Desenvolvimento de módulos para o Drupal 6

Everson Santos Araujo

Lavras
Minas Gerais - Brasil
2009

Everson Santos Araujo

Desenvolvimento de módulos para o Drupal 6

Monografia de Pós Graduação “Lato Sensu” apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, para obtenção do título de Especialista em “Administração em Redes Linux”.

Orientador:
Prof. Msc. Joaquim Quinteiro Uchôa

Lavras
Minas Gerais - Brasil
2009

Everson Santos Araujo

Desenvolvimento de módulos para o Drupal 6

Monografia de Pós-Graduação “Lato Sensu” apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, para obtenção do título de Especialista em “Administração em Redes Linux”.

Aprovada em 19 de Abril de 2009

_____

Prof. Msc. Kátia Cilene Amaral Uchôa

_____

Prof. Msc. Herlon Ayres Camargo

_____

Prof. Msc. Joaquim Quinteiro Uchôa
(orientador)

Lavras
Minas Gerais - Brasil
2009

Este trabalho é dedicado ao espirito colaborativo que permite a continuidade da pesquisa e o aumento do conhecimento humano.

Agradecimentos

Agradeço primeiramente a Deus, responsável pela criação primeira.

Agradeço à minha família em nome de meus pais, Helio e Valmira, e minha irmã, Evellyn, que sempre me proporcionaram um ambiente propicio ao aprendizado, sempre com carinho, atenção e incentivo.

Agradeço à minha esposa Vanusa pela paciência e noites de sono perdidos, compreensão e ajuda nas mais variadas tarefas para permitir minha concentração no trabalho.

Agradeço a meu filho, Heitor. Nova razão de minha vida e felicidade, sempre carinhoso e amável, uma grande motivação para seguir em frente e querer sempre o melhor, para oferecer-lhe o melhor.

Agradeço à UFLA e todos os professores que participaram direta ou indiretamente do curso, oferecendo seu tempo e conhecimento para que pudesse chegar ao final dessa jornada. Em especial ao professor Joaquim, por ter me orientado e forçado sempre que necessário para que o trabalho fosse bem elaborado, a professora Kátia por ter me apoiado no início do curso e ser componente da banca e ao professor Herlon pela honra de compor a banca e avaliar o trabalho.

E por fim a todos os companheiros de jornada, os alunos que participaram dos estudos em grupo e das aulas virtuais, equipe de suporte do curso e da instituição que sempre atenderam bem as requisições e todos os demais que possam não ter sido citados mas que também foram importantes.

Sumário

Lista de Figuras

Lista de tabelas

Resumo

Este trabalho apresenta o gerenciador de conteúdo Drupal com foco em desenvolvimento de módulos. Demonstra-se o padrão de código adotado e os recursos disponibilizados pela biblioteca do Drupal. Por fim é demonstrado a utilização destas informações no desenvolvimento do módulo de publicação de periódicos ejournal.

Palavras-chave: Drupal, módulos, php, desenvolvimento, gerenciador de conteúdo, ejournal.

1 Introdução

A necessidade de organização de informação é recorrente a todos os sistema computacionais e, com o crescente uso da *web*, torna-se indispensável a utilização de sistemas de gerenciamento de informação que permitam acesso através da mesma.

O Sistema de Gerenciamento de Conteúdo Drupal vem suprir essa necessidade, ao oferecer funções para gerenciar informação através de um ambiente *web* altamente flexível. Este ambiente permite sua utilização mesmo em situações que não são cobertas por seu sistema básico. Uma vez que permite a criação de módulos, que irão comunicar-se com o núcleo do sistema, a fim de fornecer funcionalidades adicionais para gerenciar a informação.

O Drupal oferece em seu conjunto de funcionalidades padrão uma vasta coleção de recursos, mesmo assim, não é capaz de se adaptar às necessidades específicas de todos os sistemas de gerenciamento de informação. Para que essa adaptação completa seja possível o Drupal oferece um ambiente rico para utilização de códigos adicionais através de módulos. Estes módulos são capazes de refinar cada um de seus recursos básicos podendo então oferecer um ambiente que se adapta a qualquer caso de uso.

Apesar de existir documentação oficial a respeito da criação de módulos, esta documentação não é muito bem organizada tendo partes publicadas em vários locais diferentes, não oferece uma gama de exemplos práticos e encontra-se disponível apenas em inglês.

O objetivo geral desta monografia é apresentar técnicas e normas de programação utilizadas na criação de módulos para o gerenciador de conteúdo Drupal em sua versão 6. Demonstrando a empregabilidade destas técnicas na criação de um ambiente de publicação científica.

O Capitulo 2 apresenta o Drupal e sua comunidade, bem como um histórico do surgimento do mesmo e um resumo dos recursos oferecidos pelo sistema.

O Capitulo 3 descreve os recursos de programação utilizados no Drupal, de forma a reconhecer o que pode ser conseguido através de seu sistema básico.

O Capítulo 4 apresenta informações específicas para o desenvolvimento de módulos de Drupal, indicando como se inicia um projeto, qual sua organização e formato.

O Capítulo 5 refere-se ao ambiente de publicação ejournal, um módulo de Drupal que será utilizado para apresentar a empregabilidade dos recursos apresentados.

E por fim o Capítulo 6 apresenta as conclusões finais do trabalho e aponta para trabalhos futuros.

2 Drupal

O Drupal surgiu em 2001 a partir da liberação do código de um sistema de notícias criado por Dries Buytaert, que tinha o intuito inicial de ser utilizado entre seus companheiros de faculdade para troca de informações. Segundo Santos (2008, p. 18) este sistema foi disponibilizado na internet sob a Licença Pública Geral GNU (GPL), após Dries ter saído da graduação, com o nome de Drop.org. E foi disponibilizado por ainda haver interesse de comunicação entre os colegas da faculdade.

Com o tempo a página Drop.org mudou de audiência criando um ambiente para conversas sobre novas tecnologias para internet. Com isso se tornou também um ambiente de experimentação para estas tecnologias, tendo sido aos poucos estendido para oferecer acesso a cada uma das novas idéias que surgiam das discussões de seus membros.

O nome para Drop.org, surgiu a partir de um erro de digitação de Dries. Dries verificava a disponibilidade para utilizar a palavra belga *dorpje*, que significa vilarejo, e acidentalmente buscou por *drop*. Ao perceber que o nome estava livre, decidiu por utiliza-lo. Já o nome Drupal, pronunciado como “droo-puhl” em inglês, foi escolhido através da derivação da palavra belga *drupel*, que significa gota.

Segundo VanDyk (2007, p. 1), Drupal é um sistema gerenciador de conteúdo de código aberto que utiliza como interface a *web*, e que é modular e possui ênfase na colaboração entre os usuários. É altamente extensível e tem como principio ter um código limpo e que utilize o mínimo de recursos computacionais. Além disso segue todos os padrões definidos para *web*. O núcleo do Drupal possui as funcionalidades básicas e mais comuns, contudo para agregação de funcionalidades adicionais é necessário instalar módulos.

Como recursos básicos, o Drupal oferece:

- Livro colaborativo – Permite a criação de livros entre vários autores.

- *URLs* amigáveis – Facilita o reconhecimento do conteúdo através do endereço onde o mesmo se encontra.
- Módulos – Possibilita estender as funcionalidades básicas.
- Ajuda *online* – Informa a respeito da utilização de seus recursos através de documentação sempre atualizada.
- Personalização – Cada aspecto do sistema pode ser alterado para um proposito específico.
- Busca – Sistema de busca em todo o conteúdo armazenado.
- Gerenciamento de usuários – Permite gerenciar o cadastro de usuários do sistema.
- Permissão baseada em perfil – Utiliza um sistema baseado em perfil, em que permissões são definidas e associadas aos usuários.
- Enquete – Possibilita a criação de enquetes, com gerenciamento automático de cada voto.
- Temas – Utiliza temas para descrever seu visual, separando assim a interface da programação.
- Comentários – Cada informação gerenciada pode receber comentários de usuários ou visitantes anônimos.
- Controle de versão – Utiliza um sistema de versão para controlar alterações realizadas nas informações, permitindo recuperar informações alteradas.
- Agregador de conteúdo – Pode receber informações de terceiros e agregá-las.
- Atalhos permanentes – Permite criar endereços que não irão se alterar mesmo que a informação seja editada.
- Independência de sistema operacional e de banco de dados – O Drupal funciona em qualquer sistema operacional que permita a execução de arquivos em PHP e, por padrão, pode utilizar os banco de dados MySQL e PostgreSQL.
- Multi-língua – Possui um sistema que permite a tradução do conteúdo em diferentes idiomas.
- Auditoria – Guarda informações sobre cada alteração realizada no sistema.

- Administração *web* – Toda sua administração funciona através de um navegador *web*, sendo assim, não precisa de um programa específico para gerenciamento de seu conteúdo.
- Fórum de discussão – Permite argumentação organizada através de tópicos entre os usuários do sistema.
- *Cache* – Acelera o acesso aos dados através de técnicas para guardar páginas previamente processadas.

Na Figura 1 pode-se ver um exemplo de uma página de Drupal com sua instalação padrão e ainda sem nenhum conteúdo adicionado.

