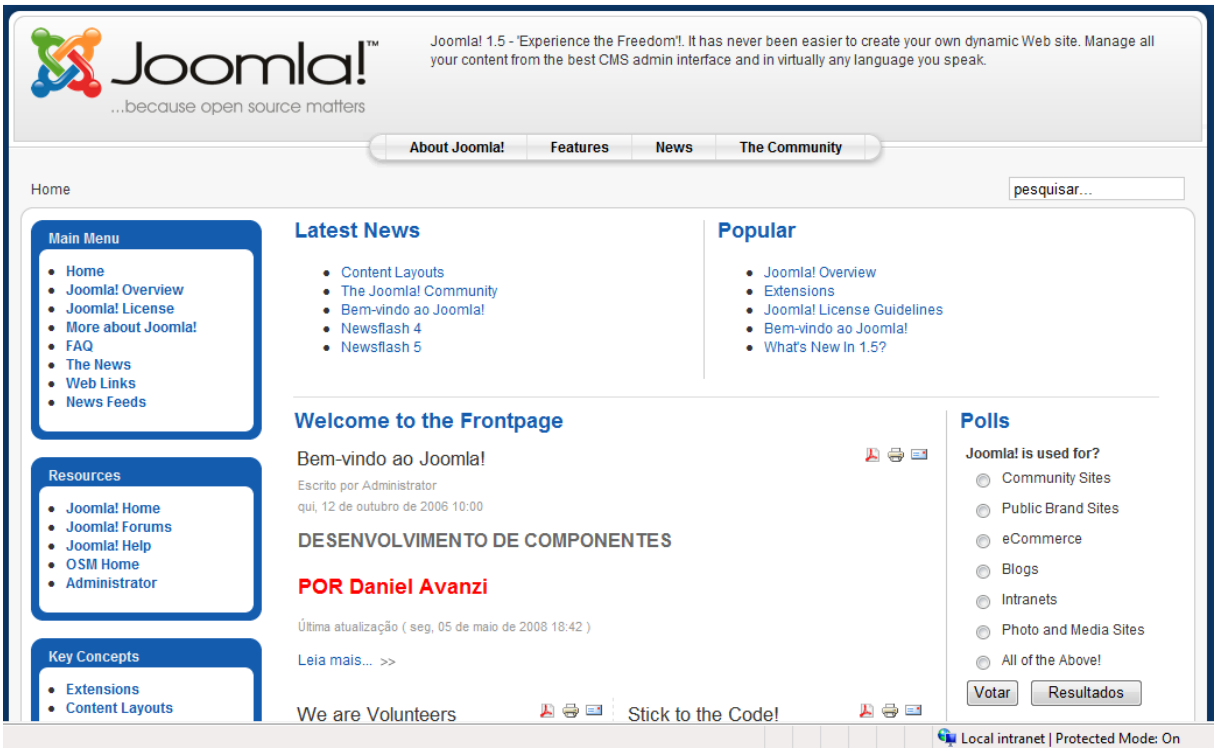

Figura 3: *Front-end* do Joomla.

Na Figura 3 é mostrada a visão dos usuários com restrições, podendo ser um visitante, editor ou publicador, definidos por Duarte e Lanna (2007) como:

-Público: todo mundo (permissão apenas para leitura e navegação);

-Registrado: membros do portal (leitura e *downloads*);

-Autor: cria o conteúdo, mas não pode publicar;

-Editor: edita conteúdos novos e existentes;

-Publicador: aprova um conteúdo para publicação.

5.3.2.2 Área de Administração (*Back-end*)

Back-end é a área onde somente o administrador tem acesso, pois aqui é determinado toda a estrutura que será exibida no *website* público, como menu, notícias, cadastros, *downloads* e outros. Sua exibição exige autenticação (GAMELEIRA, 2006).

Figura 4: *Back-end* do Joomla.

Na figura 4 é ilustrado o *Back-end* do Joomla, nele os itens exibidos dependem do nível de usuário definido para área de administração. Qualquer estrutura de exibição do *website* é definida nessa área. Duarte e Lanna (2007) definem os usuários como:

-Gerente: gerencia o conteúdo do portal;

-Administrador: permissão para instalar extensões e criar contas;

-Super Administrador: controle total.

O conjunto dessas duas áreas de controle forma uma ferramenta CMS completa, permitindo o gerenciamento da estrutura e do conteúdo por diversos usuários, cada um com sua participação e responsabilidade.

5.3.3 Extensões

Segundo Duarte e Lanna (2007), as extensões são instaladas no Joomla somente pela área *Back-end* a partir de pacotes contidos em arquivos .zip (formato de compactação de arquivos), cada extensão deve conter um arquivo no formato XML (*eXtensible Markup Language*) com as informações de instalação. A área que permite a instalação de extensões é mostrada na figura 5.

Figura 5: Instalação de Extensões.

5.3.3.1 *Templates*

Segundo Gameleira (2006), um *template* não é um *website* e também não deve ser considerado um *design* completo do mesmo. *Template é* um conjunto de arquivos que contém a disposição (localização) das informações que o Joomla usa para controlar a apresentação do conteúdo.

Em um *website*, mais de um *template* pode ser usado para produzir o efeito completo do mesmo. Ele funciona em conjunto com o conteúdo armazenado no banco de dados Joomla, disponibilizando as informações de acordo com sua estrutura. Sua utilização permite que com pouco esforço e tempo, toda a estrutura do *website* possa ser modificada (GAMELEIRA, 2006).

Segundo Duarte e Lanna (2007), existem *templates* para área de *Front-end* e *Back-end*, ambos podendo ser provindos de diferentes fontes: os baixados e instalados gratuitamente, os comprados (geralmente para ser utilizado em algo específico) e os desenvolvidos, que refletem as necessidades específicas

do desenvolvedor. A figura 6 mostra a utilização do mesmo *template* na estrutura de dois *websites*.

Figura 6: Utilização do mesmo *template* (DUARTE e LANNA, 2007).

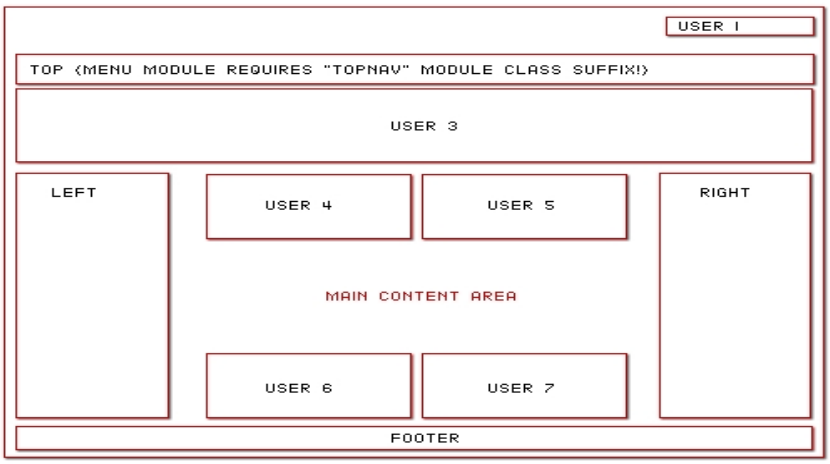

Figura 7: Exemplo da estrutura de um *template* (GAMELEIRA, 2006).

5.3.3.2 Módulos

A vantagem de um *website* dinâmico é possuir inúmeros recursos nativos do sistema que podem ser habilitados quando necessários. Essas funcionalidades que tornam o *website* interativo no Joomla são possíveis também através dos módulos, representado na figura 8.

Segundo Gameleira (2006), os módulos são extensões que ampliam a capacidade do Joomla agregando novas funcionalidades de exibição,

disponibilizando as informações contidas no banco de dados em qualquer lugar do *template*.

Sua instalação é feita somente pela área de *Back-end*, sendo que cada módulo pode ter seu próprio parâmetro de configuração. É responsável por consultar dados como *login*, enquetes, ferramenta de busca, gerenciador de menus e outras, permitindo o acesso ao banco de dados quando necessário (DUARTE e LANNA, 2007).

Figura 8: Módulos (DUARTE e LANNA, 2007).

Gameleira (2006) destaca que apesar dos módulos terem acesso ao banco de dados, eles não podem fazer alterações. Sua função é simplesmente a consulta dos dados e a estrutura de exibição dessas informações.

Segundo Duarte e Lanna (2007), os recursos nativos são aqueles que estão disponíveis para uso, sem que precise ser adicionado pacote algum ao seu Joomla, como:

- Cadastramento de usuários e proteção de senha (acesso);
- Painel administrativo geral para gerenciamento de conteúdo;
- Enquête e pesquisas de opinião;
- Notícias;
- Banners comerciais;
- Sistema de busca dentro do *website* com estatística.

Segundo LeBlanc (2007), diferente dos componentes, os módulos podem ser aplicados diversas vezes na mesma página, possibilitando a execução de diferentes menus.

5.3.3.3 Componentes

São extensões que possuem funcionalidades específicas para suprirem as necessidades do *website.* Quando a funcionalidade a ser implantada no *website* não está disponível nos componentes nativos e ausentes também nos encontrados na *Web*, pode ser desenvolvido um novo componente especificamente para satisfazer a necessidade do seu *website* (GAMELEIRA, 2006).

Posteriormente, a funcionalidade desenvolvida pode ser distribuída para que outros desenvolvedores possam utilizá-la em seus projetos, agilizando o processo de criação e enriquecendo a ferramenta Joomla.

Segundo Gameleira (2006), a grande diferença entre os componentes e os módulos, é que o primeiro permite a manipulação das informações contidas no banco de dados, diferente dos módulos que apenas realizam a leitura.

LeBlanc (2007), diz que de todas as extensões do Joomla, o componente é o mais essencial. Pois ele que representa a funcionalidade ocorrida na tela, sendo possível a execução de um componente por vez.

Figura 9: Componente não nativo do Joomla (DUARTE e LANNA, 2007).

5.4 Desenvolvimento de Componentes

Segundo Pfaffenseller (2000), o desenvolvimento de aplicações é um processo desgastante que consome recursos financeiros e pessoais. Como os sistemas têm se tornado cada vez maiores, há um grande necessidade de buscar mecanismos para reduzir sua complexidade.

A programação orientada a objetos trás novos conceitos (classes, objetos, herança, encapsulamento, associação e outros) que proporcionam um grande avanço no desenvolvimento de aplicações. Esses conceitos possibilitam a reutilização de código, que vem amenizar os problemas de consumo excessivo de recursos (DAVID, 2007).

Segundo Almeida (2002), a reutilização é caracterizada pela utilização de produtos de *software* em uma situação diferente daquela para quais estes produtos foram originalmente construídos. A reutilização pode ser muito bem empregada através do uso de componentes.

Componente é o nome dado para o elemento de *software* que encapsula uma série de funcionalidades. Suas funcionalidades são independentes de qualquer outra unidade de *software,* podendo ser utilizados com outros componentes para a formação de um sistema mais complexo (SILVA, 2000).

Brown e Wallnau (1996) definem o componente como: "*uma não-trivial, quase independente, e substituível parte de um sistema que cumpre uma função clara no contexto de uma arquitetura bem definida*". Eles possuem uma interface própria, possibilitando assim seu uso em diferentes aplicações. Aplicam também o conceito de herança, sendo esse similar a programação orientada a objeto.

Os componentes podem ser utilizados em diferentes sistemas com a mesma funcionalidade ou serem adaptados conforme a necessidade. Geralmente os que não mudam suas funcionalidades são conhecidos como "*comerciais de prateleiras*" (*ou commercial off-the-shelf* - COTS). Os COTS são geralmente comprados e em sua maioria não possuem código-fonte, fazendo que o engenheiro de *software* não tenha um conhecimento profundo do funcionamento interno. Assim, a integração do componente com o *software* pode não ser perfeita, deixando a desejar em suas funcionalidades (BROWN e WALLNAU, 1996).

