Framework MIOLO 2.5

Vilson Cristiano Gärtner
vilson@miolo.org.br

Ely Edison Matos
ely.matos@ufjf.edu.br

versão do documento: 1.0

20/03/2009

Índice

1. Framework MIOLO

O MIOLO é um framework para criação de sistemas de informação acessíveis via WEB, baseado na linguagem PHP5, scripts javascript e conceitos de POO (Programação Orientada a Objetos), gerando páginas HTML. Um framework é um conjunto de classes cooperantes que constroem um projeto reutilizável para uma classe específica de software . O framework dita a arquitetura da aplicação . A aplicação é construída criando subclasses das classes do framework . Assim, o framework enfatiza a reutilização de projetos .

Como o MIOLO é o "kernel" de todos os sistemas criados, os mesmos podem ser facilmente integrados, funcionando como módulos de um sistema mais complexo. Além de proporcionar as funcionalidades para o desenvolvimento de sistemas, o MIOLO também define e implementa toda uma sistemática e uma metodologia para que os resultados esperados sejam obtidos de forma simples.

O pré-requisito para utilizar o MIOLO é ter conhecimento de programação com PHP5 e de POO. Para criar sistemas utilizando o MIOLO é necessário conhecer algumas regras básicas, entre elas, as definições de separação das classes (classes-forms-handlers) dos sistemas/módulos, configuração (localização dos arquivos de programas e do MIOLO, BD, temas,...), o ciclo de vida de execução de uma requisição do cliente, além das principais classes, métodos e controles. Este guia oferece uma visão geral sobre estes tópicos.

Algumas das principais funções implementadas pelo framework são:

- Arquitetura em camadas
- Possibilita o padrão MVC (Model-View-Controller)
- Rico conjunto de componentes UI , usando PHP5 e Javascript (DOJO)
- Modelo de programação event-driven , com forte uso de AJAX
- Gerenciamento de sessão e estado
- Aplicações cross-browser
- Segurança (autenticação, permissões, logs)
- Abstração de acesso a Banco de Dados
- Camada de persistência de objetos : MOQL (Miolo Object Query Language)
- Customização da UI através de temas e templates
- Geração de arquivos PDF (EzPDF, JasperReports)

2. Estrutura da Aplicação

Arquitetura

O MIOLO adota a arquitetura em camadas (figura 1).

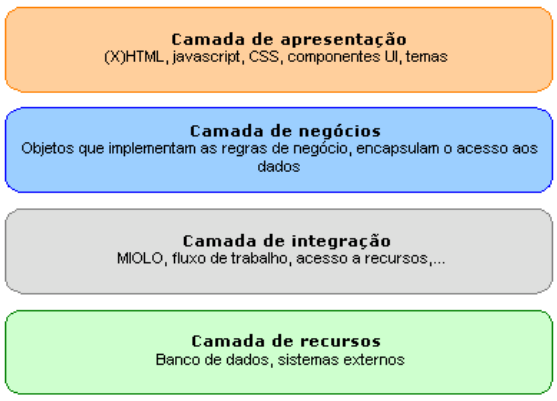

Figura 1 – Arquitetura em Camadas

Camada de apresentação

São as classes do framework responsáveis pela geração de arquivos, renderização dos controles HTML e criação dos scripts javascript enviados ao cliente, com base no tema em uso. Engloba também as classes criadas

pelos usuários para definir a interface da aplicação (geralmente nos diretórios forms, menus e reports de cada módulo).

Camada da lógica de negócio (domínio)

São as classes criadas pelo desenvolvedor para representar o domínio da aplicação (as regras de negócio). São usadas pela camada de apresentação e nos handlers, acessando o banco de dados através da camada de recursos.

Camada de integração

Classe MIOLO: representa o framework, expondo métodos que fazem a integração entre as diversas camadas. Implementa o padrão Facade.

Handlers: São as classes que representam a parte funcional da aplicação, criadas pelo desenvolvedor para fazer o tratamento dos dados enviados pelo cliente. Acessa a camada de negócios para desempenhar suas funções e usa a camada de apresentação para definir a saída para o cliente. Estão localizadas no diretório handlers de cada módulo.

Util e Services: São as classes do framework responsáveis por oferecer recursos e funcionalidades utilizadas tanto pelo framework quanto pelas aplicações dos usuários, encapsulando o acesso a recursos da linguagem ou do sistema operacional.

Camada de recursos

São as classes do framework responsáveis por abstrair o acesso às bases de dados, tornando as classes da camada de domínio independentes do SGBD usado.

A figura 2 mostra a organização destas camadas na estrutura do framework.