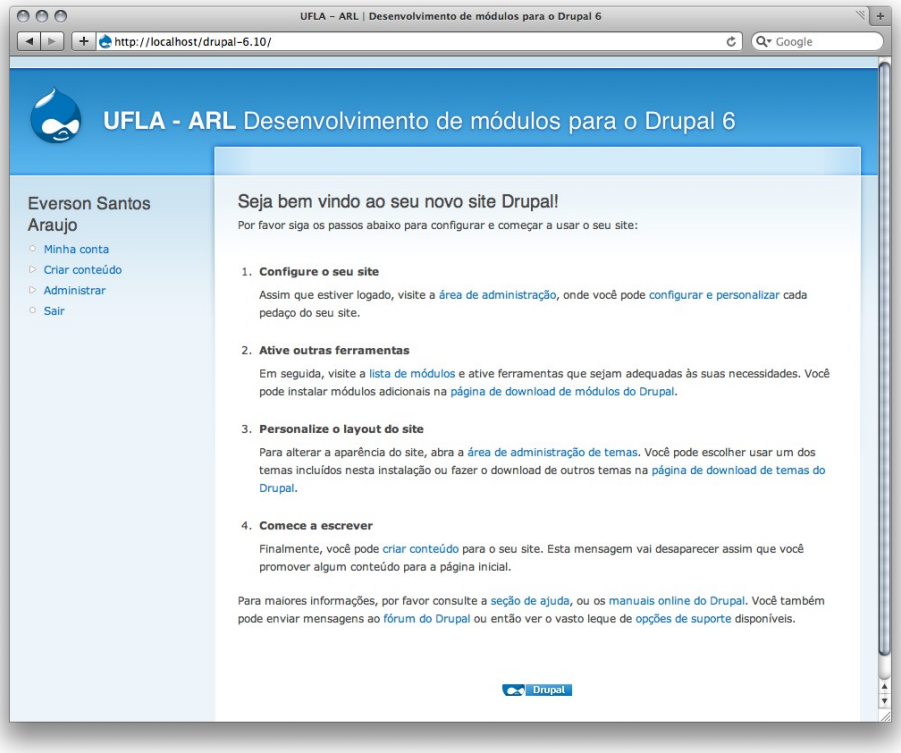

Figura 1: Instalação padrão do Drupal 6

De acordo com Mercer (2008, p. 11), uma grande vantagem da utilização de Drupal é que o seu código é muito bem escrito. Isto facilita sua

modificação caso seja necessário desenvolver novas interações no gerenciamento de conteúdo. O modo como o Drupal foi desenvolvido permite obter grandes vantagens quando comparado com outras plataformas.

2.1 Requisitos de sistema

O Drupal necessita de um ambiente *web* composto por servidores que permitam execução de páginas com interpretação de arquivos PHP e de um Sistema Gerenciador de Banco de Dados suportado.

Especificadamente o Drupal 6 já foi utilizado com sucesso nos servidores de página Apache httpd e Microsoft IIS. Para Apache httpd deve-se utilizar a versão 1.3 ou superior sendo executado em Unix/Linux, OS X ou Windows. Já para Microsoft IIS versão 5 ou superior sendo executado exclusivamente em Windows.

Com relação ao PHP é recomendável a utilização de versão 5.2 ou superior, suportando PHP 4.3.5 ou superior. E como banco de dados são suportados por padrão o MySQL versão 4.1 ou superior e PostgreSQL versão 7.4 ou superior.

2.2 Comunidade

Todo desenvolvimento colaborativo para o Drupal baseia-se em uma comunidade denominada Drupal.org. Esta comunidade possui apoio da Drupal Association, uma sociedade sem fins lucrativos registrada na Bélgica.

Atualmente, a comunidade Drupal.org[1] possui mais de 350.000 membros cadastrados e provê suporte e documentação para o desenvolvimento do Drupal, através de ferramentas de discussão entre os desenvolvedores e espaço para organização de código fonte e suas várias versões. Existe também o

1 http://drupal.org

sistema de grupos[2], que permite uma melhor organização entre grupos de interesse específico dentro do projeto.

Alguns projetos que utilizam Drupal incluem:

- ARL: Administração de Redes Linux, http://arl.ginux.ufla.br/
- Big Ideas da Universidade de BERKELEY, http://bigideas.berkeley.edu/
- Discovery Channel Club, http://discoveryclub.de/
- FOX Movies, http://foxsearchlight.com
- GREENPEACE, http://www.greenpeace.org.uk
- HARVARD, http://harvardscience.harvard.edu/
- Instituto de Artes da Califórnia, http://www.calarts.edu/
- Instituto de Matemática, Estatística e Computação Científica, http://www.ime.unicamp.br/
- Kernel Trap, http://kerneltrap.org/
- Yahoo! Research, http://research.yahoo.com/
- Linux Journal, http://www.linuxjournal.com/
- Meio Bit, http://meiobit.pop.com.br/
- MTV Flux, http://www.mtv.co.uk/channel/flux
- Museu de Artes de Indianópolis, http://www.imamuseum.org/
- NASA - Academia de Engenheiros, http://appel.nasa.gov
- nobios | Everson S.A, http://everson.por.com.br/
- Projeto Musicbox da Sony, http://musicbox.sonybmg.com
- PlayStation na Ásia, http://asia.playstation.com
- Senadora americana Hillary Clinton, http://www.votehillary.org/CMS/
- Spread Firefox, http://www.spreadfirefox.com/
- UBUNTU Linux, http://www.ubuntu.com
- Universidade de Washington (Ciências & Artes), http://artsci.wustl.edu/
- University of Prince Edward Island, http://upei.ca
- Warner Bros Records, http://www.warnerbrosrecords.com/

2 http://groups.drupal.org/

3 Desenvolvimento para Drupal

Drupal é um projeto colaborativo baseado em uma comunidade de desenvolvedores, para que seu código possa ser mantido por outros colaboradores são aplicadas regras de padronização.

Os módulos escritos para Drupal, segundo Butcher (2008, p. 9), são arquivos comuns de códigos para rotinas em PHP agregados a arquivos de suporte e que utilizam a arquitetura do Drupal para integração de novos componentes funcionais. Sendo assim, um módulo em Drupal é mais uma noção que permite utilizar bons princípios de design e desenvolvimento. Estes princípios pregam uma atenção especial para formatação de entrada e saída de dados, bem como uma preocupação quanto a localização e documentação dos arquivos.

3.1 Padrão de código

O padrão de código adotado pelo Drupal é baseado no padrão PEAR - *PHP Extension and Application Repository* (CONVISSOR, JANSEN & MERZ, 2009) e se aplica obrigatoriamente a todo código que deva fazer parte do próprio Drupal.

3.1.1 Indentação

O padrão de indentação determina qual o espaçamento a ser utilizado para identificar blocos de código aninhados. A indentação padrão utilizada no Drupal é de dois espaços em branco sem utilização de caractere de tabulação, também é recomendável a remoção de espaço adicional ao final de cada linha.

3.1.2 Estruturas de controle

As estruturas de controle devem ter um espaço entre a palavra de controle e a abertura dos parênteses, para que possam ser facilmente distinguidas de funções. A Figura 2 permite observar um exemplo à respeito desta definição.

```
if (condição1 || condição2) {
  ação1;
}
elseif (condição3 && condição4) {
  ação2;
}
else {
  ação_padrão;
}
```

Figura 2: Estruturas de controle

A utilização de chaves é recomendada mesmo em situações em que seja tecnicamente opcional, a sua utilização facilita a leitura e entendimento dos blocos de código e diminui a possibilidade de surgirem erros de lógica com a inclusão de novas linhas de código.

3.1.3 Atribuição

Para atribuição de valores deve ser utilizado um espaço em branco ao redor do sinal de igual. Em casos de blocos de atribuição, onde se faz várias atribuições seguidas, é recomendável a utilização de espaços adicionais com a finalidade de facilitar a leitura e identificação do início do conteúdo, como pode ser visto na Figura 3.

```
$var1      = 'valor';
$variavel  = foo($bar);
```

Figura 3: Atribuição de valores

3.1.4 Chamadas de função

Chamadas a funções devem ser realizadas sem adição de espaço entre o nome da função, a abertura do parêntese e, quando houver, o primeiro parâmetro enviado para a função. Para cada parâmetro adicional deve-se separar com a utilização de vírgula e espaço, exceto o último parâmetro que não deve ser espaçado do parêntese de fechamento e do ponto e vírgula, um exemplo de tal utilização pode ser conferido na Figura 4.

```
$var = foo($bar, $bar2, $bar3);
```

Figura 4: Chamadas de função

3.1.5 Declaração de função

A declaração de função deve seguir o mesmo padrão da chamada. Parâmetros que possuam valores padrão devem ser colocados no final da lista de argumentos. O retorno, quando necessário, deve ser de algo com significado. Pode ser visto na Figura 5 um exemplo de uso deste padrão.

```
function foo($bar, $bar2, $bar3='valor padrão') {
  ação;
  return $retorno;
}
```

Figura 5: Declaração de função

3.1.6 Matriz

Matrizes devem ser formatadas com espaços separando cada um de seus elementos e operadores de atribuição, todas as recomendações anteriores sobre espaçamento em chamada de função devem ser levadas em consideração, é possível observar este padrão na Figura 6.

```php
$array = array('bar', 'foo' => 'bar2');
```
Figura 6: Matrizes

Caso a linha de definição dos parâmetros da matriz ultrapasse 80 caracteres, deve-se então ser realizada em formato amplo. Dessa forma cada elemento deve ser declarado em uma linha própria com uma indentação de um nível, para facilitar o acompanhamento do início e fim da matriz.

É recomendável manter a vírgula de separação de elementos inclusive no último elemento da lista. Esta pratica facilita a inclusão de novos elementos, pois não é necessário retornar para incluir a vírgula em uma linha que não está sofrendo alteração direta, um exemplo de aplicação do formato amplo para declaração de matrizes pode ser visto na Figura 7.

```php
$array = array(
  'bar',
  'foo' => 'foo bar bar2',
);
```
Figura 7: Matrizes com mais de 80 caracteres

3.1.7 Comentários

A inclusão de comentários de linha é altamente recomendado, sendo para documentação ou para explicação de determinada seção de código.

Quando para documentação a recomendação é pela convenção Doxygen (HEESCH, 2009), este sistema permite extrair a documentação diretamente do arquivo fonte, ajudando o programador a obter uma referência de todo o código.