O desenvolvimento orientado a componentes organiza a estrutura de um *software* como uma interligação entre aplicações (componentes) independentes. O reuso de componentes previamente desenvolvidos, permite a redução de tempo e esforço para a obtenção de um *software*. No entanto, em alguns casos sua adequação a uma situação específica pode ser muito complexa, sendo mais viável o desenvolvimento de um novo componente (SILVA, 2000).

Almeida (2002) diz que ao desenvolver um componente para uma devida aplicação, a maior preocupação é na futura reutilização do mesmo em outros sistemas. Para melhor adaptação posterior, a sistematização do processo de análise e criação de componentes se torna um passo fundamental. Para que a reutilização possa ser efetiva devem ser consideradas todas as fases do processo de desenvolvimento de software, com os métodos, técnicas e ferramentas que suportem desde a identificação e especificação do componente e do domínio do problema, até o seu projeto e implementação já realizados em uma aplicação.

Para que uma aplicação seja desenvolvida baseada em componentes, é necessário o uso do processo de Engenharia de *Software* Baseada em Componentes (ESBC). Parecida com a engenharia de *software* convencional ou a orientada a objetos, exige como passo inicial uma análise de requisitos complexa definida por uma equipe de desenvolvedores, seguida pela definição do *design* da arquitetura. O próximo passo descrito por Brown e Wallnau (1996), difere da engenharia convencional, pois nessa etapa ao invés de desenvolver as funcionalidades exigidas nos requisitos, outros passos serão seguidos, devendo ser verificados se:

- há "*componentes de prateleiras"* disponíveis para implementar o requisito;
- componentes re-usáveis desenvolvidos internamente estão disponíveis para implementar o requisito;
- as interfaces para os componentes disponíveis são compatíveis dentro da arquitetura do sistema a ser construído.

A prioridade é sempre a utilização de componentes já existentes, podendo inclusive ser alteradas algumas partes dos requisitos de acordo com o componente encontrado. O processo de ESBC, além de identificar os possíveis componentes, também ajusta a interface de cada um para que não haja incongruência arquitetônica. Outros dois processos ocorrem junto a Engenharia de *Software* Baseada em Componentes (BROWN e WALLNAU, 1996):

1) Engenharia de Domínio: Objetiva identificar, construir, catalogar e disseminar um conjunto de componentes de *software* que tenham usabilidade para aplicações existentes e futuras, dentro de um domínio de aplicação específico, catalogadas por suas funções similares. O objetivo é estabelecer um mecanismo em que engenheiros de *software* possam compartilhar estes componentes.
2) Desenvolvimento Baseado em Componentes (DBC): qualificação, adaptação e composição.

A etapa de que examina os componentes re-usáveis, para averiguar

se cumprem com os requisitos do sistema é chamada de qualificação. São analisados aspectos como desempenho, confiabilidade e usabilidade.

As modificações de alguns aspectos do componente ocorrem na etapa de adaptação. É um passo essencial, pois dificilmente um componente não necessitará de alguns ajustes para seu reuso. As abordagens mais comuns de adaptação são (BOSCH, 1999):

- Encapsulamento caixa-branca (*white box wrapping*): a implementação do componente é diretamente modificada para resolver incompatibilidades. É possível apenas com o acesso ao código-fonte, portanto, os componentes COTS dificilmente poderão ser adaptados por essa abordagem;
- Encapsulamento caixa-cinza (*grey box wrapping*): a adaptação é feita através do uso de uma biblioteca do componente que fornece uma linguagem de extensão do mesmo, possibilitando a remoção dos conflitos;
- Encapsulamento caixa-preta (*black box wrapping*): é o caso mais encontrado, onde não é possível ter acesso ao código-fonte, sendo necessária a adaptação do componente pela interface.

O fim do processo de adaptação se resume em constantes testes para verificar a existência de erros. Em alguns casos o desenvolvimento de um componente acaba sendo mais viável do que a adaptação de um existente, a escolha depende apenas do engenheiro de *software* que analisará o caso.

E por fim, a etapa de composição irá agrupar os componentes permitindo a realização de tarefas. Essa integração será feita por uma biblioteca especializada de componentes e serviços (BROWN e WALLNAU, 1996).

Então, para os componentes serem reutilizados, precisarão ser qualificados e adaptados. Caso a reutilização não supra todas as necessidades, um novo componente deve ser desenvolvido, sempre seguindo os padrões criados para seu desenvolvimento, para uma futura reutilização.

6 DESENVOLVIMENTO DE COMPONENTES PARA O JOOMLA

Segundo Gameleira (2006), o conceito de componentes quando se fala do Joomla, é exatamente o mesmo estudado na ESBC. Os componentes são elementos de conteúdo que funcionam como pequenas aplicações, tendo como ponto principal o objetivo de reutilização de código.

O termo reutilização vem justamente do ponto em que os desenvolvedores pegam os componentes já prontos e aplicam em seus *websites* (GAMELEIRA, 2006).

Entretanto, ao se utilizar o CMS Joomla, algumas funcionalidades necessárias para cumprir os requisitos de um projeto podem não estar disponíveis. Nesse caso, componentes que cumpram funcionalidades específicas do projeto devem ser desenvolvidos (GAMELEIRA, 2006).

Segundo Mendes (2005), a base do desenvolvimento está na linguagem PHP. O componente deve conter alguns arquivos básicos para seu funcionamento, como:

- Arquivo responsável pela lógica do componente;
- Arquivo que controla a saída das informações;
- Arquivo de classes abstratas que ligam ao banco de dados;
- Arquivo responsável pela instalação;
- Arquivo de remoção do componente;
- Arquivo XML que armazena todas as informações.

Os arquivos apresentados são os básicos no desenvolvimento de componentes, podendo variar de acordo com a complexidade do mesmo (MENDES, 2005).

Duarte e Lanna (2007) concluem que após o desenvolvimento do componente, os arquivos devem ser adicionados a uma compactação do formato .zip e instalados pela área *back-end* do Joomla.

Devido ao Joomla ser um Sistema de Gerenciamento de Conteúdos relativamente novo, as informações de desenvolvimento de componentes é escassa.

O projeto a ser implantado tem como interesse documentar tal procedimento e aplicar um componente que seja de benefício para FALM.

6.1 Estrutura Básica

Todo componente deve conter alguns arquivos básicos considerados essenciais para seu funcionamento. A estrutura de funcionamento do componente possui uma arquitetura moderna, adotando técnicas que separam a utilização da administração, e a lógica do conteúdo (LEBLANC, 2007).

6.1.1 *Back-end*

Segundo LeBlanc (2007), a parte administrativa do componente do Joomla, ou seja, todos os arquivos que representarão as funcionalidades do *back-end*, devem ser armazenadas no diretório *administrator*/*components*/ dentro de uma pasta nomeada com o nome do componente com um prefixo *com_*. Essa pasta deverá conter arquivos que representam as funções a seguir:

- admin.XXX.php - Esse arquivo será o primeiro a ser executado quando for acessado a área *back-end* do componente Joomla. Ele deve conter todas as funcionalidades do componente, podendo se estender a outros arquivos quando o código for muito complexo.
- admin.XXX.html.php - Arquivo que contém todas as formatações de saída de informações. Todo conteúdo a ser exibido é gerado a partir desse arquivo.
- tabelas/XXX.php - As tabelas devem ficar em uma pasta localizada junto com os arquivos administrativos do componente. Esse tipo de arquivo deve conter uma classe descrevendo os campos da tabela e os atribuindo o valor *Null*.
- toolbar.XXX.php - É o arquivo responsável por armazenar todas as funcionalidades das opções dispostas na barra de ferramentas do componente. Possui implementações do tipo de inserção, exclusão, publicação, edição e outras

funcionalidades básicas na administração do componente.

- toolbar.XXX.html.php - Arquivo que contém os códigos necessários para exibição das opções da barra ferramentas. É mostrado como uma barra disposta de ícones que irá chamar a respectiva solicitação, como mostra a figura 10.

Figura 10: Itens da barra de ferramentas.

6.1.2 *Front-end*

LeBlanc (2007), diz que os arquivos que realizarão as funcionalidades voltadas para a área dos usuários (*front-end*), devem ficar em *components* dentro do mesmo padrão de pasta descrito na área administrativa: *com_nome_componente*.

Deve conter dois arquivos básicos na pasta:

- XXX.php - Primeiro arquivo a ser acessado na execução do componente quando sua utilização é no *front-end*. Contém toda a lógica do componente, ou seja, dentro desse arquivo está contido toda a lógica de programação do mesmo.
- XXX.html.php - Pega os resultados gerados pelo arquivo anterior, e os exibe na tela, possibilitando a interação do usuário com o componente.

6.2 Funções Básicas

Algumas classes do *framework* Joomla são necessárias no desenvolvimento de componentes, pois possuem funções que auxiliam na administração do banco de dados e controle de entrada e saída de informações. O domínio da linguagem PHP fica indispensável na utilização dessas classes (LEBLANC, 2007).

6.2.1 Segurança

Segundo LeBlanc (2007), em todo desenvolvimento *Web* a questão de segurança é indispensável nos dias atuais. Diversos tipos de interesses fazem com que a vontade em burlar os sistemas cresça a cada dia. O principal esquema de segurança no desenvolvimento de componentes para o Joomla é barrar os usuários que tentam o acesso sem executar o *framework.*

Esse esquema funciona da seguinte forma: uma constante do *framework* é instanciada assim que o mesmo é acessado; todo arquivo do componente deve fazer a consulta da constante para continuar com a execução.

- *defined('_JEXEC') or die ('Acesso Indevido');*

A função *defined* é do PHP, e ela verifica se uma dada constante existe e é definida, retornando um resultado do tipo lógico. A constante *JEXEC* é do joomla, e representa se o mesmo está sendo executado através do formato lógico.

6.2.2 Principais Classes para o Desenvolvimento de Componentes

- JToolbarHelper: Usado para inserir imagens de ferramentas administrativas no *back-end*. Possui funções como:

:title () – insere um texto e uma figura na barra administrativa de componentes;

:save() – exibe o botão de salvar;

:apply() – botão de aplicar;

:cancel() – cancelar;

:publishList() – publicar;

:unpublishList() – despublicar;

:editList() – editar;

:deleteList() – deletar;

:addNew() – adicionar;

:custom() – adiciona as próprias imagens.

- JFilterOutput: possui funções que trabalham com saídas dinâmicas:

:ampReplace() – forma de HTML dinâmico que executa uma ação ao clicar;

:cleanText() – limpa o texto de toda formatação;

:objectHTMLsafe() – cria um objeto seguro para exibir no formulário;

:stringURLsafe() – Processa as strings e substitui todo caractere UTF-8 por não acentuado ASCII-7.

- JHTML: possui funções que geram HTML para serem usados na criação de elementos gráficos como as caixas de seleção. Para melhor desempenho, essas funções são carregadas na memória apenas quando necessário:

:_() – a função a ser executada é passada como primeiro parâmetro, seguido dos parâmetros adicionais (se houver).

:_('behavior.calendar') – adiciona um JavaScript que mostra um calendário;

:_('grid.id') – mostra uma caixa de seleção;

:_('grid.published') – mostra figuras que representam a publicação;

:_('select.genericList') – cria um *combobox*.

- JApplicationHelper: é uma classe estática que possui funções de solicitações:

:getPath() – adiciona arquivo que contém no nome: *admin_html, front_html* e *class*;

:getClientInfo() – recebe informações de um determinado cliente (id).