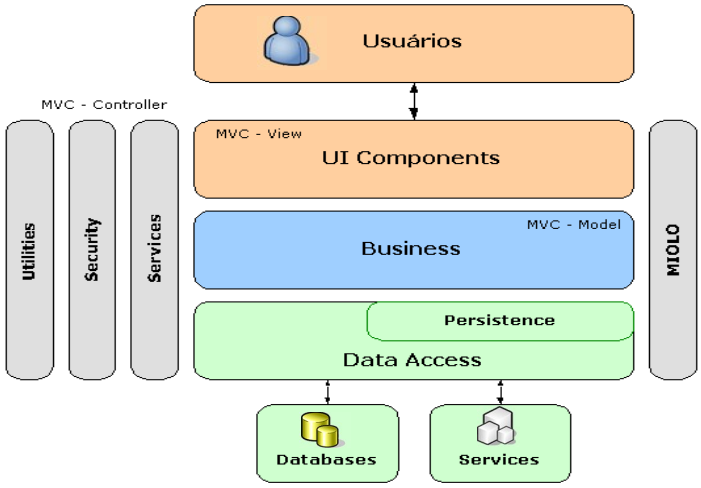

Figura 2 – Implementação das camadas

Conceitos básicos

Aplicação

O framework MIOLO tem por objetivo a construção de sistemas de informação baseados em web, oferecendo a infra-estrutura necessária para que o desenvolvedor se preocupe apenas com o domínio da aplicação e não com os detalhes de implementação. Estes sistemas são construídos através do desenvolvimento de módulos. O conjunto de módulos é chamado **aplicação**. Assim, de forma geral, cada instalação do framework está associada a uma única aplicação, composta por um ou vários módulos integrados.

Todos os arquivos do framework ficam sob um mesmo diretório (ex: /usr/local/miolo) e cada instalação do framework possui um arquivo de configuração (em <miolo>/etc/miolo.conf).

A aplicação implementa o padrão Front Controller (fornecendo um único ponto de entrada para a aplicação), sendo acessada através do arquivo <miolo>/html/index.html, que inclui o handler <miolo>/html/index.php.

Módulo

Um módulo é parte de uma aplicação. Ele reflete um sub-domínio da aplicação, agrega as classes de negócio que estão fortemente relacionadas e provê o fluxo de execução (através de handlers) e a interface com o usuário (forms, grids, reports) para se trabalhar com as classes de negócio .

Um módulo é caracterizado por um nome, usado como subdiretório do diretório <miolo>/modules. Cada módulo tem uma estrutura padrão de diretórios, usada pelo framework para localizar os recursos, e possui seu próprio arquivo de configuração (<miolo>/modules/<modulo>/etc/module.conf).

Controles (Widgets)

Os controles são componentes de interface com o usuário, usados na renderização das páginas HTML . Um controle pode agregar outros controles , possui propriedades e eventos associados e são implementados em PHP5 e/ou Javascript .

Página

A página é um controle específico (instanciado da classe MPage) que serve de base para a renderização de uma página HTML.

Tema

Um tema é um controle específico (instanciado da classe MTheme) que trabalha como um container para os controles que vão ser renderizados na página HTML.

Handler

Um handler é um objeto da classe MHandler . Sua função é tratar a solicitação feita pelo usuário através do browser . Todas as solicitações são direcionadas a um handler .

Em cada módulo é definida uma classe Handler<Modulo>, instanciada pelo MIOLO quando é feita a análise da solicitação do usuário (<miolo>/modules/<modulo>/handlers/handler.class.php). O controle é passado para esta classe, que inclui o código específico para tratar a solicitação. O código de cada handler é armazenado em <nome_handler>.inc.php .

Namespace

Namespaces são apenas "apelidos" para diretórios . O objetivo do uso de namespaces é a possibilidade de mudança da localização física dos arquivos, sem a necessidade de se alterar o código já escrito. Os namespaces são usados basicamente no processo de importação (include) de arquivos, em tempo de execução . A configuração dos namespaces fica no arquivo de configuração miolo.conf .

Exemplo : modules::admin::business::user

Estrutura de diretórios

```
-----<diretório-base>(p.ex. /usr/local/miolo25)
         |
         +--- classes
         |         +--- contrib
         |         +--- database
         |         +--- doc
         |         +--- etc
         |         +--- extensions
         |         +--- ezpdf
         |         +--- flow
         |         +--- model
         |         +--- persistence
         |         +--- pslib
         |         +--- security
         |         +--- services
         |         +--- ui
         |         +----+--- controls
         |         +----+--- painter
         |         +----+--- report
         |         +----+--- themes
         |         |                +--- miolo
         |         |          +--- kenobi
         |         |          +--- clean
         |         |          +--- .....
         |         +--- utils
```

```
|
+--- docs
+--- etc
|        +--- miolo.conf
|        +--- mkrono.conf
+--- html
|        +--- downloads
|        +--- images
|        +--- reports
|        +--- scripts
+--- locale
+--- modules
|        +--- admin
|        +--- modulo1
|        +--- modulo2
|        +--- ....
+--- var
         +--- db
         +--- log
         +--- report
         +--- trace
```