Para explicação utiliza-se escrita convencional com primeira letra em maiúsculo e pontuação, um exemplo desta utilização pode ser visto na Figura 8.

```php
// Comentário de exemplo.
$variavel = foo($bar);
```
Figura 8: Comentário em linha

3.2 Segurança

Todo sistema de informação requer um certo nível de segurança, existem vários métodos de ataques a sistemas de informação disponíveis como páginas. As técnicas mais comuns são as de *cross site scripting*[3] e *SQL injection*[4]. Para evitar estes, e outros tipos de ataque, o Drupal oferece funções dedicadas à filtragem e escapamento da entrada e saída de dados.

Para utilização destes recursos de segurança foram definidas três categorias: texto plano, hipertexto e hipertexto administrativo.

Para texto plano é necessário utilizar a função *check_plain()* antes da apresentação do texto ao usuário. Esta função realizará as conversões necessárias para que o texto tenha todos seus caracteres tratados realmente como texto, não permitindo assim a execução de marcações maliciosas.

Para hipertexto deve-se utilizar a função *check_markup()*, esta tratará de limpar os dados produzindo marcação HTML que pode ser livremente apresentada.

Já o hipertexto administrativo é utilizado em seções em que se torna impraticável a utilização do *check_markup()* devido a necessidade de permitir a utilização de algumas marcações específicas. Usa-se então *filter_xss_admin()*, uma função que permite somente as marcações HTML que possam ser consideradas seguras, tentando assim bloquear a utilização de códigos maliciosos mesmo no ambiente de administração da página.

Em específico para segurança contra *SQL injection* é altamente recomendável a utilização das funções de abstração de banco de dados através da função *db_query()* utilizando substituição de argumentos, ao invés de concatenação de literais. E como boa prática sempre enviar a requisição para *db_rewrite_sql()* quando precisar apresentar informações ao usuário, pois esta função se encarrega de acrescentar verificações de permissão de acesso.

---

3 *Cross site scripting* – técnica que permite ao atacante inserir códigos maliciosos em páginas através de formulários que não filtram o conteúdo enviado pelo usuário.

4 *SQL injection* – técnica que permite manipular informações no banco de dados através da definição de variáveis que sejam utilizadas em instruções SQL.

3.3 Abstração de banco de dados

A camada de abstração de banco de dados permite o uso de variados servidores sem necessitar serem realizadas alterações nas requisições *SQL*. Esta camada oferece a preservação da sintaxe básica de *SQL*, enquanto permite ao Drupal controlar as partes específicas de cada requisição que precisem ser escritas de forma diferente para cada tipo de Sistema Gerenciador de Banco de Dados (SGBD).

Por padrão o Drupal inclui definição de abstração para MySQL e PostgreSQL, mas não há impedimento quanto a criação de extensões que complementem esse suporte inicial com a adição de qualquer outro SGBD. Na Tabela 1 é apresentada uma listagem das funções de abstração de Banco de Dados que estão disponíveis no sistema que acompanha o Drupal.

Tabela 1: Funções de abstração de Banco de Dados

Função	Descrição
db_affected_rows	Determina o número de tuplas alteradas na última requisição efetuada.
db_check_setup	Verifica as configurações de acesso ao banco de dados.
db_column_exists	Verifica a existência de determinada coluna em uma tabela.
db_connect	Realiza uma conexão ao banco de dados.
db_decode_blob	Retorna um texto convertido de binário.
db_distinct_field	Adiciona a função DISTINCT a uma requisição, caso a mesma não tenha sido manualmente inserida.
db_encode_blob	Retorna uma conversão apropriada para binário.
db_error	Determina se a requisição anterior retornou algum erro.
db_escape_string	Prepara dados para utilização em uma requisição, prevenindo ataques de SQL injection.
db_escape_table	Restringe nomes de tabelas e colunas para utilizarem apenas caracteres permitidos.
db_fetch_array	Requisita uma tupla, em formato de vetor, referente à requisição anterior.
db_fetch_object	Requisita uma tupla, em formato de objeto, referente à requisição anterior.
db_is_active	Retorna um valor lógico informando a disponibilidade do banco de dados.
db_last_insert_id	Retorna o último id de tupla que foi inserido.
db_lock_table	Trava uma tabela, iniciando automaticamente uma transação.
db_placeholders	Retorna identificação de tipo de dados, para um vetor com um único tipo.

Função	Descrição
db_prefix_tables	Adiciona um banco de dados como prefixo em todas as tabelas da requisição.
db_query	Executa uma requisição no banco de dados ativo.
db_query_range	Executa uma requisição de seleção no banco de dados ativo com uma quantidade de tuplas limitada.
db_query_temporary	Executa uma requisição de seleção guardando seus resultados em uma tabela temporária.
db_result	Retorna um resultado individual da requisição anterior.
db_rewrite_sql	Reescreve requisições a nós, taxonomia e comentários. Utilizada para requisições de listagem.
db_set_active	Ativa um banco de dados para requisições futuras.
db_status_report	Apresenta o estado do banco de dados.
db_table_exists	Verifica se uma dada tabela existe.
db_unlock_tables	Destrava todas as tabelas travadas, finalizando automaticamente uma transação.
db_version	Retorna a versão do Sistema Gerenciador de Banco de Dados.
pager_query	Executa uma requisição paginada.
tablesort_sql	Cria uma clausula de ordenação na requisição.
update_sql	Executa uma requisição de atualização retornando sucesso ou falha.

fonte: API Reference | Drupal API (2009)

3.4 Sistema de menus

O sistema de menus do Drupal define a navegação e o roteamento de requisições de páginas para chamadas a procedimentos, além disso permite controlar o nível de acesso para cada caminho definido.

O sistema de menus segue uma hierarquia definida por caminho, sendo que para cada implementação de um gancho são definidos itens de menu com caminhos específicos e únicos.

O Drupal busca responder a cada requisição realizada através da verificação da existência do caminho, caso não seja encontrado um caminho completo, é realizado a busca por cada caminho imediatamente anterior, até que seja encontrado algum ou retornado um erro de caminho inexistente.

É possível ver um exemplo na Figura 9, nesta é utilizado o caminho */exemplo* para indicar a execução de uma chamada a um possível menu com caminho exemplo. Como o mesmo não está definido em nenhum dos ganchos de menu, é retornada a mensagem de erro de página não encontrada.

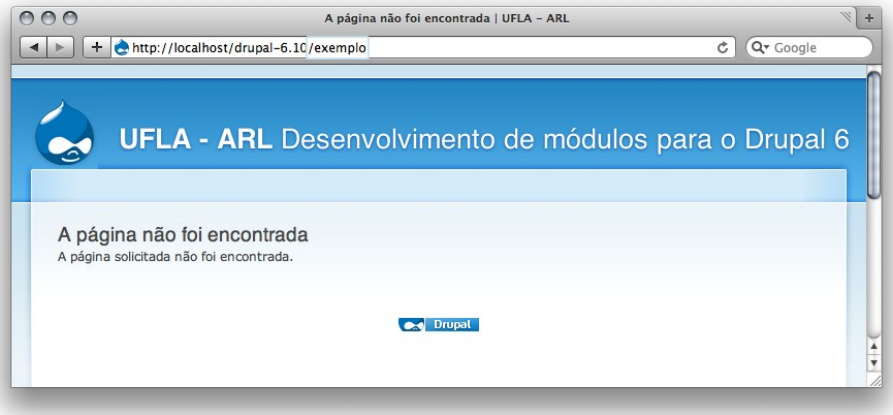

Figura 9: Caminho não encontrado

A implementação de caminhos para menus dá-se pela chamada do gancho *hook_menu()*. Quando utilizada em módulo este gancho segue um padrão de nomenclatura com o nome do modulo seguido de *_menu()*, com isso, para um módulo de nome exemplo teremos *exemplo_menu()*. É possível ver uma implementação básica desta chamada na Figura 10. A partir desta implementação, cada chamada ao caminho */exemplo* deve executar a função *exemplo_ola()*, que por padrão esta disponível no mesmo arquivo.

```
function exemplo_menu() {
  $items['exemplo'] = array(
    'title'           => 'Exemplo',
    'page callback'   => 'exemplo_ola',
    'access callback' => TRUE,
    'type'            => MENU_CALLBACK,
    );
  return $items;
}
```

Figura 10: Implementação de gancho para menu

Devido a sequência de busca, este mesmo código de chamada para *exemplo_ola()* será executado para uma chamada a */exemplo/qualquer/coisa*, caso não haja um caminho definido para */exemplo/qualquer/coisa* nem */exemplo/qualquer*. O restante do caminho, *qualquer* e *coisa* é enviado à chamada da função através de argumentos.

Encontra-se na Figura 11 um exemplo de função que irá retornar 'Ola Mundo!' se não receber nenhum argumento, ou uma listagem dos valores de cada um dos argumentos recebidos.

```php
function exemplo_ola() {
  if (func_num_args() == 0) {
    $retorno = 'Ola mundo!';
  }
  else {
    $argumentos = func_get_args();
    foreach ($argumentos as $valor) {
      $retorno .= '<li>'.$valor.'</li>';
    }
  }
  return $retorno;
}
```

Figura 11: Exemplo de função

A lista de chaves que podem ser utilizados em cada item da matriz do sistema de menu e uma descrição de sua funcionalidade pode ser conferida na Tabela 2.

Tabela 2: Argumentos aceitos para itens no sistema de menu

Argumento	Função
title	Título do item de menu, sem tradução. Obrigatório.
title callback	Função utilizada para gerar o título.
title arguments	Argumento para ser enviado à função de geração de título.
description	A descrição do item de menu, sem tradução.
page callback	Função executada para apresentar uma página quando for acessado o caminho especificado.
page arguments	Uma matriz de argumentos a serem passados para a função de apresentação de página. Valores inteiros correspondem à posição dos argumentos recebidos.

access callback	Função para retornar um valor lógico que determina se o usuário possui acesso ao item.
access arguments	Uma matriz de argumentos a serem passados para a função de verificação de acesso. Valores inteiros correspondem à posição dos argumentos recebidos.
weight	Um valor inteiro que determina a posição do item no menu, quanto maior o valor mais abaixo o item ficará.
menu_name	Adiciona o item a um menu customizado.
type	Máscara que descreve as propriedades do item de menu. Pode ser alimentado com uma das constantes: MENU_NORMAL_ITEM – para um item que pode ser administrado. MENU_CALLBACK – definir um caminho para enviar suas requisições à função especificada. MENU_SUGGESTED_ITEM – o módulo sugere o item, que pode ser ativado pelo administrador. MENU_LOCAL_TASK – adicionado como chamada a execução de um trabalho para seus subitens. MENU_DEFAULT_LOCAL_TASK – cada trabalho deve prover uma chamada que retorna a seu antecessor.