- JTable: possui funções que administram os registros da tabela através de ações como inserção, leitura, exclusão e atualização:

:construct() – construtor da classe;

:addIncludePath() – inclui as tabelas contidas no caminho descrito como parâmetro;

:getInstance() – retorna os dados da tabela indicada;

:delete() – método padrão de exclusão de tabelas;

:load() – carrega uma fila do banco de dados e atribui as propriedades do objeto;

:reset() – volta para as propriedades padrão;

:bind() – associa os valores com o objeto;

:store() – insere uma nova linha ou modifica uma antiga;

:toXML() – exporta os itens da lista para XML.

- JRequest: Esta classe serve para fornecer ao Joomla uma interface comum de acesso às variáveis requisitadas. O tipo de requisição pode ser $_POST, $_GET e o próprio $_REQUEST. Durante a transmissão, as variáveis passam através de um filtro de entrada que evita novas injeções.

:get() – retorna uma solicitação de matriz;

:getVar() – retorna uma determinada variável;

:set() – seta a variável requerida;

:clean() – limpa uma requisição de um *script* injetado;

:getBool() – usado para retornar variáveis lógicas.

- JLoader: Classe que importa arquivos:

:JImport() – importa a biblioteca desejada;

:JExit() – uma função global para encerrar o *framework*;

:__autoLoad() – importa as principais classes automaticamente, nunca sendo imposta pelo desenvolvedor.

- JFactory: trabalha com o objeto corrente:

:getDBO() – pega o objeto do banco de dados

:getLanguage() – retorna a linguagem global do objeto;

:getMailer() – pega as referências do objeto de controle de emails.

- JDataBase: classe pai de todas as tabelas:

setQuery() – armazena um consulta SQL para usar mais tarde;

getInstance() – retorna as propriedades do objeto de banco;

getQuery() – retorna a consulta corrente;

loadObjectList() – retorna uma matriz com os registros do banco;

query() – executa a consulta;

getDBO() – retorna propriedades do banco;

6.3 Organização

Para que realmente a funcionalidade desenvolvida possa ser considerada um componente, deve-se criar um arquivo de instalação que organize todas as pastas e arquivos a modo que possam ser implantadas em outros sistemas (DUARTE e LANA, 2007).

Segundo LeBlanc (2007), esse arquivo é do formato .zip, pois essa extensão é a única que o Joomla reconhece na instalação de arquivos. O primeiro arquivo que o Jooma vai procurar nessa pasta compactada, é o do formato XML. Seu nome vai ser o mesmo do componente, e deve conter toda distribuição dos arquivos contidos, incluindo a chamada ao código que irá criar as tabelas.

Em geral, o arquivo XML armazena informações como:

- Versão do XML;
- Codificação de caracteres;
- Tipo de instalação (“*component*”);
- Versão do componente;
- Autor;
- Data de criação;
- Email do criador;
- Tipo de licença;
- Descrição do componente;

- Arquivo de instalação;
- Arquivo de desinstalação
- Instalação;
- Desinstalação;
- Arquivos do *front-end*;
- Arquivos do *back-end*;
- Inserção na barra de ferramentas do componente.

O arquivo de instalação contém as informações que serão exibidas ao usuário após a instalação do componente. É do formato PHP e geralmente apresenta apenas uma mensagem de instalação concluída. O arquivo de desinstalação tem o mesmo funcionamento, mas para enviar a mensagem de remoção concluída.

Segundo LeBlanc (2007), a instalação é nada menos que a criação das tabelas que o componente irá usar. É chamado um arquivo com extensão SQL (*Structured Query Language*, do português Linguagem de Consulta Estruturada), contendo todo o código (em SQL) para criar as tabelas. A desinstalação é um arquivo do mesmo gênero com códigos SQL de exclusão das tabelas. Todos os arquivos são citados no código, pois sua execução só é realizada caso esteja contido no arquivo XML.

O arquivo *.zip* deve conter todo o código responsável pela instalação do componente disponível no primeiro nível de pasta, juntamente com os arquivos responsáveis pelo *front-end*. Dentro de uma subpasta deve conter os arquivos que representam as funcionalidades administrativas, seguido de uma outra subpasta que armazena as tabelas, como mostra a figura 11 (LEBLANC, 2007).

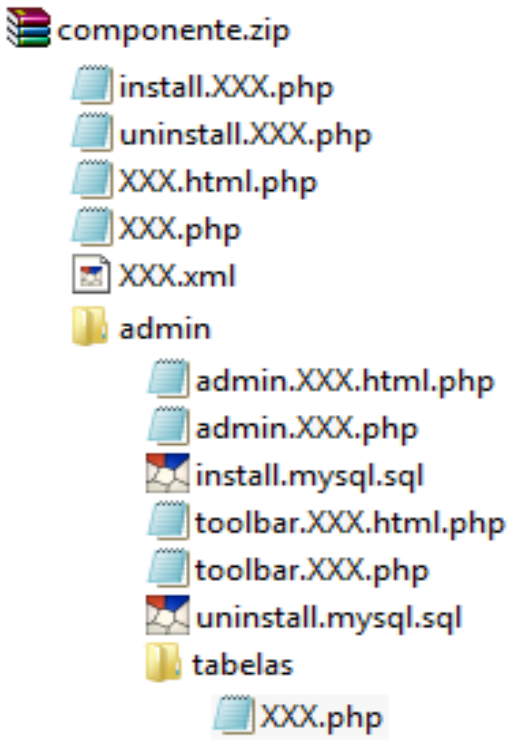

Figura 11: Estrutura do Arquivo de Instalação.

7 CONSTRUÇÃO DE UM COMPONENTE PARA FALM

O objetivo deste capítulo é por em prática os conceitos apresentados no desenvolvimento de um componente do JOOMLA, criando algo útil para ser utilizado na FALM. O componente tem como objetivo listar cursos disponíveis em uma suposta semana de informática, cadastrados e detalhados pelo administrador, e possibilitar a inscrição dos alunos verificando se o mesmo já está cadastrado na semana de informática. Apesar de parecer algo comum, o componente adota todos os métodos recomendados em seu desenvolvimento.

Sua abordagem de desenvolvimento é baseada no encapsulamento caixa-branca, que proporciona total domínio das funcionalidades do componente através do acesso ao código-fonte.

7.1 Desenvolvimento do *Back-end*

Antes de começar com as definições de código, uma pasta deve ser criada para armazenar todos os arquivos que representarão a funcionalidade administrativa do componente. Essa pasta vai receber o nome do componente com o prefixo *com_* adicionado, e deve ficar no diretório *administrator/components*.

Quando o componente for chamado pelo *back-end*, automaticamente o Joomla retira o prefixo padrão, e executa o arquivo PHP que tiver o mesmo nome. Como as definições são da parte administrativa, os arquivos PHP devem receber o prefixo *admin* para melhor organização do componente.

7.1.1 Executando o Componente

Toda requisição do *back-end* é feita a partir do diretório *administrator/index.php*. Para ter acesso ao componente (como ainda não está criado um *link*) é acrescentado na URL uma variável com o nome de *option* que recebe o nome da pasta onde estão contidos os arquivos do componente, como a seguir: *http:/.../seusite/administrator/index.php?**option=com_curso***. Deve-se criar o arquivo *admin.curso.php* e adicionar algumas linhas apenas para testar o caminho de acesso como mostra o quadro 1. O resultado é mostrado na figura 12.

```
<?php       //admin.curso.php
  defined( '_JEXEC' ) or die( 'Acesso Restrito' );
  echo '<div class="componentheading">Controle de Curso</div>';
?>
```

Quadro 1: Primeiras definições da área administrativa.

Figura 12: Primeiro acesso ao componente de curso.

A função *defined*() serve para impedir o acesso ao componente fora do Joomla. Assim que o framework é executado, o valor da variável *JEXEC* passa de *false* para *true*, executando assim o restante do código, dando maior segurança em relação à acessibilidade.

7.1.2 Registrando o Componente no Banco de Dados

Digitar a URL do componente não é a única forma de acesso a ele. Pode ser adicionado nas barras de administração do Joomla a gerência do componente. Para tal efeito, algumas informações devem ser adicionadas no banco de dados, dentro da tabela responsável pelos componentes a serem exibidos, com os valores e características desejadas. Para inserir deve-se executar o código SQL descrito no quadro 2.

```
INSERT INTO jos_components (name, link, admin_menu_link, admin_menu_alt, `option`,
            admin_menu_img, params)
VALUES ('Curso', 'option=com_curso', 'option=com_curso',
           'Gerenciar Curso', 'com_curso', 'js/ThemeOffice/component.png', '');
```

Quadro 2: Adicionar o componente na barra de ferramentas.

Name: nome do componente;

Link: link do front-end;

Admin_menu_link: link do back-end;

Admin_menu_alt: descrição do componente;

Option: pasta do componente;

Admin_menu_img: imagem.

É importante ressaltar que a inserção de informações na tabela responsável pelo controle de componentes, deve ocorrer apenas no desenvolvimento do componente, pois quando já pronto o mesmo, o arquivo de instalação executa tal processo de forma diferente, podendo ser observado na descrição do arquivo XML contido no fim deste trabalho.

Com a inserção das informações do componente no banco de dados, um *link* para acesso da área do *front-end* pode ser facilmente adicionado no menu principal do site da seguinte forma: Menus > Main Menu > Novo > Link Interno > Curso > Insira um Título > Salvar. A figura 13 mostra o *link* do componente sendo adicionado.

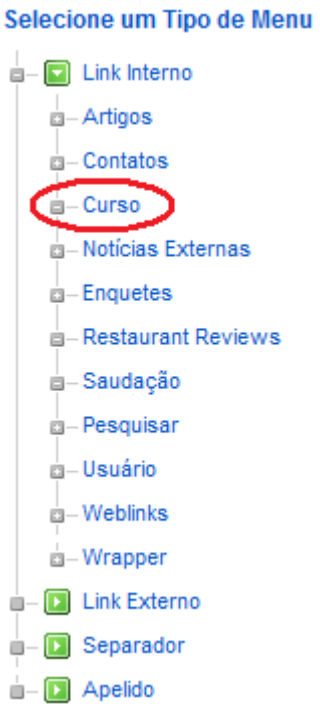

Figura 13: Criando um link de acesso rápido para o *front-end*.

O *front-end* irá apresentar o *link* na página principal no lugar que o administrador escolher, como mostra a figura 14.

Figura 14: Link de acesso no *front-end*.

7.1.3 Criando a Barra de Ferramentas

Todos os componentes do Joomla podem herdar os botões básicos de Salvar, Deletar, Editar e Publicar. Para começar, um arquivo do tipo *php* deve ser criado com o nome de *toolbar.curso.html.php* dentro do diretório *administrator/ components/com_curso* como mostra o quadro 3.

```
<?php            //toolbar.curso.html.php
  defined( '_JEXEC' ) or die( 'Acesso Restrito' );
  class TOOLBAR_curso {
    function _NEW() {
      JToolBarHelper::save();
      JToolBarHelper::apply();
      JToolBarHelper::cancel();
    }
    function _DEFAULT() {
      JToolBarHelper::title( JText::_( 'Controle de Curso' ), 'generic.png' );
      JToolBarHelper::publishList();
      JToolBarHelper::unpublishList();
      JToolBarHelper::editList();
      JToolBarHelper::deleteList();
      JToolBarHelper::addNew();
    }
  }
?>
```

Quadro 3: Criando a barra de ferramentas.

Os arquivos que contém os códigos responsáveis pela saída de informações geralmente são organizados em classes. No caso apresentado a classe definida é a TOOLBAR_curso, cujo cada função representa as ferramentas que estarão contidas na barra. A classe *JToolBarHelper* contém funções que geram todo o HTML necessário para construir as barras de ferramentas, podendo ser chamadas

quando desejado.