- classes **–** contém as classes que formam o kernel do MIOLO
- classes/contrib – classes de terceiros, que podem ser usadas no framework.
- classes/database – classes que implementam o acesso a banco de dados (a camada DAO – Data Acess Objects).
- classes/doc – classes para geração da documentação.
- classes/extensions – classes que estendem a funcionalidade do framework, por herança ou composição de componentes existentes, mas que não fazem ainda parte do "core" do Miolo.
- classes/etc – arquivos auxiliares, como o autoload.xml que define a localização dos arquivos que implementam as classes.
- classes/ezpdf – classes da biblioteca ezPDF, para geração de arquivos PDF.
- classes/flow – classes relacionadas ao fluxo de execução de uma requisição.
- classes/model – classes relacionadas à camada Business.
- classes/persistence – classes que implementam o mecanismo de persistência de objetos em bancos de dados.
- classes/pslib – classes utilizadas para geração de arquivos PostScript.
- classes/security – classes relacionadas às tarefas de segurança (autenticação, autorização, criptografia, etc).
- classes/services – classes utilitárias e de serviços gerais.
- classes/ui – classes relacionadas à interface com o usuário (controles, renderização html, relatórios em pdf).
- classes/util – classes utilitárias.
- modules – contém um subdiretório para cada módulo do sistema. Cada módulo possui uma estrutura de diretórios pré-definida.
- var/db – contém um banco de dados Sqlite para armazenamento de dados relativos à execução das aplicações.
- var/log – contém os arquivos de logs gerados pelo MIOLO e pelos sistemas.
- var/report – contém os arquivos PDF gerados pelas rotinas de reports.
- var/trace – contém os arquivos usados no processo de debug da aplicação.
- locale – contém o sistema usado para internacionalização.
- etc/miolo.conf – arquivo principal de configuração do MIOLO.
- html – contém as páginas do sistema, bem como os subdiretórios para imagens, scripts e temas. Deve ser o único diretório visível via Web.
- docs – textos de documentação.

Arquivos principais

<miolo>/html/index.html – É o arquivo acessado pelo servidor web e que inicia o processo de criação do ambiente para o sistema. Neste arquivo é criado um frameset com um frame ("content") utilizado para criação do conteúdo das páginas do sistema propriamente dito (visíveis para o usuário). Neste frame "content" é chamado o arquivo index.php. O objetivo do uso de frames é evitar que as urls usadas pelo framework sejam exibidas no browser.

<miolo>/html/index.php - O arquivo index.php (que pode ter um nome diferente, caso a configuração em miolo.conf seja modificada) é o manipulador principal do MIOLO, utilizado por todos os módulos e sempre definido

nos links que são criados pelos menus, formulários e funções de criação automáticas de links. A principal função do arquivo index.php é instanciar um objeto MIOLO (que é a classe principal do framework, atuando como uma fachada) e executar o método HandlerRequest, que vai tratar a solicitação do usuário (feita via browser).

<miolo>/etc/miolo.conf - Arquivo no formato XML que mantém as configurações do ambiente.

<miolo>/classes/support.inc – Neste arquivo estão definidas as funções globais do framework.

<miolo>/html/scripts/m_*.js - Esses arquivos contêm as principais funções javascript utilizadas pelos componentes e pelo framework.

<miolo>/classes/miolo.class.php - Essa é a principal classe do MIOLO, implementada com o padrão Singleton. Contém os principais métodos do framework, atuando como uma fachada (padrão façade) para acesso aos serviços implementados nas demais classes.

URL

A URL padrão do Miolo está estruturada da seguinte forma:

```
http://host.dominio/index.php?module=<module>&action=<action>[& lista de parâmetros]
```

Esta estrutura é generalizada para:

A - Executar uma ação (handler)

```
Ex: http://host.dominio/index.php?module=common&action=main:login
```

- host.dominio: é o nome de domínio do site
- index.php: o manipulador principal do miolo
- module=<módulo>: nome do módulo a ser usado
- action=<ação>: string no formato "handler1:handler2:...:handlerN", onde cada handler indica o arquivo que será chamado dentro do módulo, na seqüência estabelecida
- item=<item>: variável auxiliar que pode ser usada no processamento da página
- outras variáveis: criadas pela aplicação e repassadas via url para auxiliar na manipulação da página

B – Acessar arquivos armazenados dentro dos módulos (dentro do diretório html)

```
-   Imagens: http://host.dominio/index.php?module=common&action=images:save.png
-   PDF: http://host.dominio/index.php?module=common&action=files:exemplo.pdf
-   Texto: http://host.dominio/index.php?module=common&action=files:exemplo.txt
-   CSS: http://host.dominio/index.php?module=example&action=themes:blue:miolo.css
```

- host.dominio: é o nome de domínio do site
- index.php: o manipulador principal do miolo
- module=<módulo>: nome do módulo a ser usado
- action=namespace: string que indica a localização do arquivo, com base no namespace

Obs: o namespace é relativo ao diretório html dentro do módulo.