3.5 Geração de formulários

O Drupal oferece um sistema de formulários que permite gerar, validar e processar formulários através da definição de seus campos em formato de uma matriz de valores e propriedades.

Para que se entenda melhor está interação devemos entender como os formulários são processados. Módulos definem formulários utilizando matrizes associativas, que são lidas através de uma chamada a *drupal_get_form()*. É enviado como argumento a esta o nome da função que realiza as definições de formulário, este argumento passa a ser reconhecido como uma chave para o formulário que foi gerado.

Pode-se observar a utilização deste sistema na Figura 12, neste a função *exemplo_ola()* é utilizada para requisitar um formulário que se encontra na função *exemplo_formulario()*. Dessa forma, o nome 'exemplo_formulario' passa a ser a chave para o formulário e através desta chave são reconhecidos os nomes de todas as funções que farão o seu processamento.

Os elementos que podem ser criados pelo sistema de formulários, podem ser encontrados na Tabela 3.

```php
function exemplo_ola() {
  return drupal_get_form('exemplo_formulario');
}
function exemplo_formulario() {
  $form['nome_completo'] = array(
    '#type' => 'textfield',
    '#title' => t('Seu nome'),
    '#description' => t('Preencha com seu nome completo.'),
  );
  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => t('Enviar')
  );
  return $form;
}
```

Figura 12: Utilização de formulários

Tabela 3: Tipos de elemento para geração de formulário

Tipo	Descrição	Propriedades
button	Cria um botão de ação. Ao ser pressionado, é realizada a chamada ao Drupal que valida e reconstrói o formulário.	#ahah, #attributes, #button_type, #executes_submit_callback, #name, #prefix, #suffix, #type. #value, #weight
checkbox	Cria uma caixa de seleção.	#ahah, #attributes, #default_value, #description, #prefix, #required, #return_value, #suffix, #title, #type. #weight
checkboxes	Cria um conjunto de caixas de seleção. A propriedade #options recebe uma matriz associativa com a chave sendo utilizada como valor de retorno e o valor é apresentado para seleção.	#attributes, #default_value, #description, #options, #prefix, #required, #suffix, #title, #tree, #type. #weight
date	Cria uma caixa de seleção de data. Possui valor padrão a data de hoje. Retorna uma matriz com três elementos: *year*, *month*, *day*.	#attributes, #default_value, #description, #prefix, #required, #suffix, #title, #type. #weight
fieldset	Agrupamento de itens.	#attributes, #collapsed, #collapsible, #description, #prefix, #suffix, #title, #type. #weight
file	Cria um campo para envio de arquivos. É necessário alterar os atributos do formulário incluindo o tipo de condificação, exemplo: $form['#attributes'] = array('enctype' => "multipart/form-data");	#attributes, #description, #prefix, #size, #suffix, #title, #type. #weight
form	Um formulário para receber elementos.	#action, #attributes, #method, #prefix, #submit, #suffix, #theme, #validate

Tipo	Descrição	Propriedades
hidden	Guarda informações sem apresenta-las.	#prefix, #suffix, #type, #value
image_button	Apresenta um botão de ação utilizando uma imagem.	#ahah, #attributes, #button_type, #executes_submit_callback, #name, #prefix, #src, #suffix, #submit, #type. #value, #weight
markup	Gera uma marcação genérica.	#attributes, #prefix, #suffix, #type. #value, #weight
item	Apresenta um item para visualização.	#description, #prefix, #required, #suffix, #title, #type, #value, #weight
password	Cria um campo de texto que não permite visualizar seu conteúdo.	#ahah, #attributes, #description, #maxlength, #prefix, #required, #size, #suffix, #title, #type. #weight
radio	Cria uma caixa de opção.	#ahah, #attributes, #default_value, #description, #prefix, #required, #suffix, #title, #type. #weight
radios	Cria um conjunto de caixas de opção. A propriedade #options recebe uma matriz associativa com a chave sendo utilizada como valor de retorno e o valor é apresentado para seleção.	#attributes, #default_value, #description, #options, #prefix, #required, #suffix, #title, #type. #weight
select	Formata uma caixa de seleção suspensa.	#ahah, #attributes, #default_value, #description, #multiple, #options, #prefix, #required, #suffix, #title, #type. #weight
submit	Cria um botão para enviar o formulário.	#ahah, #attributes, #button_type, #executes_submit_callback, #name, #prefix, #suffix, #type. #value, #weight
textarea	Cria um campo de texto com múltiplas linhas.	#ahah, #attributes, #cols, #default_value, #description, #prefix, #required, #suffix, #title, #type. #rows, #weight
textfield	Cria um campo de texto de linha única.	#ahah, #attributes, #autocomplete_path, #default_value, #description, #field_prefix, #field_suffix, #maxlength, #prefix, #required, #size, #suffix, #title, #type. #weight
value	Um valor criado internamente no formulário mas não apresentado na tela.	#type, #value
weight	Cria um menu de seleção de peso.	#attributes, #delta, #default_value, #description, #prefix, #required, #suffix, #title, #type. #weight

O sistema de formulários permite a criação de funções de validação através de chamada a funções relacionadas à chave utilizada para o formulário, sendo a função de validação nomeada pela chave seguido de *_validate()*. Para o exemplo supra-citado deve-se utilizar a função *exemplo_formulario_validate()*, como é possível perceber na Figura 13.

Esta função de validação recebe como argumentos o formulário e seu estado atual. Através disto é possível identificar e validar cada um dos valores recebidos ao acessar valores na matriz *values* que são identificados pelo nome do campo no formulário. Quando é identificado um erro de validação, a função *form_set_error()* é utilizada para indicar o campo e a mensagem de erro da validação que será retornada.

```
function exemplo_formulario_validate($form, &$form_state) {
  if (strlen($form_state['values']['nome_completo'])<2) {
    form_set_error('nome_completo',
      t('Seu nome deve ser preenchido com pelo menos 2 caracteres.'));
  }
}
```

Figura 13: Validação de formulário

O Drupal encarrega-se então de criar a marcação para o formulário, realizar a validação quando houver uma função para isso, e em caso de erros de validação, marcar os campos que tenham sido mal preenchidos.

Pode-se observar na Figura 14, um retorno de erro da validação devido ao preenchimento incorreto do campo nome_completo que possui o título “Seu nome”.

Segundo a definição da função de validação, o campo nome_completo deve receber pelo menos dois caracteres e a mensagem de erro apresenta essa limitação ao usuário. Dessa forma ele poderá reconhecer e corrigir o problema que está sendo apresentado.

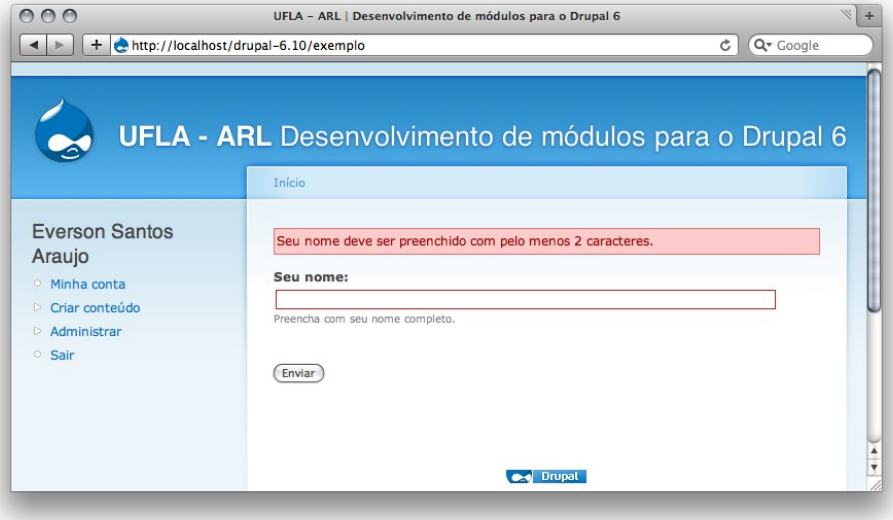

Figura 14: Retorno de erro da validação.

Existe também uma chamada para função a ser executada quando o recebimento do formulário for realizado com êxito. Esta função é nomeada pela chave do formulário seguido de *_submit()*.

Pode-se perceber a utilização desta função na Figura 15, que apresenta uma mensagem de agradecimento quando o formulário for preenchido corretamente.

```
function exemplo_formulario_submit($form, &$form_state) {
  $nome = $form_state['values']['nome_completo'];
  drupal_set_message(t('Obrigado, %nome.', array('%nome' => $nome)));
}
```

Figura 15: Função para êxito no recebimento do formulário

3.6 Nós

Drupal utiliza a noção de nó para manipular informação, desse modo cada conteúdo que seja indexado pelo Drupal recebe um identificador de nó, um

título e um corpo. A partir deste conjunto básico e para cada tipo de conteúdo este nó pode ser estendido para receber informações adicionais que se tornarem necessárias.

A Figura 16 mostra a representação de tipos de conteúdo que existem por padrão no Drupal, a partir destes é possível perceber como o Drupal define nó de forma abstrata para poder ser facilmente estendido a qualquer tipo de conteúdo, porém mantendo uma base única para facilitar o tratamento da informação.