Depois de definir os itens da barra de ferramentas é necessário criar um arquivo com o código que representa as funcionalidades que irá chamar a barra de ferramenta solicitada, mostrado no quadro 4.

```
<?php                //toolbar.curso.php
  defined( '_JEXEC' ) or die( 'Acesso Restrito' );
  require_once( JApplicationHelper::getPath( 'toolbar_html' ));
  switch($task) {
    case 'edit':
    case 'add':
      TOOLBAR_curso::_NEW();
      break;
    default:
      TOOLBAR_curso::_DEFAULT();
      break;
  }
?>
```

Quadro 4: Estrutura de caso para exibir a barra de ferramentas.

A função *require_once* incluí e avalia o arquivo especificado durante a execução. É utilizado em casos que o mesmo arquivo pode acabar sendo incluído mais de uma vez durante a execução, quando na verdade ele só poderia ser incluído apenas uma. A função *getPath* da classe *JApplicationHelper* permite chamar o arquivo (no caso *toolbar.curso.html.php*) sem comprometimento com o nome do componente. Assim, mesmo com mudanças no nome dos arquivos, a saída será gerada sem modificações. Esse é um método de organização que difere bem a lógica do conteúdo a ser apresentado, facilitando também uma futura reutilização do arquivo.

Depois de requerer os arquivos de saída com auxílio da classe *JApplicationHelper*, é preciso definir a barra que deverá ser mostrada. A variável de requisição *$task* é automaticamente registrada para uso global pelo Joomla e usada para direcionar o fluxo lógico do componente, podendo ser a padrão, como mostra a figura 15, ou inserção, representado na figura 16.

Figura 15: Barra de Administração Padrão

Figura 16: Barra de Inserção

Como a função que irá funcionar ao clicar no botão *NOVO* ainda não está especificada, para ter acesso a barra de inserção é necessário acrescentar na URL do browser a seguinte extensão: *&task=add*.

Para adicionar os próprios ícones na barra de ferramentas, é necessário usar a função *custom()* que está disponível na classe *JToolBarHelper*. É necessário passar como parâmetro a tarefa a ser realizada pelo botão, o ícone ativo, o inativo e a descrição do mesmo.

7.1.4 Criando uma Tabela no Banco de Dados

As informações do componente são armazenadas no banco de dados relacionado ao *website*. No caso do componente de cursos, será necessário criar uma nova tabela para armazenar as informações do mesmo, que poderão ser inseridas, modificadas ou excluídas pelo usuário. Através do código SQL, crie uma tabela com o nome de *jos_curso*, como mostra o quadro 5. O prefixo *jos_* é usado para padronizar as tabelas que armazenam informações de componentes do Joomla.

```
CREATE TABLE `jos_curso` (
  `id` int(11) NOT NULL auto_increment,
  `nome_curso` varchar(255) NOT NULL,
  `professor` varchar(255) NOT NULL,
  `data` date NOT NULL,
  `vagas` int NOT NULL,
  `descricao` text NOT NULL,
  `published` tinyint(1) unsigned NOT NULL default '0',
  PRIMARY KEY (`id`)
);
```

Quadro 5: Criando a tabela dos cursos.

7.1.4.1 Criando a Classe da Tabela

É necessário a criação de funções para cada ação que o componente irá realizar, como de inserção, atualização e exclusão. No entanto, esse trabalho pode ser evitado no Joomla com o uso da classe *JTable*, que dispõe de

funções prontas para criar, ler, atualizar e deletar informações o banco de dados.

Para usufruir de tal benefício, é necessário criar uma classe com as informações da tabela que herde as funcionalidades da *JTable*, como mostra o quadro 6. Esse arquivo será do formato *php*, e estará localizado dentro do diretório *administrator/components/com_curso/tabelas*.

```
<?php       //tabelas.curso.php
  defined( '_JEXEC' ) or die( 'Acesso Restrito' );
  class TableCurso extends JTable {
     var $id = null;
     var $nome_curso = null;
     var $professor = null;
     var $data = null;
     var $vagas = null;
     var $descricao = null;
     var $published = null;
     function __construct(&$db) {
        parent::__construct( '#__curso', 'id', $db );
     }
  }
?>
```

Quadro 6: Classe da tabela de cursos.

Ao herdar os atributos da classe *JTable*, deve-se adicionar as colunas da tabela e atribuir seus valores como nulo. Posteriormente, é necessário sobrescrever o método *__construct* da classe *constructor* usando o nome da tabela (onde #__ é apenas um prefixo de interpretação), a chave primária, e o objeto do banco. Assim, a classe *TableCurso* possuirá métodos como *bind(), store(), load()* e *delete()*, permitindo um bom gerenciamento do banco com a inserção de um simples código SQL.

7.1.5 Criando um Formulário de Cadastro de Cursos

Após criar a classe que representará sua tabela, é preciso desenvolver uma interface que possa controlar os dados do banco, sempre separando a saída de informações da lógica. O documento que executará tal funcionalidade é o já criado *admin.curso.php*. No quadro 7 é mostrado a parte funcional da inserção de cursos, conhecida como parte lógica do componente. As informações a serem exibidas é mostrada no quadro 8, sempre descrita no arquivo *admin.curso.html.php*.

```
<?php     //admin.curso.php
   defined( '_JEXEC' ) or die( 'Acesso Restrito' );
   require_once( JApplicationHelper::getPath(  'admin_html' ));
   JTable::addIncludePath(JPATH_COMPONENT.DS.'tabelas');
   switch($task) {
      case 'add':
      editarCurso( $option );
         break;
   }
   function editarCurso( $option ) {
      $row =& JTable::getInstance('Curso', 'Table');
      $lists = array();
      $lists['published'] = JHTML::_('select.booleanlist', 'published', 'class="inputbox"',
            $row->published);
      HTML_Curso::editarCurso($row, $lists, $option);
   }
?>
```

Quadro 7: Parte funcional da inserção de cursos.

Como já visto anteriormente, a descrição da primeira linha é um padrão de segurança do Joomla. Na segunda linha é descrita a função *require_once(JApplicationHelper::getPath('admin_html'))* irá incluir o arquivo *admin.curso.html.php*. A função *getPath()* pega alguns dados (como *admin_html, front_html e class*) e retorna o caminho absoluto dos arquivos do componente correspondente. No entanto não é necessário especificar o nome do componente, podendo haver alterações e reaproveitamento de código.

O banco de dados ainda não está sendo usado, por isso deve-se incluir a classe da tabela já criada. A função que executará tal processo é descrita em *JTable::addIncludePath(JPATH_COMPONENT.DS.'tabelas')*. A função *addIncludePath* inclui todas as classes dos arquivos que estão no diretório *tabelas*. O Joomla usa o *JPATH_COMPONENT* para definir o caminho absoluto do diretório, sendo que a constante *DS* é um separador de diretório. O *switch()* verifica a variável *$task* que contém o nome da função desejada. Por fim a função *editarCurso()* prepara alguns elementos HTML antes de chamar a função *editarCurso* (da classe *HTML_curso*) que irá exibir as informações na sua tela. A matriz *$row* recebe os campos da tabela através da função *getInstance()*, da classe *JTable*. A matriz *$lists* é criada para receber as informações do HTML que será mostrado. A classe *HTML_curso* é definida no quadro 8, dentro do arquivo *admin.curso.html.php*.

```
<?php    defined( '_JEXEC' ) or die( 'Acesso Restrito' );
  class HTML_curso {
    function editarCurso($row, $lists, $option) {
      $editor =& JFactory::getEditor();  ?>
      <form action="index.php" method="post" name="adminForm" id="adminForm">
        <fieldset class="adminform">
          <legend>Formulário - Cadastro de Curso</legend>
          <table class="admintable">
          <tr> <td width="100" align="right" class="key">
               Curso:
            </td>
            <td>
              <input class="text_area" type="text" name="nome_curso" id="nome_curso"
                size="50" maxlength="250"
                value="<?php echo $row->nome_curso;?>" />
            </td>    </tr>
          <tr>   <td width="100" align="right" class="key">
               Professor:
            </td>
            <td>
              <input class="text_area" type="text" name="professor" id="professor"
                size="50" maxlength="250"   value="<?php echo $row->professor;?>" />
            </td>  </tr>
          <tr>  <td width="100" align="right" class="key">
                 Data:
            </td>
            <td>
                  <?php echo JHTML::calendar($row->data, 'data', 'data'); ?>
            </td>  </tr>
          <tr>   <td width="100" align="right" class="key">
               Vagas:
            </td>
            <td>
              <input class="text_area" type="text" name="vagas" id="vagas" size="2"
                maxlength="2"
                value="<?php echo $row->vagas;?>" />
            </td> </tr>
          <tr>  <td width="100" align="right" class="key">
               Descrição:
            </td>
           <td>
              <?php echo $editor->display( 'descricao',$row->descricao,'100%', '150', '40', '5' ) ; ?>
            </td>  </tr>
          <tr>   <td width="100" align="right" class="key">
               Público:
            </td>
            <td>
              <?php  echo $lists['published'];  ?>
            </td>  </tr>
          </table>
        </fieldset>
        <input type="hidden" name="id" value="<?php echo $row->id; ?>"/>
        <input type="hidden" name="option" value="<?php echo $option; ?>"/>
        <input type="hidden" name="task" value=""/>
      </form>   <?php
    } ?>
```

Quadro 8: Mostrar formulário de cadastro de curso.

A função *editarCurso()* tem como parâmetros os dados da tabela, uma matriz com os fragmentos HTML e o nome do componente. Assim, essa função é totalmente dedicada a informações que serão exibidas na tela, com diversos elementos da UI (*User Interface)*.

A primeira linha da função *$editor = & JFactory::getEditor(),* adiciona um objeto que representará os editores HTML. A classe *JFactory* possui outras funções como *getDate* e *getDocument*. A função *display()*, membro do objeto editor, retorna um HTML rico nas propriedades de edição de texto, e tem como parâmetros o nome da caixa de edição, a variável do banco e as dimensões. O restante do código é apenas um HTML básico, para gerar a saída de informações na tela.

Para verificar o resultado do formulário, basta digitar no navegador *http://seusite/administrator/index.php?option=com_curso&task=add*. O resultado é mostrado na figura 17.

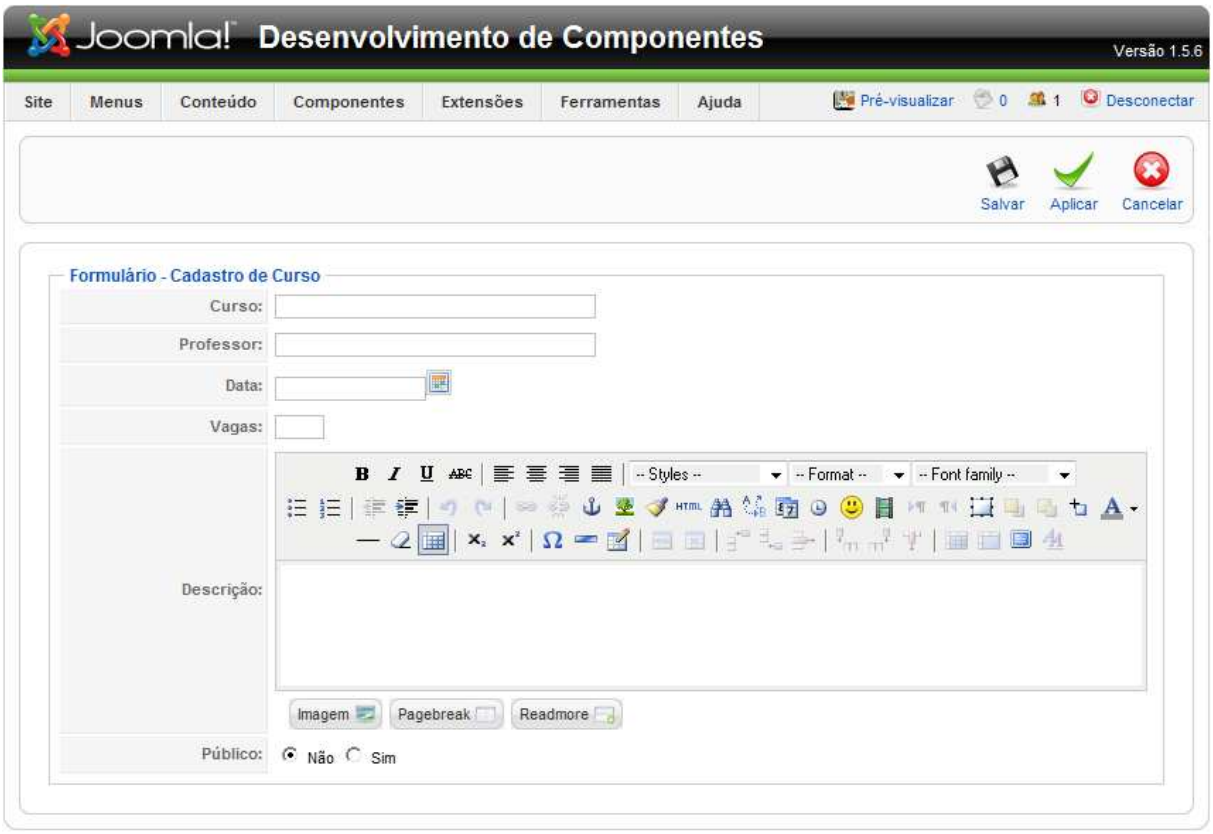

Figura 17: Formulário de Cadastro de Cursos.