Além disso, está implementada a escrita de URLs amigáveis:

```
Ex: http://host.dominio/index.php/common/images/save.png
```

Fluxo de Execução

O fluxo básico de execução a partir de uma solicitação do usuário, feita via browser, é apresentado na figura 3.

Quando a página index.php é executada, ela instancia a classe MIOLO (permitindo acesso às funções principais do framework) e executa o método MIOLO::HandlerRequest. Este método analisa a URL para verificar se está sendo solicitado um arquivo ou a execução de um handler. Caso seja um arquivo, este é localizado e enviado para o browser. Caso seja solicitada a execução de um handler, o arquivo support.inc é incluído (com as funções

globais), é feita a inicialização das propriedades do objeto MIOLO, a inicialização do tratamento da sessão do usuário, a verificação das informações de login (caso existam), a obtenção do tema a ser renderizado, a inicialização da página e finalmente é chamado o método MIOLO::InvokeHandler.

O método InvokeHandler chama o handler "main" do modulo admin (ou o que for indicado em miolo.conf em <options><startup>). Este por sua vez é responsável por executar o handler seguinte na seqüência de ações solicitadas. Após o último handler ser executado, o estado das variáveis é salvo e é feita a renderização da página.

O método InvokeHandler executa o método Dispatch da classe Handler do módulo. Neste processo são definidas variáveis globais, que poderão ser acessadas pelo handler. De forma geral, um handler fará a chamada ao handler seguinte, afim de que sejam processados os demais handlers da "action" passada como parâmetro na url, até que todos os handlers da seqüência tenham sido executados. É importante ressaltar que, por default, o handler "main" do módulo startup (definido em miolo.conf) sempre será executado.