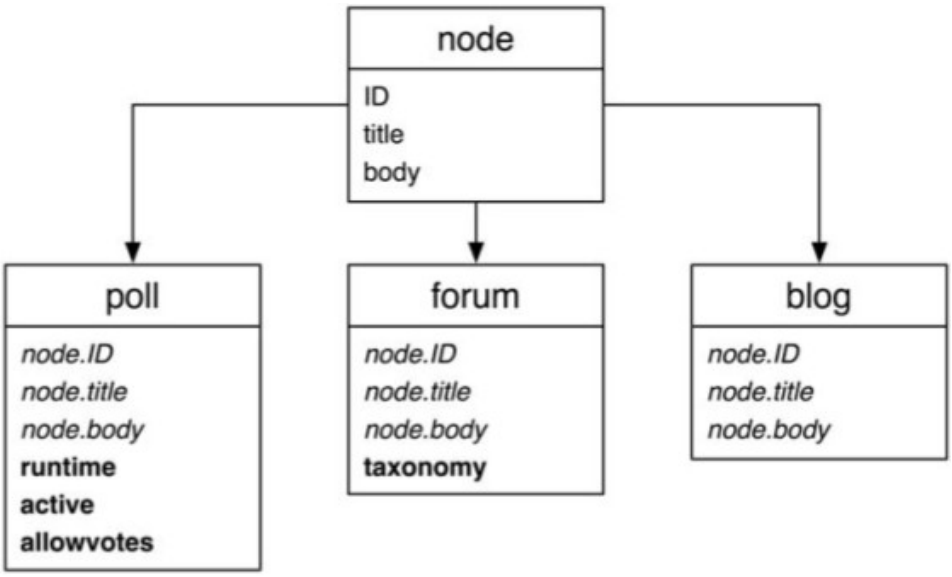

Figura 16: Tipos de conteúdo derivando de nó

Fonte: VANDYKE, 2008, p. 138

Cada tipo de conteúdo adiciona atributos específicos, como por exemplo o tipo de conteúdo enquete (*poll*) que deve guardar informações como: duração (*runtime*), informação se a enquete está atualmente ativa (*active*) e se é permitido aos usuários votarem (*allowvotes*).

3.7 Taxonomias

As taxonomias são utilizadas para classificar coisas, o Drupal possui um módulo de taxonomia que acompanha o sistema básico e que tem a função de administrar os termos utilizados para classificação.

O módulo de taxonomia guarda as informações sobre cada um dos termos cadastrados utilizando números de identificação. Desta forma os termos podem ser alterados sem que se perca as ligações que forem realizadas do mesmo com cada um dos itens ao qual se referem.

Através deste sistema é possível definir vocabulários que podem ser associados a tipos de nós, este tipo de associação permite realizar uma categorização mesmo entre nós de tipos diferentes.

Existem dois tipos de taxonomias, as listagens diretas e as hierarquias. Nas listagens diretas todos os termos tem o mesmo peso e estão diretamente ligados, já na hierarquia existe a ordenação de pais e filhos, sendo que em cada termo deve-se escolher seus termos superiores.

O sistema de taxonomias ainda permite a definição de termos sinônimos que estarão ligados ao termo principal e poderão ser utilizados na categorização sendo interligados a um único termo.

Existem funções específicas para recuperação de informações sobre vocabulários. A função *taxonomy_vocabulary_load()* recebe como parâmetro um identificador de vocabulário e retorna um objeto contendo os termos associados. Já a função *taxonomy_get_vocabularies()* recebe como parâmetro um tipo de nó e retorna uma matriz com todos os vocabulários interligados a este tipo de nó.

3.8 Constantes e variáveis globais

O Drupal define um conjunto de constantes e variáveis globais que devem ser conhecidas para evitar que haja sobreposição de informações e permitir o melhor uso das informações que já estejam disponíveis. Na Tabela 4 pode-se observar a listagem das variáveis globais definidas no Drupal.

Tabela 4: Variáveis globais

Variável	Descrição
$active_db	Conexão ativa com o banco de dados.
$base_path	O caminho para a localização dos arquivos do Drupal.
$base_root	A raiz do caminho da máquina.
$base_theme_info	Uma matriz de objetos que representam o tema básico.
$base_url	O caminho da máquina onde se encontra o Drupal.
$channel	Uma matriz associativa contendo título, ligação, descrição e outras chaves.
$conf	Uma matriz de variáveis persistentes guardadas na tabela 'variables'.
$cookie_domain	O domínio a ser utilizado em *cookies* de sessão.
$custom_theme	Nome do tema que deve sobrepor o tema padrão.
$db_prefix	Prefixo para ser utilizado em todas as tabelas do banco de dados.
$db_type	Tipo de banco de dados que está sendo utilizado.
$db_url	O caminho para o banco de dados.
$element	Uma matriz estruturada que descreve os dados que serão apresentados.
$forum_topic_list_header	Uma matriz de informações sobre cabeçalhos de tópicos.
$id	Estatística ativa.
$image	Imagem utilizada atualmente pelo agregador.
$installed_profile	Nome do perfil que acaba de ser instalado.
$install_locale	A configuração de localização a ser utilizada na instalação.
$item	Um literal ou matriz geral.
$items	Uma matriz de itens usadas pelo agregador.
$language	Um objeto contendo informações sobre a linguagem ativa.
$last_result	Recurso da última requisição ao banco de dados.
$menu_admin	Valor lógico indicando se está sendo executada a checagem de acesso do menu.
$multibyte	O modo atual de multibyte. Valores possíveis: UNICODE_ERROR, UNICODE_SINGLEBYTE, UNICODE_MULTIBYTE.
$nid	Identificação de nó ativo.
$pager_page_array	Resultado da função *pager_query()* que é utilizada por outras funções.
$profile	O nome do perfil instalado atualmente.
$queries	Uma matriz com as requisições que foram realizadas ao banco de dados.
$recent_activity	Estatísticas de atividade recente.
$tag	Nome de marcador ativo.

Variável	Descrição
$theme	Nome do tema ativo.
$theme_engine	O motor de tema relacionado ao tema ativo.
$theme_info	Objeto contendo informações do tema ativo.
$theme_key	Nome do tema ativo.
$timers	Repositório para temporizadores.
$update_free_access	Permite a execução do arquivo update.php quando não é possível identificar-se como administrador.
$update_mode	Desabilita chamadas a ganchos durante o processo de atualização.
$user	Representação do usuário atual, guarda preferências e outras informações.
$xrds_current_service	Matriz usada pelo sistema de autenticação OpenID.
$xrds_open_elements	Matriz usada pelo sistema de autenticação OpenID.
$xrds_services	Matriz usada pelo sistema de autenticação OpenID.

3.9 Comentários finais

Neste capítulo foram apresentados os principais conceitos necessários para o desenvolvimento no Drupal. É importante observar que boa parte dos conceitos apresentados não são utilizados diretamente, mas são importantes para conhecer o funcionamento do Drupal.

Em geral, estes recursos são utilizados através de módulos para prover novos recursos e funcionalidades ao serem agregados ao núcleo do Drupal. O próximo capítulo irá abordar detalhadamente o desenvolvimento de módulos.

4 Desenvolvimento de módulos

Um módulo é uma coleção de funções que se agregam ao Drupal com a finalidade de prover novas funcionalidades. Estas funcionalidades são adicionadas através da utilização de ganchos existentes no núcleo do sistema que permitem a execução de códigos em pontos específicos do processamento da página. Os módulos para Drupal dividem-se de acordo com sua fonte em três categorias:

- Núcleo – são desenvolvidos e aprovados pelo time de núcleo e a comunidade.
- Contribuídos – são desenvolvidos pela comunidade, utilizam o sistema do Drupal como repositório e são compartilhados através da Licença Pública GNU (GPL).
- Customizados – são criadas pelos desenvolvedores de um site e podem ou não ser compartilhados.

Cada módulo possui uma pasta própria utilizando o seu nome e localizada dentro da pasta *modules*, que se encontra dentro da pasta *all*, que por sua vez se encontra dentro da pasta *sites*, como pode ser observado na Figura 17. Esta obrigatoriedade permite garantir a organização dos arquivos de cada modulo para que não interfiram nos demais.

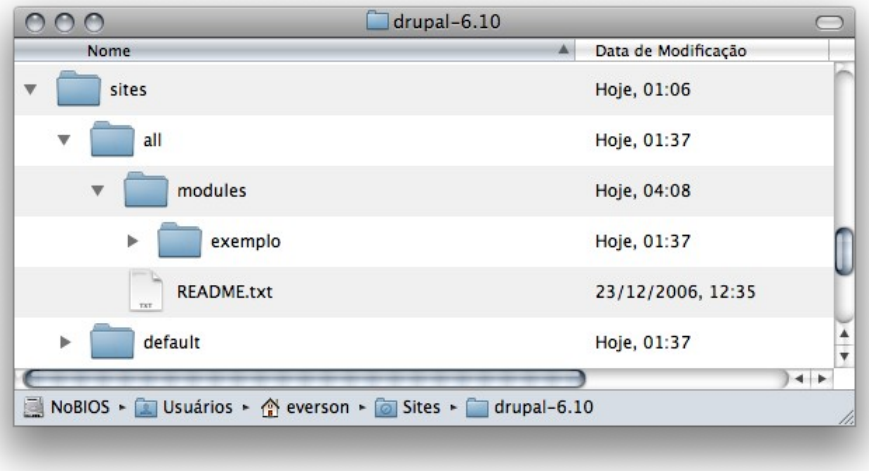

Figura 17: Localização da pasta modules

4.1 Informações sobre módulo

Para que um modulo seja reconhecido pelo sistema de gerenciamento de módulos do Drupal, é necessário criar um arquivo com meta-informações, este arquivo é nomeado como *nomedomodulo.info*. Neste deve-se apresentar o nome completo do módulo, descrição e demais informações relevantes à sua instalação e manipulação.