7.1.6 Armazenando as Informações do Formulário

Após criar um formulário de edição, é necessário armazenar as informações contidas nele no banco de dados. A função receberá o nome de *salvarCurso()*, e será criada dentro do arquivo *admin.curso.php*, mostrada no quadro 9.

```
function salvarCurso( $option) {                    //admin.curso.php
   global $mainframe;
   $row = & JTable::getInstance('curso', 'Table');
   if (!$row->bind(JRequest::get('post'))) {
      echo "<script> alert('".$row->getError()."'); window.history.go(-1); </script>\n";
      exit();
   }
   $row->descricao = JRequest::getVar( 'descricao', '', 'post', 'string',
      JREQUEST_ALLOWRAW );
   if (!$row->store()) {
      echo "<script> alert('".$row->getError()."'); window.history.go(-1); </script>\n";
      exit();
   }
   $mainframe->redirect('index.php?option=' . $option, 'Dados Adicionados com
      Sucesso');
}
```

Quadro 9: Função que salva os dados do formulário.

Na primeira linha da função, está a variável global *$mainframe*, possuindo vários métodos que podem ser usados para controlar as variáveis e cabeçalhos de endereço. Em seguida é adicionado o *$row* como instância da classe *tableCurso*. Em seguida, é chamado a função *bind()* do *$row* que irá permitir que as variáveis possam ser manipuladas diretamente. Essa função é associada com todos os elementos das variáveis do objeto. A função *JRequest:get()* é usada para reduzir o risco de injeções SQL na manipulação dos dados. Caso a função falhe por algum motivo, será mostrado na tela um alerta *JavasScript*, retornando a tela anterior. Após a associação, a manipulação das variáveis da *$row* pode ser realizada diretamente. O campo *descrição* necessita de um manuseio especial para atribuir os valores de saída. Deve ser usado a função *getVar()* da classe *JRequest*, passando como parâmetros o nome da variável, o valor padrão o tipo de requisição e o formato.

A função *store()* é chamada para concluir a inserção ou atualização dos dados, que por sua vez vai ser definida pelo valor do *id*, que quando esta sendo criado, não apresenta valor. Caso a função acima seja concluída com sucesso, sem entrar na condição de erro, a função *redirect()* do *$mainframe* é

acionada para redirecionar o endereço da página junto com uma mensagem confirmando o sucesso da operação.

Para poder executar a função que irá salvar as informações do formulário, é necessário adicionar o código em negrito mostrado no quadro 10 no arquivo *admin.curso.php*. O resultado é mostrado na figura 18.

```
switch($task) {                 //admin.curso.php
      case 'add':
        editarCurso( $option );
        break;
      case 'save':
        salvarCurso( $option);
        break;
```

Quadro 10: Adicionado a opção de salvar na estrutura de caso.

Figura 18: Mensagem de dados adicionados com sucesso.

7.1.7 Listando os Dados da Tabela

Agora com a possibilidade de inserção de novos cursos no banco de dados, deve ser criada uma função que exiba esses dados, para que possam ser manipulados de acordo com a necessidade do desenvolvedor. O código que representa tal funcionalidade é mostrado no quadro 11, descrito dentro do arquivo responsável pela lógica administrativa *admin.curso.php.*

```
function mostrarCurso( $option ) {   //admin.curso.php
   $db =& JFactory::getDBO();
   $query = "SELECT * FROM #__curso";
   $db->setQuery( $query );
   $rows = $db->loadObjectList();
   if ($db->getErrorNum()) {
      echo  $db->stderr();
      return false;
   }
   HTML_curso::mostrarCurso( $option, $rows );
}
```

Quadro 11: Mostrar cursos cadastrados.

A função carrega os dados a serem mostrados. A variável *$db* recebe os atributos de manuseio de informações do banco de dados através da função *getDBO()*, que pertence a classe *JFactory*. A segunda linha atribui uma consulta SQL para a variável *$query*. A função *setQuery()* pega e armazena os valores da tabela para serem usados mais tarde. Quando é chamada a função *loadObjectList()*, os valores são carregados em matrizes, que por sua vez é usado como parâmetro da função *mostrarCurso()* que ficará no arquivo *admin.curso.html.php,* gerando a tela que contém as informações do banco de dados, como mostra o quadro 12.

```
function mostrarCurso( $option, &$rows ) {  ?>        //admin.curso.html.php
        <form action="index.php" method="post" name="adminForm">
        <table class="adminlist">
          <thead>
            <tr>
                 <th width="20">
                <input type="checkbox" name="toggle" value="" onclick="checkAll(<?php
                  echo count( $rows ); ?>);"/>
              </th>
              <th class="title">Curso</th>
              <th width="50%">Professor</th>
              <th width="10%">Data</th>
              <th width="5%">Vagas</th>
              <th width="5%" nowrap="nowrap">Publicar</th>
            </tr>
          </thead>  <?php
          $k = 0;
          for ($i=0, $n=count( $rows ); $i < $n; $i++) {
            $row = &$rows[$i];
            $checked = JHTML::_('grid.id', $i, $row->id );
            $published = JHTML::_('grid.published', $row, $i );              ?>
            <tr class="<?php echo "row$k"; ?>">
              <td>
                <?php echo $checked; ?>
              </td>
              <td>
                <?php echo $row->nome_curso; ?>
              </td>
              <td>
                 <?php echo $row->professor; ?>
              </td>
              <td>
                 <?php echo $row->data; ?>
              </td>
              <td>
                 <?php echo $row->vagas; ?>
              </td>
              <td align="center">
                 <?php echo $published;?>
               </td>
            </tr>
            <?php
            $k = 1 - $k;
          }           ?>
        </table>
        <input type="hidden" name="option" value="<?php echo $option;?>" />
        <input type="hidden" name="task" value="" />
        <input type="hidden" name="boxchecked" value="0" />
        </form>
        <?php
    }
```

Quadro 12: Exibe cursos cadastrados.

Essa função começa pela definição de um formulário que chama o *index.php*, com o nome de *adminForm*. Em seguida é adicionado uma tabela com características da classe *adminlist*. Todos os itens são comuns, exceto o primeiro que tem a ação de selecionar todos os registros pela caixa de seleção. Após apresentar as informações da tabela na tela, ela deve ser preenchida com os dados do banco. É criado uma estrutura de repetição com o comando *for* que irá ser executado exatamente a quantidade de registros no banco. Para preencher graficamente as informações da caixa de seleção e do estado de publicação do componente, é utilizado uma função da classe *JHTML* que irá colocar a imagem de acordo com o registro do mesmo. Por exemplo, caso a variável *published* esteja com valor “0”, será atribuído uma imagem de negação, como um “X”. Caso seja “1”, uma imagem de valor positivo é adicionada, no caso algo como “V”.

Depois do formulário, tem-se três variáveis escondidas. A primeira possui o valor do nome do componente em execução, a segunda com o valor da ultima tarefa realizada, e a ultima com o valor da quantidade de caixas de seleção marcadas. O arquivo de escolha deve ser atualizado dentro da estrutura *swith*, de acordo com o quadro 13. O resultado da função que irá mostrar os cursos cadastrados é mostrado na figura 19.

```
. . .                              //admin.curso.php
    case 'save':
      salvarCurso( $option );
      break;
    default:
      MostrarCurso( $option);
      break;
```

Quadro 13: Inserção da função mostrar curso na estrutura de caso.

Figura 19: Listando os Cursos Cadastrados

7.1.8 Editando os Dados da Tabela

Para executar a edição de um registro que já esta salvo no banco, é preciso acrescentar algumas linhas de código na função *editarCurso()* e na estrutura de caso do arquivo *admin.curso.php*, como mostra os quadros 14 e 15 respectivamente.

```
function editarCurso( $option ) {     //admin.curso.php
    $row =& JTable::getInstance('Curso', 'Table');
    $cid = JRequest::getVar( 'cid', array(0), '', 'array' );
    $id = $cid[0];
    $row->load($id);
```

Quadro 14: Função para editar curso.

```
. . .                              //admin.curso.php
    case 'edit':
    case 'add':
      editarCurso( $option );
      break;
```

Quadro 15: Inserção do editar na estrutura de caso.

Para poder editar os cursos dinamicamente, é necessário acrescentar algumas linhas de código na função *mostrarCurso()* da classe

HTML_curso, como é mostrado no quadro 16.

```
. . .                              //admin.curso.php
jimport('joomla.filter.output');
$k = 0;
for ($i=0, $n=count( $rows ); $i < $n; $i++) {
$row = &$rows[$i];
$checked = JHTML::_('grid.id', $i, $row->id );
$published = JHTML::_('grid.published', $row, $i );
$link = JFilterOutput::ampReplace( 'index.php?option=' . $option .
'&task=edit&cid[]='. $row->id );
?>
<tr class="<?php echo "row$k"; ?>">
  <td>
    <?php echo $checked; ?>
  </td>
  <td>
    <a href="<?php echo $link; ?>">
    <?php echo $row->nome_curso; ?></a>
  </td>
```

Quadro 16: Editar cursos dinamicamente.

Para que a atualização possa ocorrer deve-se usar a função *ampReplace()* da classe *JFilterOutput*. Para ter acesso a classe é preciso importá-la de uma biblioteca com a chamada de *jimport('joomla.filter.output')*. Portando a função *jimport()* serve para adicionar algum código quando necessário.

Agora é necessário adicionar a funcionalidade do botão *APLICAR* da barra de ferramentas. Dentro do arquivo *admin.curso.php* deve ser adicionado algumas informações, como é mostrado no quadro 17. Primeiramente, deve-se acrescentar como parâmetro da função *savarCurso()* a variável *$task*, que armazena a tarefa requisitada pelo usuário. Depois é preciso acrescentar uma estrutura de caso para que a resposta do usuário possa receber um retorno de acordo com a tarefa executada.

```
…                                          //admin.curso.php
    case 'apply':
    case 'save':
      salvarCurso( $option, $task );        break;
…
  Function salvarCurso ($option, $task) {
...
    switch ($task) {
      case 'apply':
        $msg = 'Dados Aplicados com Sucesso!';
        $link = 'index.php?option=' . $option . '&task=edit&cid[]='. $row->id;
        break;
      case 'save':
        default:
          $msg = 'Dados Adicionados com Sucesso!';
          $link = 'index.php?option=' . $option;
          break;      }
    $mainframe->redirect($link, $msg);
```

Quadro 17: Adequando a estrutura de caso de acordo com a solicitação.