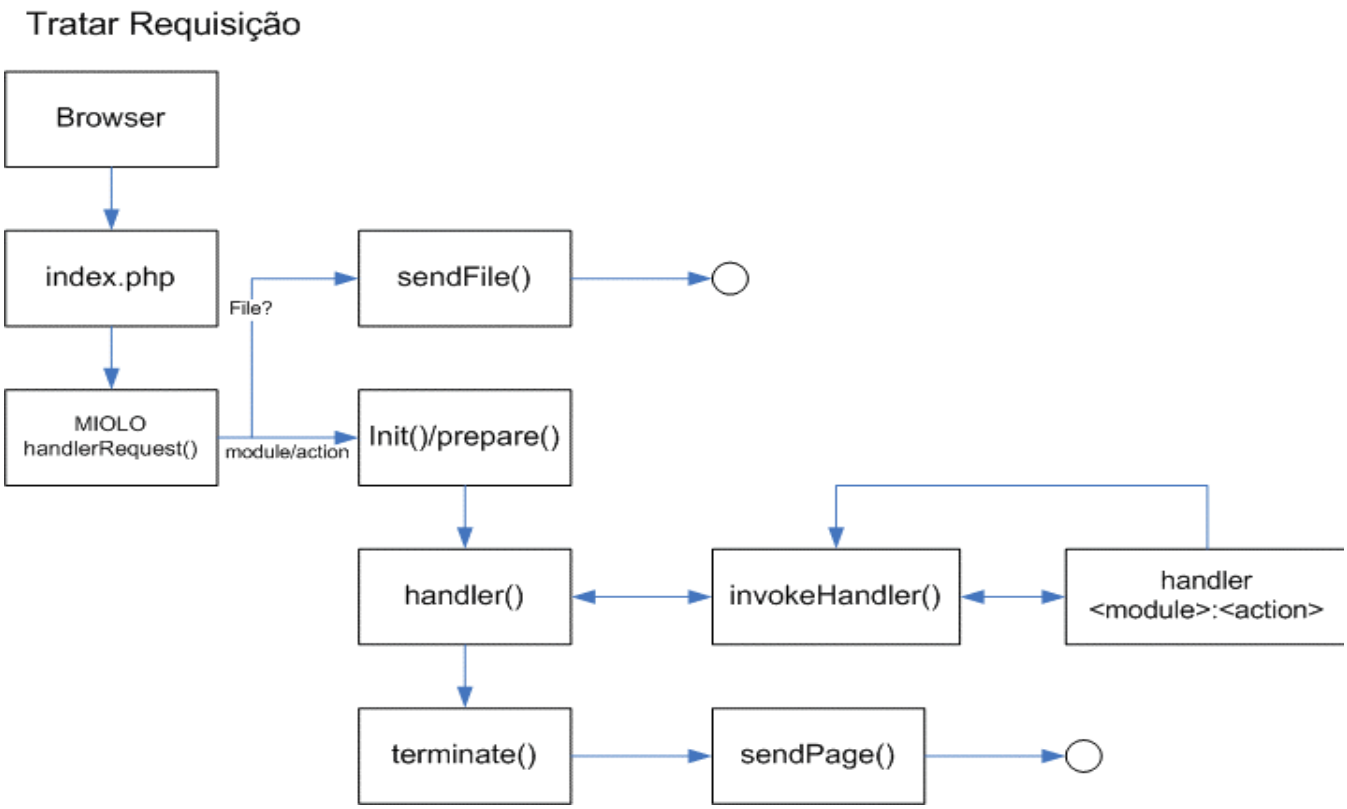

Figura 3 – Fluxo de Execução

Módulos

No MIOLO, todos os módulos ficam localizados abaixo do diretório "modules". Cada módulo ou sistema, para possibilitar o total reaproveitamento de código (seja de formulários, menus ou instruções SQL), deve separar essas informações, colocando-os em seus respectivos diretórios. A estrutura de diretórios obedece à seguinte regra:

```
-----<diretório-base>(p.ex. /usr/local/miolo2)
       |
       +--- modules
       |       +--- module1
       |       |      +--- classes
       |       |      +--- forms
       |       |      +--- menus
       |       |      +--- sql
       |       |      +--- handlers
       |       |      +--- grids
       |       |      +--- reports
       |       |      +--- etc
       |       |      +--- html
       |       |      |     +--- images
```

```
|       |       |      +--- files
|       |       +--- ui
|       |              +--- controls
|       |
|       +--- module2
|       |
|       +--- module3
```

Variáveis globais

As seguintes variáveis são definidas como globais, sendo disponibilizadas para todos os handlers:

- $**MIOLO**: acesso a instancia da classe principal do framework
- $**page**: acesso ao objeto MPage
- $**context**: acesso ao objeto Mcontext
- **$theme**: acesso ao objeto Mtheme
- $**auth**: acesso ao objet MAuth
- $**perms**: acesso ao objeto Mperms
- $**state**: acesso ao objeto Mstate
- $**log**: acesso ao objeto Mlog
- $**navbar**: acesso ao objeto Navigation
- $**module**: nome do módulo do handler em execução (ex: 'admin')
- $**action**: path do handler em execução
- $**item**: campo item da url atual
- $**url**: url completa do handler da ação sendo executada

3. Camada de Apresentação

A camada de aplicação do Miolo (versão 2.5) utiliza o framework DOJO de forma integrada. Praticamente toda a comunicação entre o browser e o servidor é feito através de chamadas AJAX.

A aplicação criada com o Miolo é uma "single-page application" (SPA). Na primeira chamada à aplicação (ou quando é chamado um endereço diretamente no browser) é criada uma "página principal" ou "página base" (template base.php – v. Temas), que vai servir de base para a renderização dos conteúdos obtidos via Ajax. Na filosofia de temas do Miolo, uma página é dividida em "elementos do tema" (ThemeElement - representados por elementos). A cada chamada Ajax, será obtido como retorno o conteúdo HTML que deve ser renderizado em cada elemento do tema.

Uma página HTML poderá conter vários forms. O form (elemento HTML <form>) é renderizado dentro de um elemento do tema ( ). Cada form agrupa os controles cujos conteúdos serão enviados juntos em uma submissão. Os diálogos (janelas) são também representados por <form> inseridos em .

O form principal é chamado "__mainForm" (representado por `<div id="__mainForm">`). O objetivo deste __mainForm é servir de base para a renderização dos controles enviados pelo servidor. Dentro deste __mainForm são renderizados os vários elementos do tema (top, menus, navigation, content, etc).

A "single-page", base da aplicação, tem o seguinte formato:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<title>Miolo Framework</title>

<link rel="stylesheet" type="text/css" href="http://127.0.0.1/themes/blue/dojo.css">
<link rel="stylesheet" type="text/css" href="http://127.0.0.1/themes/blue/miolo.css">

<meta http-equiv="Content-Type" content="0">
<meta name="Generator" content="MIOLO Version Miolo 2.5; http://www.miolo.org.br">
<script type="text/javascript"> djConfig={usePlainJson:true,  parseOnLoad:true}</script>
<script type="text/javascript" src="http://127.0.0.1/scripts/dojoroot/dojo/dojo.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/prototype/prototype.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_miolo.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_page.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_ajax.js"></script>
```

```
<script type="text/javascript" src="http://127.0.0.1/scripts/m_encoding.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_form.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_compatibility.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_md5.js"></script>
<script type="text/javascript" src=" http://127.0.0.1/scripts/jscookmenu/jscookmenu.js"></script>
<script type="text/javascript" src=" http://127.0.0.1//scripts/jscookmenu/jscookmenu_office.js"></script>