Seu formato dá-se por campo = valor em uma única linha, e possui como campos obrigatórios: *name*, *description* e *core*. Pode-se ver um exemplo deste arquivo na Figura 18, onde se define o módulo exemplo utilizando o arquivo *exemplo.info*.

```
name = Exemplo
description = Exemplo de módulo para Drupal 6.
package = TCC - UFLA, ARL
core = 6.x
```

Figura 18: Exemplo de arquivo com informações do módulo.

Os campos para informações são:

- *name* – O nome apresentado pelo módulo. Deve seguir o guia de capitalização onde apenas a primeira letra de cada palavra é apresentada em maiúsculo.
- *description* – Apresenta uma descrição curta, de apenas uma linha, que irá informar ao administrador do site qual é a função principal do módulo. É recomendável não utilizar descrições com mais de 255 caracteres.
- *core* – Apresenta a versão do Drupal ao qual o módulo é compatível.
- *dependencies* – Uma matriz contendo o nome dos módulos que são necessários. Deve-se escrever o nome dos módulos em minúculo, sem espaços e excluindo o sufixo *.module*. Deve ser identificado como matriz, sendo assim, deve seguir o modelo de *dependencies[] = modulo*.

- *package* – Agrupamento para pacotes, permite que seus módulos possam ser organizados em pacotes.
- php – Especifica uma versão mínima do PHP para ser executado.

Além do arquivo contendo as informações sobre o módulo, é necessário o arquivo que será responsável por implementar as funções do módulo. Este arquivo é nomeado como *nomedomodulo.module*, segundo o exemplo criado para definir informações, tem-se o arquivo *exemplo.module*.

4.2 Instalação do módulo

Durante a ativação do módulo, o Drupal permite que sejam executadas funções que preparam o ambiente para o bom funcionamento do módulo. Estas funções são geralmente utilizadas para definir tabelas e campos específicos do módulo.

Para definir as funções de instalação é utilizado um arquivo *.install*, e este arquivo deve ter o nome do módulo. Neste arquivo deve existir a função com o nome do módulo seguido de *_install()*.

Além de servir para instalação, este arquivo também serve para atualização entre versões do módulo, definindo o que deve ser alterado na estrutura do sistema para que a nova versão funcione corretamente. Estas funções são nomeadas com o nome do módulo seguido de *_update_N()*, onde N define um número de identificação da atualização que está sendo instalada. Assim, o Drupal pode acompanhar quais atualizações já foram realizadas e quais ainda precisam ser aplicadas.

Este mesmo arquivo deve descrever as funções que serão executadas quando o módulo for removido do sistema, fazendo assim com que o ambiente fique livre dos dados que foram criados e eram necessários exclusivamente para o funcionamento do módulo, estas funções de remoção são nomeadas com o nome do módulo seguido de *_uninstall()*.

4.3 Ganchos e gatilhos

O processamento do Drupal é realizado através de uma série de eventos, estes eventos internos são definidos como ganchos pois ao ocorrerem o Drupal permite que módulos realizem operações no caminho da execução. O desenvolvimento típico de módulos dá-se pela decisão sobre quais eventos do Drupal deseja reagir, ou seja, quais ganchos devem ser implementados.

A utilização dos ganchos dá-se através da criação de funções que são nomeadas utilizando o nome do módulo seguido do nome do gancho. A palavra chave *hook* é utilizada para indicar o local onde deverá ser utilizado o nome do módulo. Por exemplo, caso esteja sendo criado o módulo de nome exemplo e para este seja necessário implementar o *hook_help()*, o nome de sua função será definida como *exemplo_help()*.

A Tabela 5 apresenta os ganchos que existem no sistema padrão e que permitem a interação necessária para criação de módulos.

Tabela 5: Ganchos

Gancho	Descrição
hook_access	Define restrições de acesso.
hook_actions_delete	Permite execução de código após a remoção de uma ação.
hook_action_info	Declaração de informações sobre ação do Drupal.
hook_action_info_alter	Altera ações declaradas por outros módulos.
hook_block	Declara um bloco ou grupo de blocos.
hook_boot	Realiza trabalho de configuração.
hook_comment	Ações a serem realizadas em comentários.
hook_cron	Realiza ações periódicas.
hook_db_rewrite_sql	Reescreve requisições ao banco de dados, geralmente para controle de acesso.
hook_delete	Responde à ação de remoção de nós.
hook_disable	Ações necessárias antes da desabilitação de um módulo.
hook_elements	Permite aos módulos declararem tipos de elementos para formulários.
hook_enable	Realiza ações necessárias após a ativação do módulo.
hook_exit	Permite realizar ações de limpeza.

Gancho	Descrição
hook_file_download	Controle de acesso a arquivos privativos e definição de cabeçalhos.
hook_filter	Define filtros de conteúdo.
hook_filter_tips	Provê recursos de ajuda para utilização de filtros.
hook_flush_caches	Adiciona uma lista de tabelas de cache que serão limpas.
hook_footer	Insere marcações para fechamento de comandos em HTML.
hook_form	Apresenta um nó de formulário.
hook_forms	Mapeia um identificador de formulário com suas funções de criação.
hook_form_alter	Altera formulários antes de sua apresentação.
hook_form_FORM_ID_alter	Permite alteração específica em um id de formulário definido no nome da função.
hook_help	Prover ajuda ao usuário.
hook_hook_info	Expande uma lista de gatilhos que seu módulo define para que os usuários definam ações.
hook_init	Realiza ações de configuração.
hook_insert	Responde à inserção de nós.
hook_install	Utilizado para instalação de versões de esquema para o banco de dados, e demais trabalhos de configuração.
hook_link	Define ligações internas do Drupal.
hook_link_alter	Altera definições de ligações antes de apresentar nós.
hook_load	Carrega informações específicas ao nó.
hook_locale	Permite ao módulo definir um grupo de texto que podem ser traduzidos.
hook_mail	Prepara uma mensagem para ser enviada por correio eletrônico.
hook_mail_alter	Altera aspectos de correios eletrônicos enviados pelo Drupal.
hook_menu	Define itens de menu e chamadas de páginas.
hook_menu_alter	Permite alteração da definição de menus.
hook_menu_link_alter	Permite alteração de ligações dos menus.
hook_nodeapi	Ações em nós definidos por outros módulos.
hook_node_access_records	Guarda permissões para um nó no banco de dados.
hook_node_grants	Informa ao sistema de acesso a nós quais permissões o usuário possui.
hook_node_info	Define tipo de nó.
hook_node_operations	Operações de massa a serem executadas em nós.
hook_node_type	Ao ser realizada mudança em um tipo de nó.
hook_perm	Define permissões de usuário.

Gancho	Descrição
hook_ping	Utilizado para notificar outras páginas sobre atualizações na sua página.
hook_prepare	É chamado após o carregamento e antes da apresentação de um nó em um formulário de adição ou edição.
hook_profile_alter	Realizar alterações em itens do perfil antes de serem apresentados.
hook_requirements	Verificar requisitos para instalação e apresentar um relatório de estado.
hook_schema	Define a versão atual do esquema de banco de dados.
hook_schema_alter	Realiza alterações na versão atual do esquema de banco de dados.
hook_search	Define uma nova rotina de busca.
hook_search_preprocess	Pré-processamento de texto para indexação da busca.
hook_system_info_alter	Altera informações recebidas de arquivos de informação sobre módulos e temas.
hook_taxonomy	Atua em alterações de termos de taxonomia.
hook_term_path	Permite aos módulos criarem caminhos alternativos a termos.
hook_theme	Registra uma implementação de tema.
hook_theme_registry_alter	Altera informação de registro de tema.
hook_translated_menu_link_alter	Altera a ligação de um item de menu após sua tradução mas antes da apresentação.
hook_translation_link_alter	Realiza alterações em ligações traduzidas.
hook_uninstall	Utilizado para remover tabelas e variáveis criadas pelo módulo.
hook_update	Responde a ação de atualização de um nó.
hook_update_index	Atualiza a indexação do Drupal.
hook_update_last_removed	Retorna um número de atualização que não está mais disponível.
hook_update_N	Realiza uma atualização.
hook_update_status_alter	Altera a informação sobre atualizações disponíveis para projetos.
hook_user	Atua em ações da conta do usuário.
hook_user_operations	Operações de massa a serem realizadas em usuários.
hook_validate	Valida dados de um nó com formulário.
hook_view	Apresenta um nó.
hook_watchdog	Guarda uma mensagem para um evento.
hook_xmlrpc	Registra chamadas para chamadas de procedimento remoto em XML.
module_hook	Determina se um módulo implementa um gancho.
module_implements	Determina quais módulos implementam um gancho.

Gancho	Descrição
module_invoke	Chama um gancho em um módulo específico.
module_invoke_all	Chama um gancho em todos os módulos ativos que possuam uma implementação.

fonte: LEE, Wilson. (2009)

Em adição ao sistema de ganchos existe o sistema de gatilhos. Um gatilho permite ao administrador da página escolher uma ação que será realizada quando ocorrer um evento definido. Então diferente dos ganchos que só podem ser manipulados através de alteração de arquivos em PHP, os gatilhos permitem uma maior facilidade de administração através de um ambiente próprio.

A manipulação de gatilhos é possível através da ativação do módulo *trigers*, que apesar de acompanhar o Drupal não está habilitado por padrão.

O sistema de gatilhos permite criar eventos através do gancho *hook_hook_info*, através deste se descreve uma matriz associativa que indica o módulo à partir do qual será identificado o evento, o gancho que está sendo criado e cada uma dos tipos de eventos e sua descrição. Um exemplo de implementação pode ser visto na Figura 19.

```
function exemplo_hook_info() {
  return array(
    'exemplo' => array(
      'exemplo' => array(
        'ver'=>array(
          'runs when' => t('Exemplo é visto')
          ),
        'enviar'=>array(
          'runs when' => t('Formulário de exemplo é enviado')
          ),
      ),
    ),
  );
}
```

Figura 19: Utilização de gancho para definir informação sobre eventos

Também é possível definir ações ao utilizar o gancho *hook_action_info*, neste deve-se retornar o caminho para a ação e uma matriz com informações adicionais como uma descrição e uma lista dos eventos suportados.