7.1.9 Excluindo os Registros

Inicialmente é preciso criar a chamada da função de excluir registros na estrutura de caso do arquivo *admin.curso.php*, como é mostrado no quadro 18.

```
    case 'remove':    //admin.curso.php
      removerCurso( $option );
      break;
```

Quadro 18: Opção de remover na estrutura de caso.

E depois criar a função *removerCurso()* dentro do mesmo arquivo, representado no quadro 19.

```
  function removerCurso( $option ) {      //admin.curso.php
    global $mainframe;
    $cid = JRequest::getVar( 'cid', array(), '', 'array' );
    $db =& JFactory::getDBO();
    if(count($cid)) {
      JArrayHelper::toInteger($cid);
      $cids = implode( ',', $cid );
      $query = "DELETE FROM #__curso WHERE id IN ( $cids )";
      $db->setQuery( $query );
      if (!$db->query()) {
        echo "<script> alert('".$db->getErrorMsg()."'); window.history.go(-1); </script>\n";
      }
    }  $mainframe->redirect( 'index.php?option=' . $option );
  }
```

Quadro 19: Função que remove o curso.

A variável *$cid* contem as caixas de marcação selecionadas. Deve-se extrair a variável *$cid* do formulário para ver se há algum *id* na matriz. Se existir, a estrutura de repetição entra em funcionamento até rodar todos esses *ids*. Qualquer erro que ocorrer o Joomla redirecionará para a tela anterior.

7.2 Desenvolvimento do *Front-end*

No *front-end* os cursos cadastrados na área administrativa serão exibidos para os usuários. Um *link* para cada curso irá mostrar os detalhes do mesmo, informando uma breve descrição, a quantidade de vagas, o professor que irá ministrar e a data. Caso o usuário tenha interesse em participar do curso deve se cadastrar informando o número do CPF, que será validado no cadastro geral da semana de informática e verificado sua autenticidade.

As funcionalidades do componente que são exibidas apenas para os usuários são armazenadas no diretório *components/com_curso.* O arquivo a ser inicialmente executado é o *curso.php*, tendo a mesma lógica de acesso da área administrativa.

Todas as requisições do *front-end* são realizadas a partir do diretório *index.php*. O *link* de acesso ao componente estará disponível no menu principal, pois no desenvolvimento das funções do *back-end* tal procedimento já foi realizado. O endereço de acesso do componente utilizando diretamente a URL é *http:/.../seusite/index.php?**option=com_curso***.

As funções usadas serão brevemente explanadas devido as definições já descritas na seção anterior.

7.2.1 Possibilidades das Tarefas

O arquivo de acesso inicial do componente deve conter uma estrutura de caso que encaminhe a ultima opção de tarefa solicitada, como é mostrado no quadro 20.

```
<?php       //curso.php
        defined( '_JEXEC' ) or die( 'Restricted access' );
        echo '<div class="componentheading">Cursos</div>';
        jimport('joomla.application.helper');
        require_once( JApplicationHelper::getPath( 'html' ) );
        JTable::addIncludePath(JPATH_ADMINISTRATOR.DS.
                'components'.DS.$option.DS.'tabelas');
        switch($task){
                case 'ver':
                        verCurso($option);
                        break;
                case 'matricular':
                        adicionarMatricula($option);
                        break;
                case 'cadastrar':
                        adicionarCadastro($option);
                        break;
                case 'salvar':
                        salvarCadastro($option);
                        break;
                default:
                        mostrarCursosPublicados($option);
                        break;
        }
?>
```

Quadro 20: Estrutura de caso da área pública.

Após a função de segurança que verifica o acesso ao *framework*, é exibido um título com o padrão de componente através da classe *componentheading*.

A função *jimport* está importando a biblioteca *JApplicationHelper* que auxilia a inclusão do arquivo responsável pela saída de informações, usado na função *php require_once()*. A função *addInlcudePath* irá pegar as definições de classe da tabela de acordo com o diretório indicado.

O *switch* verifica o valor armazenado na variável *$task* e encaminha para a função correspondente. As opções são de ver descrição do curso selecionado, cadastrar no curso, cadastrar na semana de informática, salvar os registros inseridos e a função padrão que irá exibir a lista de cursos.

7.2.2 Criando as Tabelas do *Front-end*

Nos quadros 21 e 22 são descritas as classes das tabelas.

```
<?php                    //tabelas/cadastro.php
defined('_JEXEC') or die('Restricted access');
class TableCadastro extends JTable{
        var $id = null;
        var $nome = null;
        var $cpf = null;
        var $email = null;
        function __construct(&$db)   {
                parent::__construct( '#__curso_cadastro', 'id', $db );
        }
}
?>
```

Quadro 21: Classe da tabela cadastro.

```
<?php                //tabelas/matricula.php
defined('_JEXEC') or die('Restricted access');
class TableMatricula extends JTable{
        var $id = null;
        var $cpf = null;
        var $id_curso = null;
        var $data_matricula = null;
        function __construct(&$db)   {
                parent::__construct( '#__curso_matricula', 'id', $db );
        }
}
?>
```

Quadro 22: Classe da tabela matrícula.

7.2.3 Listando os Cursos

A função que lista os cursos disponíveis é definido no arquivo *curso.php*, como é mostrado no quadro 23.

```
function mostrarCursosPublicados($option){    //curso.php
        $db =& JFactory::getDBO();
        $query = "SELECT * FROM #__curso WHERE published = '1' ORDER BY
                nome_curso ASC";
        $db->setQuery( $query );
        $rows = $db->loadObjectList();
        if ($db->getErrorNum()){
                echo $db->stderr();
                return false;
        }
        HTML_cursos::mostrarCursos($rows, $option);
        HTML_cursos::mostrarParticipar ($rows, $option);
}
```

Quadro 23: Função que lista os cursos publicados.

A função *getDBO()* da classe *JFactory* instancia um objeto da classe

JDataBase, atribuindo o uso da função *setQuery()*, que carrega os dados que serão mostrados para serem usados quando necessário. Quando a função *setQuery()* é chamada, é passado como parâmetro a variável que armazena a consulta a ser realizada. Ela retorna os valores que estão disponíveis para publicação, listando os cursos em ordem alfabética. Todos os dados são inseridos na variável *$rows* através da função *loadObjectList()*, que percorre todos os registros.

A função *mostrarCursos(),* localizada no arquivo *curso.html.php,* que é responsável por toda informação mostrada na tela, exibe para o usuário a lista de cursos disponíveis na semana de informática, descrita no quadro 24.

```
class HTML_cursos{   //curso.html.php
        function mostrarCursos($rows, $option){
                ?><table><?php
                foreach($rows as $row){
                        $link = JRoute::_('index.php?option=' . $option . '&id=' . $row->id .
                                '&task=ver');
                        echo '<tr><td><a href="' . $link . '">' . $row->nome_curso .
                                '</a></td></tr>';
                }
                ?></table><?php
        }
}
```

Quadro 24: Mostrar os cursos publicados.

A classe *HTML_cursos* é definida no arquivo *cursos.html.php*, e possui todas as funções que irão gerar saída de informações na tela do usuário. A primeira função definida na classe é a *mostrarCursos()*, que simplesmente exibe os nomes dos cursos armazenados. A cada nome de curso é atribuído um *link* que irá exibir os detalhes do mesmo através da utilização da classe *JRoute*, que irá administrar a página a ser exibida através de parâmetros como *id* e *task*.

```
<?php  //curso.html.php
function mostrarParticipar($rows, $option){ ?>
        <p> </p>
        <form action="<?php echo JRoute::_('index.php') ?>" method="post">
        <table>
        <tr>
                <td>
                        <strong>Participar da Semana de Informática</strong>
                </td>
        </tr>
        </table>
        <input type="hidden" name="id" value="<?php echo $row->id; ?>" />
        <input type="hidden" name="task" value="cadastrar" />
        <input type="hidden" name="option" value="<?php echo $option; ?>" />
        <input type="submit" class="button" id="button" value="Prosseguir >>" />
        </form>
        <?php
} ?>
```

Quadro 25: Exibir botão de cadastro.

A função *mostrarParticipar(),* também chamada dentro de *mostrarCursosPublicados(),* é definida na classe *HTML_cursos*, como mostra o quadro 25. Ela exibe um botão para iniciar o cadastro na semana e informática. O resultado do código descrito até o momento no *front-end* é mostrado na figura 20.

Figura 20: Lista de Cursos.

7.2.4 Cadastrar na Semana de Informática

O botão “Prosseguir” exibido na figura 20, atribui o valor “cadastrar” na variável *task*, direcionando na estrutura de caso para a função *adicionarCadastro()*, mostrada no quadro 26.

```
function adicionarCadastro($option){
        $row =&JTable::getInstance('Cadastro','Table');
        HTML_cursos::mostrarFormularioCadastro($row, $option);
}
```

Quadro 26: Função para iniciar o cadastro.

A função instancia a classe da tabela “cadastro” para a matriz *$row*, que é usada como parâmetro na função *mostrarFormularioCadastro()*, que por sua vez, está contido o código que irá gerar o formulário de cadastro a ser exibido, como é mostrado no quadro 27.

```
function mostrarFormularioCadastro($row, $option){  ?>
  <form action="index.php" method="post" name="adminForm" id="adminform">
  <fieldset class="adminForm">
  <legend>Cadastro no Evento</legend>
  <table class="adminTable">
     <tr> <td width="80" align="right" class="key"> <strong>Nome:</strong> </td> <td>
       <input class="text_area" type="text" name="aluno"
         id="aluno" size="11" maxlingth="250" value="<?php echo $row->aluno; ?>" />
     </td>
    </tr>
    <tr> <td width="80" align="right" class="key">  <strong>CPF:</strong>  </td>
     <td>
       <input class="text_area" type="text" name="cpf"
         id="cpf" size="11" maxlingth="250" value="<?php echo $row->cpf; ?>" />
     </td>
   </tr>
   <tr> <td width="80" align="right" class="key"> <strong>Email:</strong> </td>
     <td>
       <input class="text_area" type="text" name="email"
         id="email" size="40" maxlingth="250" value="<?php echo $row->email; ?>" />
     </td>
   </tr>
  </table>
  <input type="hidden" name="id" value="<?php echo $row->id; ?>" />
  <input type="hidden" name="task" value="salvar" />
  <input type="hidden" name="option" value="<?php echo $option; ?>" />
  <input name="button" type="submit" class="button" id="button" value="Cadastrar"/>
  </form>  <?php  }
```

Quadro 27: Formulário de Cadastro.

O formulário irá atribuir o valor “salvar” à variável “task” quando o botão “submit” for acionado. Isso acarretará em uma nova seleção na estrutura de caso, que chamará a função *salvarCadastro()*, descrita no quadro 28.

```
function salvarCadastro($option){
   global $mainframe;
   if(isset($_POST['button'])){
      $cpf_enviado = validaCPF($_POST['cpf']);
   }
   if ($cpf_enviado == true){
      $row =& JTable::getInstance('cadastro','Table');
      if (!$row->bind(JRequest::get('post'))) {
          echo "<script> alert('".$row->getError()."'); window.history.go(-1); </script>\n";
          exit();
      }
      $row->id = (int) $row->id;
      if (!$row->store()) {
          echo "<script> alert('".$row->getError()."'); window.history.go(-1); </script>\n";
          exit();
      }
      $mainframe->redirect('index.php?option='.$option, 'Cadastro Realizado');
   }else{
      $mainframe->redirect('index.php?option='.$option, 'CPF INVÁLIDO');
   }
}
```

Quadro 28: Função que salva o cadastro no banco.