<script type="text/javascript">
dojo.require("dojo.parser");
dojo.require("miolo.Dialog");
dojo.require("dijit.form.DateTextBox");
dojo.require("dojox.layout.ContentPane");

//-->
</script>

<script type="text/javascript">
function init()
{
       miolo.doHandler("<?php echo $action ?>",'__mainForm');
}
dojo.addOnLoad(init);
//-->
</script>

</head>
<body class="mThemeBody">
<!-- begin of page -->
<div id="page49bff27a56937">

        <div id="stdout" class="mStdOut" dojoType="dojox.layout.ContentPane"></div>

        <!-- begin of form __mainForm -->
        <div id="__mainForm__scripts" dojoType="dojox.layout.ContentPane" layoutAlign="client"
executeScripts="true" cleanContent="true">
        </div>
        <div id="__mainForm" dojoType="dojox.layout.ContentPane" layoutAlign="client"
executeScripts="true" cleanContent="true">
</div>
<!-- end of form __mainForm -->
</div>
<!-- end of page -->
</body>
</html>
```

Nesta página, temos:

- Folha de Estilos por tema: para controles do Dojo (dojo.css) e do Miolo (miolo.css)
- Scripts: scripts básicos do MIOLO, usados por vários forms, mais scripts básicos do framework DOJO
- O corpo (body) da página HTML é constituído de um único div, que representa a página (renderizada pelo objeto MPage). Em uma mesma página é possível agregar vários forms (objetos MForm ou MWindow).

O elemento principal da aplicação (onde será incluído o código HTML gerado pela execução de um handler) é o . A inclusão do código HTML é feita através de uma chamada Ajax que executa o handler (miolo.doHandler). O é usado para executar o código javascript que foi gerado no lado do servidor; esse código é executado após o carregamento do conteúdo HTML no div __mainForm. O é usado para exibição de mensagens de erro gerados pelo PHP, para saída dos comandos var_dump() e echo e para exibição de erros no processamento do javascript no lado cliente.

Uma vez que tenhamos o código HTML carregado em __mainForm, todas as chamadas seguintes devem ser feitas via Ajax, utilizando os métodos da classe javascript Miolo (em m_miolo.js). Isto não impede, porém que sejam feitas chamadas usando GET, o que irá provocar uma nova renderização da "página principal", repetindo o processo descrito acima.

Controles (widgets)

Os controles visuais são classes programadas em PHP5 e que encapsulam controles HTML (ou controles construídos em javascript). Os controles no Miolo estão organizados em uma hierarquia. Os controles devem encapsular a lógica do controle (em PHP), o código javascript associado (se houver), eventos associados e indicar as classes CSS usadas na renderização.

A renderização do controle é feita através do método generate(). O método generateInner() é usado para gerar o código HTML referente ao controle; os controles que herdam de Mdiv são renderizados dentro de uma box (tag). A renderização está encapsulada na classe MHTMLPainter (<miolo>/classes/ui/painter). Cada método desta classe recebe um objeto e gera o código HTML correspondente.

"Controles Atômicos" são aqueles renderizados diretamente através de uma tag HTML. A renderização de controles atômicos é feita através da classe MHtmlPainter – deve-se evitar escrever qualquer código HTML dentro da lógica do controle, bem como qualquer estilo que possa ser definido via CSS. Como regra geral, um controle não-atômico é formado pela composição de controles atômicos.

Atributos CSS podem ser propriedades do objeto referente ao controle (atribuídas diretamente ao objeto) ou podem ser definidos via método addStyle().

Cada controle é definido em um arquivo próprio, em <miolo>/classes/ui/controls. Os usuários também podem construir seus próprios controles com base nos já existentes, através de herança.

Events

Os eventos javascript (click, mouseover, etc) e as chamadas AJAX associadas ao controle podem ser atribuídos via método addEvent(evento, handler, preventDefault) ou através de addAttribute(evento, "javascript:<codigo>") . Todos os eventos adicionados com addEvent() devem terminar com ';' .

Controls Tree