É possível ver uma implementação de informações sobre uma ação na Figura 20, onde se define a ação *exemplo_action* e que a mesma pode ser configurada para o evento enviar na definição de gatilhos para o módulo exemplo.

```
function exemplo_action_info() {
  return array (
    'exemplo_action' => array(
      'type'         => 'exemplo',
      'description'  => t('Registrar envio'),
      'configurable' => FALSE,
      'hooks'        => array(
        'exemplo'  => array('enviar'),
       ),
      ),
  );
}
```

Figura 20: Utilização de gancho para definir ações

Estas implementações permitem ao administrador selecionar os gatilhos e ações através da interface administrativa de gatilhos, esta interface está disponível no menu Administrar com o nome de Gatilhos.

A seleção é realizada pela escolha do módulo e posterior escolha das ações para cada um dos gatilhos definidos para o módulo. No exemplo que pode ser visto na Figura 21 é possível ver a ação 'Registrar envio' ser selecionada para o gatilho 'Formulário de exemplo é enviado'. Dessa forma, a cada chamada em que exista o evento 'Formulário de exemplo é enviado', haverá a execução das ações definidas em 'Registrar envio'.

Figura 21: Interface para administração de gatilhos

4.4 Sistema de permissões

Por utilizar uma estrutura básica para guardar praticamente todo tipo de informação. O Drupal trabalha com um sistema de permissão de acesso baseado em nós. Este sistema utiliza uma verificação que retorna um valor lógico que define se será permitido ou não o acesso a determinado nó.

Para determinar o acesso a um nó é utilizada a função *node_access()*, esta verifica inicialmente se o usuário possui permissão de administração de nós (*administer nodes*), esta permissão dá acesso irrestrito a qualquer conteúdo. Então, o módulo específico do conteúdo do nó pode executar o gancho para acesso utilizando nome do módulo seguido de *_access()*. Dessa forma é possível oferecer acesso irrestrito a um perfil de usuário para um tipo de conteúdo. Caso

não exista um gancho ou o mesmo retorne um valor nulo, então a tabela de acesso a nós (*node_access*) é utilizada para determinar o acesso ao conteúdo.

O sistema de controle de acesso permite a criação de novos tipos de permissão através da chamada ao gancho de permissões, esta chamada é feita pelo nome do módulo seguido de *_perm()*. Este gancho requer o retorno de uma matriz com os valores de permissão que devam ser criados.

Na Figura 22 é possível observar um exemplo de registro de novas permissões, a nomenclatura utilizada não tem qualquer significado para o Drupal, devem então ser utilizadas frases que tenham sentido para o usuário.

```
function exemplo_perm() {
  return array(
    'usar formulário de exemplo',
  );
}
```

Figura 22: Definição de permissão

Com a definição de permissão, é possível verificar se ela existe em cada etapa que se fizer necessário verificar determinada permissão. O modo mais direto de utilização de uma determinada permissão dá-se pela chamada de verificação de acesso a um determinado caminho.

Para o exemplo criado, será alterada a chamada ao formulário de exemplo, como pode ser visto na Figura 23, que passará a estar disponível apenas para usuários que tenham a permissão 'usar formulário de exemplo'.

```
function exemplo_menu() {
  $items['exemplo'] = array(
    'title'            => 'Exemplo',
    'page callback'    => 'exemplo_ola',
    'access callback'  => 'user_access',
    'access arguments' => array('usar formulário de exemplo'),
    'type'             => MENU_CALLBACK,
    );
  return $items;
}
```

Figura 23: Verificação de acesso

Desta forma, a cada chamada ao caminho */exemplo*, será executada a função *user_access()*, com o argumento 'usar formulário de exemplo' e dessa forma será verificado se o usuário atual possui o acesso especificado.

4.5 Blocos

Os blocos são utilizados para apresentar conteúdos auxiliares. Através de seu sistema é criado um local para apresentação de uma pequena informação em alguma região definida pelo tema do site. Na Figura 25 é possível observar as regiões: Barra lateral esquerda, Cabeçalho e Barra lateral direita.

O sistema de criação de blocos permite controlar quem o verá e em quais páginas estará visível. Os blocos podem ser definidos através da interface do Drupal ou através da criação de um gancho para *hook_block*.

O gancho *hook_block* é responsável por apresentar o bloco e prover todas as funções administrativas e auxiliares que forem necessárias ao bloco.

Este gancho recebe 3 parâmetros, no primeiro se define a fase que está sendo carregada, o segundo recebe o número de identificação do bloco a ser retornado e o terceiro uma matriz com os dados do formulário de configuração do bloco. As fases que o sistema de blocos provê são:

- *list* – Deve retornar uma matriz com todos os blocos definidos pelo módulo
- *configure* – Retorna uma definição de formulário para campos que devem ser utilizados para configurar o bloco.
- *save* – Ocorre quando o formulário de configuração é recebido e retorna os dados do formulário de configuração no terceiro parâmetro do gatilho.
- *view* – Quando o bloco está sendo apresentado, retorna uma matriz com o título do bloco e seu conteúdo.

A Figura 24 mostra um código para definição de um bloco de exemplo que será utilizado para apresentar a informação de versão retornada pelo servidor de páginas Apache.

```php
function exemplo_block($op = 'list', $delta = 0, $edit = array()) {
  switch ($op) {
    case 'list':
      $blocks[0]['info']  = t('Informações do servidor');
      $blocks[0]['cache'] = BLOCK_NO_CACHE;
      return $blocks;
    case 'view':
      $block['subject'] = t('Servidor');
      $block['content'] = apache_get_version();
      return $block;
  }
}
```

Figura 24: Exemplo de definição de bloco

Após sua definição é possível habilitar o bloco e definir em qual região o mesmo será apresentado através da interface de administração de blocos, como pode ser visto na Figura 25.

Figura 25: Configuração de blocos

4.6 Registros do sistema

O Drupal oferece um sistema de registros que pode ser utilizado para gravar informações a respeito dos eventos que ocorrem no sistema, permitindo assim uma maior facilidade na depuração de código.

Este sistema de registro está disponível através da função *watchdog()*, esta função recebe como argumentos obrigatórios o tipo de mensagem e a mensagem, pode também receber a gravidade da mensagem e um atalho.

Na Figura 26 pode-se ver um exemplo de utilização do sistema de registro, neste é criado um registro para a ação de recebimento de formulário.

```
function exemplo_action(&$object, $context = array()) {
  watchdog('Exemplo', 'O formulário de exemplo foi recebido');
}
```

Figura 26: Função com exemplo de utilização do sistema de registro

É possível ver esta mensagem no sistema de registro do Drupal, como pode ser observado na Figura 27. Percebe-se que o sistema guarda a mensagem com a data da ocorrência e o usuário que estava ativo no momento.

Figura 27: Exemplo de registro

4.7 Comentário Finais

Neste capítulo foram vistos conceitos e a utilização destes em módulos, permitindo assim a adição de novas funcionalidades ao Drupal através de seus ganchos de execução.

No próximo capítulo será visto a aplicação dos conceitos vistos até aqui na migração para Drupal 6 de um módulo com suporte ao Drupal 5.

5 Estudo de caso: ejournal

5.1 Sobre o ejournal

O ejournal[5] é um módulo para Drupal criado por Roman Chyla. O mesmo foi iniciado em 2006 com suporte ao Drupal 4.6.0. Hoje está disponível oficialmente para o Drupal 5 e possui uma versão alfa para Drupal 6.

O módulo foi desenvolvido para a publicação de um periódico eletrônico e teve seu modelo baseado no Open Journal Systems (OJS)[6] que é um sistema de publicação livre amplamente utilizado.

O ejournal oferece a administração de várias publicações em um mesmo sistema. Possui os recursos de administração de edições, controle de acesso, utilização de vocabulários para classificação de conteúdo e escritores, controle de referências e citações e utiliza temas.

Estes recursos são divididos em quatro submódulos permitindo que o administrador selecione apenas as funções desejadas. Estes submódulos são:

- E-journal *access* – para controle de acesso
- E-journal *authors* – para conectar termos de taxonomia a usuários registrados no Drupal
- E-journal *citations* – para inserção de citações formatadas no artigo
- E-journal *shortly* – para publicação de nós interligados a edições

Para que possa ser instalado o ejournal depende dos módulos *taxonomy* e *token*, sendo o primeiro parte do pacote básico do Drupal e o segundo um módulo contribuído disponível nos repositórios oficiais do Drupal.

5 http://drupal.org/project/ejournal
6 http://pkp.sfu.ca/ojs/

5.2 Motivação

Com o lançamento do Drupal 6 se tornou-se necessário a atualização do ejournal, a fim de permitir a continuidade de sua utilização por quem depende do mesmo para seus sistemas. Esta versão atual do Drupal também trouxe novas funcionalidades que poderiam permitir ao ejournal cumprir melhor sua função e acrescentar recursos.

Neste tempo a revista acadêmica Bazar[7] passou a buscar um sistema de publicação que permite melhorar seus processos e adequar-se ao restante do ambiente que estava sendo utilizado. Com isso foi verificado a existência do ejournal, que não realiza tudo que é necessário para a Bazar, mas é um trabalho em andamento que pode ser adaptado para sua necessidade específica.

Das funções específicas da revista eletrônica Bazar que não são contempladas pelo ejournal, pode-se citar o controle e especificação de revisores para artigos, identificação de revisor para cada seção e comunicação administrativas sobre cada passo da submissão e revisão dos artigos.

Algumas das necessidades da Bazar já estavam disponíveis no Drupal 6, porém o ejournal ainda não estava disponível nesta versão. Disso surgiu o contato com o desenvolvedor do ejournal para apresentar essa necessidade, mas o mesmo não possuía tempo para dedicar-se a essa atualização.