A função *isset()* verifica se o botão relacionado ao *$_*POST do formulário foi acionado. Caso seja verdade, o valor do CPF digitado é passado para a função *validaCPF()* que validará sua autenticidade. Com a veracidade do CPF, a matriz *$row* é instanciada da classe *cadastro*, já definida acima. A função *bind()* irá atribuir os valores do formulário para a variável *$row*. A função *store()* finaliza o armazenamento dos registros. Na figura 21 é mostrado o formulário criado.

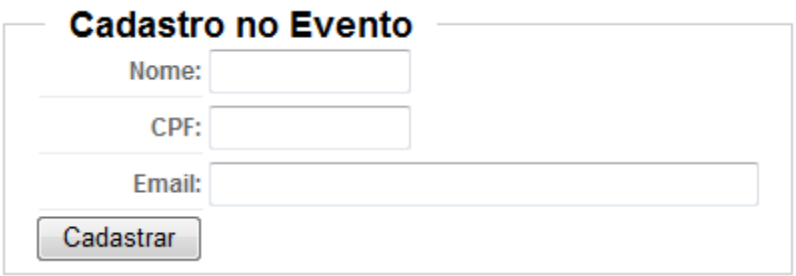

Figura 21: Formulário de Cadastro na Semana de Informática.

7.2.5 Exibir Detalhes do Curso

Ao clicar no curso que deseja ver os detalhes, é atribuído um valor (“ver”) a variável *task* que irá indicar qual função executar. Com a verificação da variável na estrutura de caso, a função executada será a *verCurso()*, que é definida no quadro 29.

```
function verCurso($option){
	$id = JRequest::getVar('id', 0);
	$row =& JTable::getInstance('curso', 'Table');
	$row->load($id);
	if(!$row->published){
		JError::raiseError( 404, JText::_( 'Id Inválido' ) );
	}
	HTML_cursos::mostrarCurso($row, $option);
	$contador_vagas = $row->vagas;
	$db =& JFactory::getDBO();
	$db->setQuery("SELECT * FROM #__curso_matricula WHERE id_curso = '$id'");
	$rows = $db->loadObjectList();
	$contador = 0;
	foreach($rows as $row){
		$contador = $contador+1;
	}
	HTML_cursos::mostrarMatricula($contador, $contador_vagas);
	HTML_cursos::mostrarFormularioMatricula($option, $id);
}
```

Quadro 29: Função lógica para mostrar o curso.

A função pega o valor do *id* do curso através do comando *$id = JRequest::getVat(‘id’,0)* e atribui os valores da classe da tabela para *$row*. A função *load()* carrega os dados publicados do *id* (curso) correspondente , que serão passados como paramento na função que vai exibir as informações na tela.

A função *mostrarCurso()*, mostrada no quadro 30, é responsável por imprimir na tela as informações do curso corrente. Em seguida é realizada uma consulta na tabela que armazena as matriculas para obter o restante de vagas.

```
function mostrarCurso($row, $option){  //curso.html.php
        ?>
        <p class="contentheading"><?php echo $row->nome_curso; ?></p>
        <p><strong>Professor:</strong> <?php echo $row->professor; ?></p>
        <p><strong>Total de Vagas:</strong> <?php echo $row->vagas; ?></p>
        <p class="createdate"><?php echo JHTML::Date($row->data_matricula); ?></p>
        <p><?php echo $row->descricao; ?></p>
        <?php $link = JRoute::_('index.php?option=' . $option) ; ?>
        <a href="<?php echo $link; ?>">&lt; retornar para a lista de cursos</a>
        <?php
}
```

Quadro 30: Função que exibe o curso escolhido.

A função *mostrarCurso()* apenas pega os valores contidos em *$row* e os exibe através de comandos básicos de HTML e PHP. A variável *$link* herda as propriedades da classe *JRoute* que auxilia na navegação, sendo exibida para retornar a lista de cursos.

7.2.6 Matricular nos Cursos

Ao detalhar o curso, o código do quadro 31 é chamado.

```
function mostrarFormularioMatricula($option, $id_curso){
  ?>
  <p> </p>
  <form action="<?php echo JRoute::_('index.php') ?>" method="post">
  <table>
          <tr>
     <td width="0" align="right" class="key">
       <strong>CPF:</strong>
     </td>
     <td>
       <input class="text_area" type="text" name="cpf"
         id="cpf" size="11" maxlingth="250" value="<?php echo $row->cpf; ?>" />
     </td>
   </tr>
  </table>
  <input type="hidden" name="id_curso" value="<?php echo $id_curso; ?>" />
  <input type="hidden" name="task" value="matricular" />
  <input type="hidden" name="option" value="<?php echo $option; ?>" />
  <input type="hidden" name="valor" value="<?php echo $row->cpf; ?>" />
  <input name="button" type="submit" class="button" id="button" value="Matricular" />
  </form>
  <?php
}
```

Quadro 31: Mostrar formulário de matrícula do curso.

A função descrita acima apenas exibe através de código HTML um formulário de matrícula. A variável *$task* recebe o valor “matricular” quando o botão “submit” é acionado.

A função *mostrarMatricula()* chamada ao exibir os detalhes do curso, conta a quantidade de inscritos e exibe o total de vagas remanescente como é mostrado no quadro 32.

```
function mostrarMatricula($contador, $contador_vagas)    {    //curso.html.php
        ?>
        <p> </p>
        <p><strong><?php echo 'Vagas Remanecentes:  ';?><?php echo
                $contador_vagas-$contador; ?></strong></p>
        <?php
}
```

Quadro 32: Mostrar vagas remanescentes.

A figura 22 representa o código descrito para exibir os detalhes do curso, juntamente com a quantidade de vagas disponíveis e o formulário de cadastro.

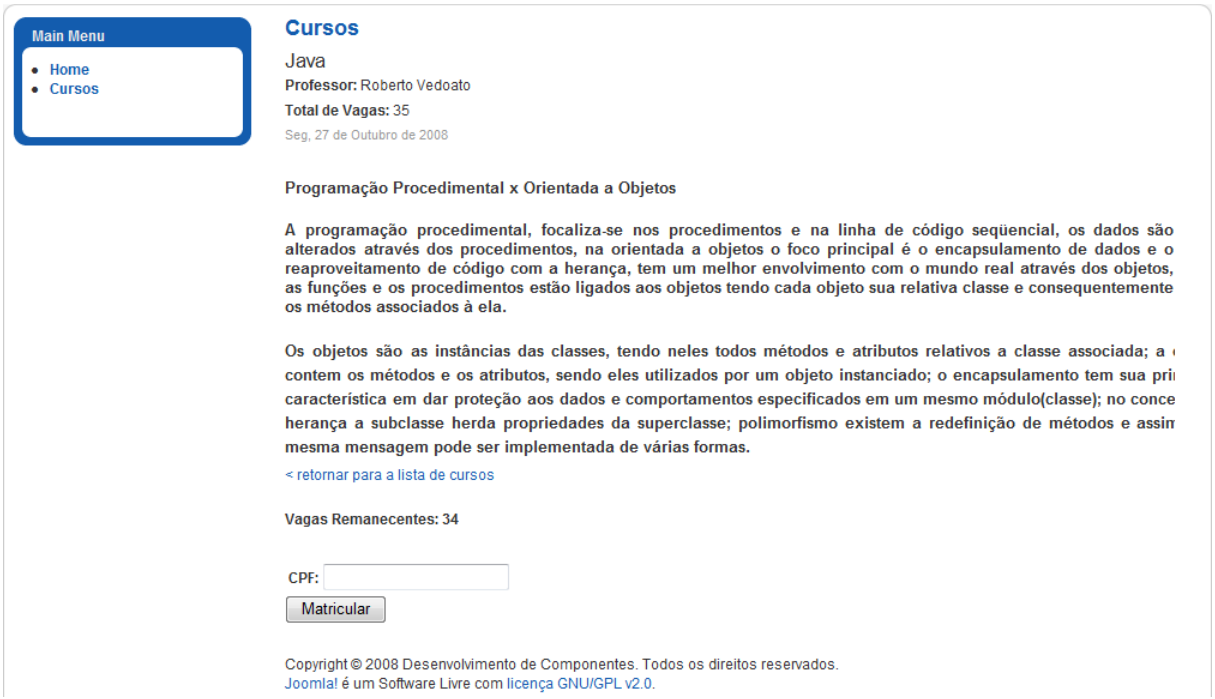

Figura 22: Detalhes Do Curso.

Caso haja interesse em participar do curso, o usuário deve entrar com o numero do CPF e confirmar a matrícula. Apenas os cadastrados na semana de informática poderão executar tal processo. As figuras 23 e 24 mostram a resposta

enviada de acordo com a validação da matrícula.

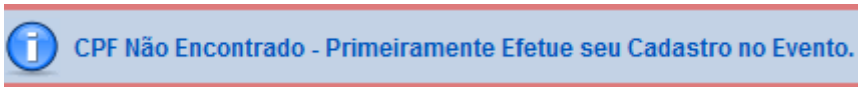

Figura 23: Mensagem de CPF não encontrado.

Figura 24: Mensagem de Matrícula Concluída.

Ao exibir o formulário, a variável de *task* recebe “matricular” como valor, mais uma vez determinado a função a ser executada na estrutura de caso montada. A chamada vai ser para a função *adicionarMatricula()*, representada no quadro 33.

```
function adicionarMatricula($option) {
    global $mainframe;
    jimport('joomla.utilities.date');
    $row =& JTable::getInstance('matricula', 'Table');
    if (!$row->bind(JRequest::get('post')))      {
            echo "<script> alert('".$row->getError()."'); window.history.go(-1); </script>\n";
            exit();
    }
    $ver=comparar($row->cpf);
    if ($ver==true) {
      $row->id = null;
      $row->id_curso = (int) $row->id_curso;
      $currDate = new JDate();
      $row->data_matricula = $currDate->toMySQL();
      if (!$row->store()) {
              echo "<script> alert('".$row->getError()."'); window.history.go(-1); </script>\n";
              exit();
      }
           $mainframe->redirect('index.php?option=' . $option . '&id=' . $row->id_curso .
           '&task=ver', 'Matrícula Concluída');
      }else{
          $mainframe->redirect('index.php?option=' . $option . '&id=' . $row->id_curso .
          '&task=ver', 'CPF Não Encontrado - Primeiramente Efetue seu Cadastro no
          Evento.');
      }
}
```

Quadro 33: Função que armazena o cadastro no banco de dados.

A função irá armazenar o cadastro de um determinado aluno no curso corrente. As funções utilizadas no código são básicas de armazenamento, e já descritas anteriormente.

7.3 Finalizando o Componente

Apesar de o componente estar funcionando, é preciso criar dois arquivos SQL para gerência do banco, dois arquivos PHP que irão informar sobre a conclusão de instalação e desinstalação, e um arquivo XML que armazenará todas as informações necessárias na instalação. Os arquivos de instalação e desinstalação são representados nos quadros 34 e 35 respectivamente.

```
<?php        //install.curso.php
defined( '_JEXEC' ) or die( 'Restricted access' );
function com_install(){
        ?>
        <div class="header">O componente Curso foi instalado com sucesso!</div>
        <p>
        Parabéns! O componente de administração de cursos foi instalado com sucesso.
        Seu acesso via Back-end está disponível na barra de Ferramenas.
        </p>
        <?php
} ?>
```

Quadro 34: Conclusão da instalação.

```
<?php     //uninstall.curso.php
defined( '_JEXEC' ) or die( 'Restricted access' );
function com_uninstall(){
        ?>
        <div class="header">O Componente Curso foi removido do seu sistema</div>
        <p>
        Desinstalação Concluída
        </p>
        <?php
}
?>
```

Quadro 35: Conclusão da desinstalação.