```
+--MStyle
+--MAttributes
+--MdragDropControl
+--+--MDraggable
+--+--MDroppable
+--MComponent
+--+--MControl
+--+--+--MPage
+--+--+--MDiv
+--+--+--+--MHr
+--+--+--+--MCSSBox
+--+--+--+--MBoxTitle
+--+--+--+--MModuleHeader
+--+--+--+--MEditor
+--+--+--+--MSeparator
+--+--+--+--MSpacer
+--+--+--+--MContainerControl
+--+--+--+--+--MBox
+--+--+--+--+--MAccordion
+--+--+--+--+--MContainer
+--+--+--+--+--+--MHContainer
+--+--+--+--+--+--MVContainer
+--+--+--+--+--+--MBasePanel
+--+--+--+--+--+--+--MPanel
+--+--+--+--+--+--+--+--MActionPanel
+--+--+--+--+--+--MBaseGroup
+--+--+--+--+--+--+--MRadioButtonGroup
+--+--+--+--+--+--+--MCheckBoxGroup
+--+--+--+--+--+--+--MLinkButtonGroup
+--+--+--+--+--+--+--MToolBar
+--+--+--+--+--MTabContainer
+--+--+--+--+--MBaseGrid
+--+--+--+--+--+--MGrid
+--+--+--+--+--+--+--MDataGrid
+--+--+--+--+--+--+--+--MDataGrid2
+--+--+--+--+--+--+--MLookupGrid
+--+--+--+--+--+--+--+--MLookupObjectGrid
+--+--+--+--+--+--+--+--MLookupQueryGrid
+--+--+--+--+--+--+--MObjectGrid
+--+--+--+--+--+--+--MPDFReport
+--+--+--+--+--+--+--MGridAjax
+--+--+--+--+--MBaseForm
+--+--+--+--+--+--MForm
+--+--+--+--+--+--+--MFormAjax
+--+--+--+--+--+--+--MCompoundForm
+--+--+--+--+--+--+--MCSSForm
+--+--+--+--+--+--+--MCSSPForm
```

```
+--+--+--+--+--+--+--MIndexedForm
+--+--+--+--+--+--+--MTabbedForm
+--+--+--+--+--MTheme
+--+--+--+--+--MThemeElement
+--+--+--+--+--MThemeBox
+--+--+--MTabbedFormPage
+--+--+--MTableCell
+--+--+--MTableRow
+--+--+--MTable
+--+--+--+--MSimpleTable
+--+--+--+--+--MTableRaw
+--+--+--+--+--+--MTableXML
+--+--+--MContent
+--+--+--+--MFileContent
+--+--+--MContentHeader
+--+--+--MAreaContainer
+--+--+--MIFrame
+--+--+--MImageFormLabel
+--+--+--MGridColumn
+--+--+--+--MGridHyperlink
+--+--+--+--+--MDataGridHyperlink
+--+--+--+--+--MObjectGridHyperlink
+--+--+--+--MGridControl
+--+--+--+--+--MDataGridControl
+--+--+--+--+--MObjectGridControl
+--+--+--+--+--MPDFReportControl
+--+--+--+--MDataGridColumn
+--+--+--+--MObjectGridColumn
+--+--+--+--MPDFReportColumn
+--+--+--MGridAction
+--+--+--+--MGridActionIcon
+--+--+--+--MGridActionText
+--+--+--+--MGridActionSelect
+--+--+--+--MDataGridAction
+--+--+--+--MObjectGridAction
+--+--+--MGridFilter
+--+--+--+--MGridFilterText
+--+--+--+--MGridFilterSelection
+--+--+--+--MGridFilterControl
+--+--+--MGridNavigator
+--+--+--MJavascriptCode
+--+--+--MFormControl
+--+--+--+--MInputControl
+--+--+--+--+--MActionControl
+--+--+--+--+--+--MButton
+--+--+--+--+--+--+--MButtonAjax
+--+--+--+--+--+--+--MInputButton
+--+--+--+--+--+--+--+--MButtonFind
+--+--+--+--+--+--+--MButtonImage
+--+--+--+--+--+--+--MButtonWindow
+--+--+--+--+--+--+--MDropDownButton
+--+--+--+--+--+--MLink
+--+--+--+--+--+--+--MActionHyperLink
+--+--+--+--+--+--+--MLinkBack
+--+--+--+--+--+--+--MOpenWindow
+--+--+--+--+--+--+--MGridHeaderLink
+--+--+--+--+--+--+--MLinkButton
+--+--+--+--+--+--+--+--MButtonClose
+--+--+--+--+--+--+--+--MButtonHelp
+--+--+--+--+--+--+--+--MButtonMinimize
+--+--+--+--+--+--+--MImageLink
+--+--+--+--+--+--+--+--MImageLinkLabel
+--+--+--+--+--+--+--+--MImageButton
+--+--+--+--+--+--+--+--+--MImageButtonLabel
+--+--+--+--+--MTextField
+--+--+--+--+--+--MCalendarField
+--+--+--+--+--+--MCurrencyField
+--+--+--+--+--+--MPasswordField
+--+--+--+--+--+--MFileField
+--+--+--+--+--+--MHiddenField
+--+--+--+--+--+--MMultiLineField
+--+--+--+--+--+--+--MmultiTextField3
+--+--+--+--+--+--+--+--MmultiTextField4
+--+--+--+--+--+--+--MmultiTextField2
+--+--+--+--+--+--MLookupField
+--+--+--+--+--+--+--MLookupTextField
+--+--+--+--+--+--+--MLookupFieldValue
+--+--+--+--+--MChoiceControl
+--+--+--+--+--+--MCheckBox
+--+--+--+--+--+--MRadioButton
+--+--+--+--+--MListControl
+--+--+--+--+--+--MSelection
+--+--+--+--+--+--+--MComboBox
+--+--+--+--+--+--MMultiSelection
+--+--+--+--+--+--MMultiSelectionField
```