5.3 Migração de módulos de Drupal 5 para Drupal 6

A versão 6 do Drupal incluiu um grande número de mudanças nas mais variadas interações entre o módulo e o núcleo, dessa forma foi necessário rever todo o código do ejournal para adequá-lo ao novo sistema.

Algumas funções tiveram seus argumentos alterados como as funções *url()*, *l()*, *hook_form_alter()*, *hook_link_alter()*, *hook_profile_alter()*, *hook_mail_alter()*, *watchdog()*, *cache_set()*, *hook_help()*, *drupal_mail()*, *hook_access()* e *hook_comment()*.

7 http://bazar.ginux.ufla.br/

As funções *url()*, *l()* e *hook_mail_alter()* recebiam uma lista com vários argumentos, dessa forma para alterar algum argumento que estivesse localizado no final da lista, era necessário identificar todos os argumentos anteriores, o que gerava muito tráfego de informação desnecessária como pode ser visto na Figura 28. No Drupal 6 estas funções passaram a receber uma matriz como argumento secundário, a qual só é preciso identificar os valores que precisam de alteração, conforme pode ser verificado na Figura 29.

```
url("user/$account->uid", NULL, NULL, TRUE);
```

Figura 28: Exemplo de chamada a url() no Drupal 5

```
url("user/$account->uid", array('absolute' => TRUE));
```

Figura 29: Exemplo de chamada a url() no Drupal 6

No gancho *hook_form_alter()* foi adicionado o estado do formulário e o identificador de formulário passou a ser o último argumento, já o gancho *hook_link_alter()* teve a ordem de seus argumentos trocados. Para o gancho *hook_profile_alter()* houve a remoção de um argumento, e as informações passaram a ser recebidas como objeto.

A função *cache_set()* teve seus parâmetros reordenados, pois um dos parâmetros que é opcional vinha antes de um parâmetro que é obrigatório o que obrigava o preenchimento do parâmetro opcional.

O módulo de registros do sistema de nome *watchdog* era muito comum em vários ambientes com o Drupal 5, então foi transformado em módulo do sistema e recebeu algumas alterações no seu tratamento para as mensagens.

Devido as alterações no modo de definir menus e caminhos o gancho para *hook_help()* também sofreu alterações, seus argumentos foram separados em caminho e argumentos adicionais, sendo que os argumentos adicionais são recebidos através de uma matriz.

A função para envio de e-mail, *drupal_mail()*, também foi alterada para seguir o padrão adotado nas demais, onde recebe-se uma matriz com os campos

adicionais. Essa mudança permitiu a separação entre a preparação da mensagem e o envio.

Para o gancho *hook_access()* foi adicionado o parâmetro conta como pode ser observado na Figura 30 onde se apresenta os parâmetros da chamada ao gancho, para evitar que módulos precisem requisitar a variável global *$user* para verificar o acesso do usuário atual.

```
function hook_access($op, $node, $account) {...}
```

Figura 30: Declaração da função hook_access()

Já o gancho *hook_comment()* teve uma operação removida, não sendo mais possível modificar elementos no formulário de comentário através do mesmo. A remoção teve como princípio a organização de gancho por função, e como já existe um gancho com a função de alterar formulários não há a necessidade de suportar a mesma operação em outro local.

As maiores alterações, no entanto, foram no sistema de menus, formulários e interação com termos de taxonomia. No sistema de menus foram modificados os argumentos da matriz que representa o menu, adicionados novos ganchos para alteração do menu e coringas para interação com dados dinâmicos. Um exemplo do que essas mudanças trouxeram pode ser observado na comparação da Figura 31, que apresenta a definição do menu para histórico de uma publicação no Drupal 5, com a Figura 32, que demonstra a mesma definição no Drupal 6. Nesta comparação é possível perceber claramente a separação entre a lógica e a definição de menu que foi conseguida com a nova implementação de menu.

```
    if (is_numeric(arg(1))) {
      if (arg(0) == 'node') {
        $items[] = array(
          'path'     => 'node/'. arg(1) .'/ejournal/history',
          'title'    => t('History'),
          'callback' => 'ejournal_article_history',
          'callback arguments' => array(arg(1)),
          'access'   => ejournal_user_access(array('chief editor',
'editor'), ejournal_journal_get_by_nid(arg(1))),
          'type'     => MENU_LOCAL_TASK,
          );
      }
    }
```

Figura 31: Definição do menu de histórico no Drupal 5

```
  $items['node/%/history'] = array(
    'title'            => t('History'),
    'page callback'    => 'ejournal_article_history',
    'page arguments'   => array(1),
    'access callback'  => 'ejournal_user_access',
    'access arguments' => array(array('chief editor', 'editor'),
ejournal_journal_get_by_nid(array(1))),
    'type' => MENU_LOCAL_TASK,
    );
```

Figura 32: Definição do menu de histórico no Drupal 6

O sistema de formulários passou a utilizar um argumento com o estado dos formulários em todos os passos de processamento, o que trouxe alterações em ordem e adição de parâmetros nos ganchos de validação, submissão e alteração de formulários. Algumas definições, como #submit, #validate e #process deixaram de permitir parâmetros customizados, por não serem mais necessários devido a possibilidade de utilizar chamadas a funções à partir de sua definição. E por fim, foram adicionadas novas funcionalidades de requisição assíncrona, conhecidas como Ajax.

O sistema de taxonomias passou a contar com o conhecimento do sistema de revisão, pode-se encontrar as alterações de termos em todas as alterações que determinado nó sofreu ao longo do tempo. Dessa forma as chamadas a funções de taxonomia devem receber não mais a identificação de nó, mas sim a identificação de versão do nó.

5.4 Discussão

Ao todo foram verificados todos os oitenta itens de conversão direta entre as versões do Drupal. Após as conversões diretas, começaram a ser realizadas as alterações de recursos.

Na tentativa de melhor utilizar os recursos disponíveis no Drupal, algumas das funções que eram realizadas por funções internas do ejournal foram substituídas por funções já disponíveis na nova versão do Drupal.

Todo o código foi corrigido quanto ao padrão adotado pelo Drupal, com correções de indentação, chamadas a funções, comentários e reorganização de atribuições para permitir uma melhor leitura do mesmo.

Com todas as alterações realizadas e adicionadas ao sistema de controle de versões oficial do Drupal, o módulo passou a ter suporte para o Drupal 6, que hoje ainda encontra-se em estágio alfa mas funcional.

Todas as alterações estão disponíveis na página do projeto através de *download* direto ou por acesso ao sistema de versões, é possível acompanhar cada passo realizado na alteração pelo ambiente de mensagens de versão[8], que conta com uma breve comentário sobre o que foi realizado e permite a comparação entre o código atual e o código anterior.

8 http://drupal.org/project/cvs/46485

6 Conclusão

Neste trabalho foram apresentados vários dos recursos disponíveis para utilização do Drupal como ferramenta de desenvolvimento para páginas de internet com funções de gerenciamento de conteúdo.

A inclusão de novas funcionalidades é possível através da criação de módulos. Estes permitem que o Drupal possa ser utilizado nas mais variadas funções de gerenciamento de informação. Através dos recursos apresentados tentou-se demonstrar toda esta flexibilidade e determinar até que ponto o Drupal oferece facilidades à inclusão de novos recursos.

Este trabalho também se preocupou em fornecer informações a respeito do padrão de desenvolvimento estabelecido e com isso oferecer uma referência e permitir uma melhor consistência de código entre módulos e os códigos que são oferecidos no núcleo do Drupal.

Pode-se perceber que o ambiente de desenvolvimento para Drupal está maduro e oferece recursos suficientes para os mais variados usos, sendo recomendável para quem pretende criar novas páginas de internet ou atualizar sistemas que não estejam oferecendo recursos suficientes.

Com a utilização das informações apresentadas foi possível migrar o módulo ejournal da versão 5 do Drupal para a versão 6. Também foi possível adicionar melhorias internas que permitiram ao módulo ser mais integrado ao sistema e oferecer uma maior facilidade de expansão e continuidade de codificação.

Este trabalho oferece várias possibilidades de trabalhos futuros, principalmente com relação a criação de novos módulos para Drupal que permitam novas interações com as informações.

Um dos aspectos que não foram abordados neste trabalho incluem a criação de temas para Drupal. Estes utilizam uma parte da programação apresentada, mas com foco em apresentação de conteúdo.

7 Referências

Api Reference | Drupal API. [on-line]. Disponível na Internet via www. Url: http://api.drupal.org. Arquivo capturado em 10 de janeiro de 2009.

BUTCHER, Matt. *Learning Drupal 6 Module Development*. Birmingham: Packt Publishing, 2008. 310p.

CONVISSOR, Daniel, JANSEN, Martin, MERZ, Alexander. *PEAR Manual*. [on-line]. Disponível na Internet via www. Url: http://pear.php.net/distributions/ manual/ pear_manual_en.html.gz. Arquivo capturado em 08 de janeiro de 2009.

HEESCH, Dimitri van. Doxygen. [on-line]. Disponível na internet via www. http://www.stack.nl/~dimitri/doxygen/. Arquivo capturado em 12 de fevereiro de 2009.

LEE, Wilson. *Core hooks cheat sheets*. [on-line]. Disponível na Internet via www. Url: http://drupal.org/files/drupal_6_core_hooks_cheat_sheet.pdf. Arquivo capturado em 06 de março de 2009.

MERCER, David. *Building powerful and robust websites with Drupal 6*. Birmingham: Packt Publishing, 2008. 362p.

SANTOS, Nei Rauni. Desenvolvimento de sites colaborativos utilizando o CMS Drupal. 2008. 61 f. TCC (Tecnólogo em Processamento de Dados) - Escola Superior de Estudos Empresariais e Informática, Faculdades ESEEI, Curitiba.

VANDYK, John K. *Pro Drupal Development, Second Edition*. Berkley: Apress, 2008. 667p.