Caso a instalação ocorra sem a existência de erros, o arquivo *install.curso.php* é executado informando o sucesso de instalação. O mesmo ocorre na desinstalação do componente, que executa o arquivo *uninstall.curso.php*.

```
CREATE TABLE IF NOT EXISTS `#__curso` (         //install.mysql.sql
`id` int( 11 ) NOT NULL AUTO_INCREMENT ,
`nome_curso` varchar( 255 ) NOT NULL ,
`professor` varchar( 255 ) NOT NULL ,
`data` date NOT NULL ,
`descricao` text NOT NULL ,
`published` tinyint( 1 ) unsigned NOT NULL default '0',
PRIMARY KEY ( `id` ) );
CREATE TABLE IF NOT EXISTS `#__curso_matricula` (
  `id` int(11) NOT NULL auto_increment,
  `id_curso` int NOT NULL,
  `cpf` int NOT NULL,
  `data_matricula` date NOT NULL,
    PRIMARY KEY (`id`));
CREATE TABLE IF NOT EXISTS `#__curso_cadastro` (
  `id` int(11) NOT NULL auto_increment,
  `aluno` varchar(255) NOT NULL,
  `cpf` varchar(11) NOT NULL,
  `email` varchar(255) NOT NULL,
    PRIMARY KEY (`id`) );
```

Quadro 36: Criando as tabelas no banco.

```
DROP TABLE #__curso;              //uninstall.mysql.sql

DROP TABLE #__curso_matricula;

DROP TABLE #__curso_cadastro;
```

Quadro 37: Removendo as tabelas do banco.

Os arquivos do formato SQL mostrados nos quadros 36 e 37, controlam as tabelas do banco de dados que devem ser usadas pelo componente, criando-as quando o mesmo é instalado, e deletando-as quando removido.

O principal arquivo de instalação é do formato XML, pois ele descreve todas as informações do componente, desde a disposição dos arquivos, até os direitos autorais e versões do mesmo. O quadro 38 descreve seu conteúdo.

```
<?xml version="1.0" encoding="UTF-8"?>      //curso.xml
<install type="component" version="1.5.0">
        <name>Curso</name>
        <author>Daniel Avanzi</author>
        <creationDate>Outubro 2008</creationDate>
        <copyright>(C) 2008</copyright>
        <authorEmail>avanzi@uol.com.br</authorEmail>
        <authorUrl>avanzi.sites.uol.com.br</authorUrl>
        <version>1.5.0</version>
        <license>Comercial/Estudo</license>
        <description>Componente de Estudo desenvolvido para ser aplicado na
        FALM</description>
        <installfile>install.curso.php</installfile>
        <uninstallfile>uninstall.curso.php</uninstallfile>
        <install>
                <sql>
                        <file driver="mysql" charset="utf8">install.mysql.sql</file>
                </sql>
        </install>
        <uninstall>
                <sql>
                        <file driver="mysql" charset="utf8">uninstall.mysql.sql</file>
                </sql>
        </uninstall>
        <files>
                <filename>curso.html.php</filename>
                <filename>curso.php</filename>
        </files>
        <administration>
                <menu>Curso</menu>
                <files folder="admin">
                        <filename>install.mysql.sql</filename>
                        <filename>uninstall.mysql.sql</filename>
                        <filename>admin.curso.html.php</filename>
                        <filename>admin.curso.php</filename>
                        <filename>tabelas/curso.php</filename>
                        <filename>tabelas/cadastro.php</filename>
                        <filename>tabelas/matricula.php</filename>
                        <filename>toolbar.curso.html.php</filename>
                        <filename>toolbar.curso.php</filename>
                </files>
        </administration>
</install>
```

Quadro 38: Arquivo XML.

A primeira definição do arquivo é o tipo e versão da extensão que vai ser instalada no Joomla. Depois vêm algumas descrições do autor e da ferramenta, seguido da indicação da disposição dos arquivos necessários.

Por fim, temos de organizar esses arquivos dentro de uma pasta compactada do formato *.zip*, como mostra a figura 25.

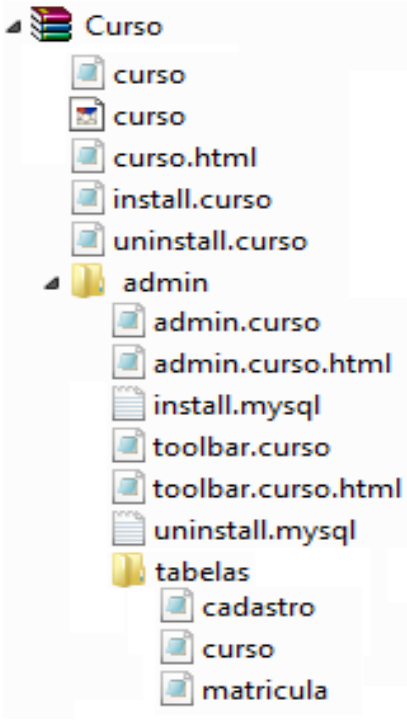

Figura 25: Arquivo de Instalação do Componente Curso.

O componente está pronto para ser instalado e usado em qualquer *framework* Joomla de versão igual ou superior a 1.5.

8 CONSIDERAÇÕES FINAIS

As exigências de tempo e complexidade pelos clientes *Web* contribuem para que novas técnicas de desenvolvimento sejam criadas. Um novo formato para atender a demanda desse mercado, são os sistemas gerenciadores de conteúdo. Eles podem acelerar a produção de forma considerada, além de possuírem uma estrutura pronta para o gerenciamento de conteúdo dinâmico. Os CMS de código aberto são tão completos quanto os proprietários, sendo merecedor de destaque o eficiente gerenciador Joomla.

As funcionalidades do Joomla são possíveis através do uso de componentes, que devem ser muito bem estruturados para que possam ser totalmente aproveitados.

Há uma ampla gama de componentes disponíveis para esse CMS, o grande problema é que o Joomla nem sempre pode contar com a existência de um componente que supra as exigências do cliente, sendo necessário neste caso, a criação dessa funcionalidade específica.

Devido à ausência de procedimentos descritos para realizar tal tarefa, tem-se a idéia de que um material básico do assunto pode ser de grande importância para dar início aos estudos de desenvolvimento de componentes para o Joomla. A aplicação prática descrita pode servir de base para o desenvolvimento futuro de funções mais complexas, pois, apesar de executar simples tarefas, adota técnicas da engenharia do Joomla pouco documentadas.

O resultado atingido pelo trabalho pode ser considerado como um passo inicial para o desenvolvimento de um componente de maior complexidade, contribuindo consideravelmente com a comunidade Joomla do Brasil.

A faculdade para a qual o componente foi desenvolvido pode também ser beneficiada com uma continuação do estudo apresentado, aperfeiçoando as definições já discutidas e aplicando em projetos reais.

REFERÊNCIAS

ALMEIDA, E. D.; LUCRÉDIO, D.; BIANCHINI, C. P.; PRADO, A. F.; TREVELIN, L. C.: **Ferramenta MVCase – Uma Ferramenta Integradora de Tecnologias para o Desenvolvimento de Componentes Distribuídos.** 2002. Disponível: <http://www.lbd.dcc.ufmg.br:8080/colecoes/sbes/2002/035.pdf>. Acesso: 12/05/08.

BOSCH.: **Information and Software Technology.** 1999. p. 257-273.

BROWN, A. W.; WALLNAU, K. C.:**Component-Based Software Engineering.** 1996. IEEE Computer Society Press. p. 7-15.

COSTA, Geciel R.: **Dicas de Joomla.** 2007. Disponível: <http://www.joomla.com.br/index2.php?option=com_content&do_pdf=1&id=185>. Acesso: 02/05/08.

COUTINHO, Júlio: **Construindo um Website Profissional e Dinâmico com o Joomla!**. Disponível: <http://www.joomlabrasilia.com.br/ead/virtual/>. Acesso: 21/10/08.

DAVID, Marcio F.: **Programação Orientada a Objetos: uma Introdução.** 2007. Disponível: <http://www.guiadohardware.net/artigo/programacao-orientada-objetos/>. Acesso: 15/05/08.

DUARTE, Ricardo. O.; LANNA, André L. P. M.: **Introdução ao JOOMLA no Projeto de Portais WEB.** UFOP – Universidade Federal de Ouro Preto. 2007. Disponível: <http://www.cirosantos.com/site/php/minicurso%20Joomla.pdf>. Acesso: 02/05/08.

GAMELEIRA, Fábio: **Joomla CMS** – Visão Geral. 2006. Disponível: <http://www.joomla.com.br/component/option,com_docman/Itemid,63/task,doc_details/gid,2>. Acesso: 05/05/08.

LEBLANC, Joseph: **Learning Joomla! 1.5 Extension Development:** Creating Modules, Components and Plug-ins with PHP. Packt Publishing Ltda, 2007. 161 p.

LEMOS, T. L. C.; OLIVEIRA, F. I.; NETO, J. P. S.; GOMES, J. T.: **Sistemas de Gerenciamento de Conteúdo Baseado em Software Livre.** 2008. Disponível: <http://www.guiadohardware.net/dicas/sgc-livre.html>. Acesso: 14/05/08.

MENDES, Matheus: **Programando Componentes para o Joomla.** 2005. Disnponível: <http://www.joomla.com.br/index2.php?option=com_content&do-pdf=1&id=96>. Acesso: 13/05/08.

MILLARCH, Francisco: **O Que é CMS e Porque Você Precisa de Um.** 2005. Disponível: <http://webinsider.uol.com.br/index.php/2005/06/08/o-que-e-cms-e-porque-voce-precisa-de-um/>. Acesso: 02/05/08.

MYSQLBRASIL: **A Empresa.** 2008. Disponível: <http://www.mysqlbrasil.com.br/>. Acesso: 05/05/08.

NIEDERAUER, Juliano: **PHP 5:** Guia de Consulta Rápida. 2ª edição. São Paulo: Novatec Editora Ltda, 2005. 144 p.

OTSUKA, Joice L.: **Fatores Determinantes na Efetividade de Ferramentas de Comunicação Mediada por Computador no Ensino à Distância.** 1997. Disponível: <http://penta.ufrgs.br/pesquisa/joice/joice_ti.html#sumula>. Acesso: 01/05/08.

PFAFFENSELLER, Moisés; PFAFFENSELLER, Matheus; KROTH, Eduardo: **Ferramenta de Apoio ao Desenvolvimento de Software Baseado em Componentes.** Disponível: <http://www.inf.ufrgs.br/~kroth/papers/fanfeseller-CBCOMP.pdf>. Acesso: 11/05/08.

ROLIM, C. R. A.; ARAÚJO, E. J. B.; TOZZI, J. C.: **Construindo Aplicações Web Dinâmicas e Otimizadas Utilizando AJAX – O Caminho da Web 2.0.** 2006. Disponível: <www.unibratec.com.br/jornadacientifica/diretorio/NOVOCON.pdf>. Acesso: 01/05/08.

SANTOS, Gildenir C.: **Criação de Páginas na Internet.** 2001. Disponível: <http://143.106.58.55/revista/include/getdoc.php?id=645&**art**icle=286&mode=pdf>. Acesso: 14/05/08.

SANTOS, Rodolfo C.: **Estudo Comparativo de Frameworks de desenvolvimento Web com Java e Aplicação a um Estudo de Caso.** Universidade Estadual do Norte do Paraná. 2007. 36 p.

SILVA, Ricardo P.: **Suporte ao Desenvolvimento e Uso de Frameworks e Componentes.** 2000. Disponível: <http://www.inf.ufsc.br/ricardo/download/tese.pdf>. Acesso: 12/05/08.

SUYAMA, M.: **História da Internet.** Disponível: <http://www.criarweb.com/internet>. Acesso: 01/05/08.

ZAKAS, N. C.; McPEAK, J.; FAWCETT, J.: **Professional AJAX.** Indianapolis. 2006.