```
+--+--+--+--+--+--MOrderedList
+--+--+--+--+--+--MUnOrderedList
+--+--+--+--+--MOption
+--+--+--+--+--MInputGridColumn
+--+--+--+--+--MInputGrid
+--+--+--+--+--MPrompt
+--+--+--+--+--MIndexedControl
+--+--+--+--MOutputControl
+--+--+--+--+--MSpan
+--+--+--+--+--+--MHint
+--+--+--+--+--MBaseLabel
+--+--+--+--+--+--MPageComment
+--+--+--+--+--+--MSeparator
+--+--+--+--+--+--MLabel
+--+--+--+--+--+--+--MRawText
+--+--+--+--+--+--MFieldLabel
+--+--+--+--+--+--MTextHeader
+--+--+--+--+--+--MText
+--+--+--+--+--+--+--MTextLabel
+--+--+--+--+--+--+--MGDText
+--+--+--+--MImage
+--+--+--+--+--MImageLabel
+--+--+--+--MDragDrop
+--+--+--+--MHtmlForm
+--+--+--+--MTextTable
+--+--+--+--MtoolBarButton
+--+--+--+--MTreeMenu
+--+--+--MValidator
+--+--+--MWindow
+--+--+--MStatusBar
+--+--+--MOptionGroup
+--+--+--MOptionListItem
+--+--+--MOptionList
+--+--+--+--MDhtmlMenu2
+--+--+--+--MMenu
+--+--+--+--+--MNavigationBar
```

AJAX

A implementação do AJAX no Miolo 2.5 usa o framework DOJO e a biblioteca CPAINT .

De forma geral, pode-se usar a string ':nome_metodo' no lugar de uma URL. Em tempo de execução é feito um POST para a URL corrente e executado o método "nome_metodo", que recebe como parâmetro um objeto com os valores dos campos do formulário . Exemplo:

```
new MButton('btnPost','Send',':doSomething');
```

Parâmetros específicos podem ser passados para o método, usando ":nome_metodo;par1;par2" .

A maior parte da interação entre o browser e o webserver é feita via AJAX . A primeira página é gerada via chamada GET, o form HTML principal (__mainForm) é gerado e a partir daí são usados os métodos Javascript (m_miolo.js):

- doPostBack(eventTarget, eventArgument, formSubmit) : simula o submit (POST) da página.
- doLinkButton(url, eventTarget, eventArgument, formSubmit) : simula o submit (POST) da página para o handler indicado por url.
- doAjax(eventTarget, eventArgument, formSubmit) : faz uma chamada Ajax.
- doHandler(url, formSubmit) : simula uma chamada GET para URL.
- doLink(url, formSubmit) : simula uma chamada GET, alterando o action do form para URL.
- doRedirect(url, element) : simula um HTTP Redirect

Do lado do servidor duas classes apóiam o uso do AJAX: MPage e Majax .

MPage é usada para estruturar a renderização do código a ser enviado ao browser. Os métodos associados ao uso de AJAX são:

- generateForm : renderiza o conteúdo de __mainForm ou de windows
- generateBase: gera a página principal, ou quando é feita uma chamada via GET (sem AJAX)
- generateAjax : responde a uma chamada AJAX (com preenchimento de um ou vários elementos)
- setElementValue($element, $value) : atribui o valor de $element usando Javascript
- copyElementValue($element1, $element2): copia o valor de $element1 para $element2 usando Javascript

MAjax encapsula a biblioteca Ajax no lado servidor . Geralmente acessada através de

```
$this->manager->ajax
```

Os principais métodos da classe MAjax são:

- setResponseControls($controls,$elements='') : $controls e $elements definem, respectivamente, o controle (ou array de controles) que serão exibidos e o id (ou array de ids) dos elementos que receberão o código HTML gerado através dos controles.
- setResponseScripts($scripts=array()) : define o conteúdo do array response->scripts.
- setResponse($controls,$elements='',$scripts=array()) : define simultaneamente os controles, os elementos e os scripts que serão enviados como retorno de uma chamada ajax.

Tema

Denominamos "tema", no ambiente MIOLO, à definição do layout da página html que será enviada para o cliente. O tema pode ser considerado como um container para os controles que serão renderizados, sendo composto por diversos elementos (título, barra de navegação, menus, área de conteúdo, barra de status). Cada elemento do tema é um controle da classe MThemeElement. A definição e manipulação do tema são sustentadas por algumas classes internas ao framework MIOLO. O tema portanto deve definir não apenas como a página será "dividida", mas também como os controles html serão renderizados.

Como exemplo, a estrutura do tema "blue", distribuído junto com o MIOLO, é a seguinte:

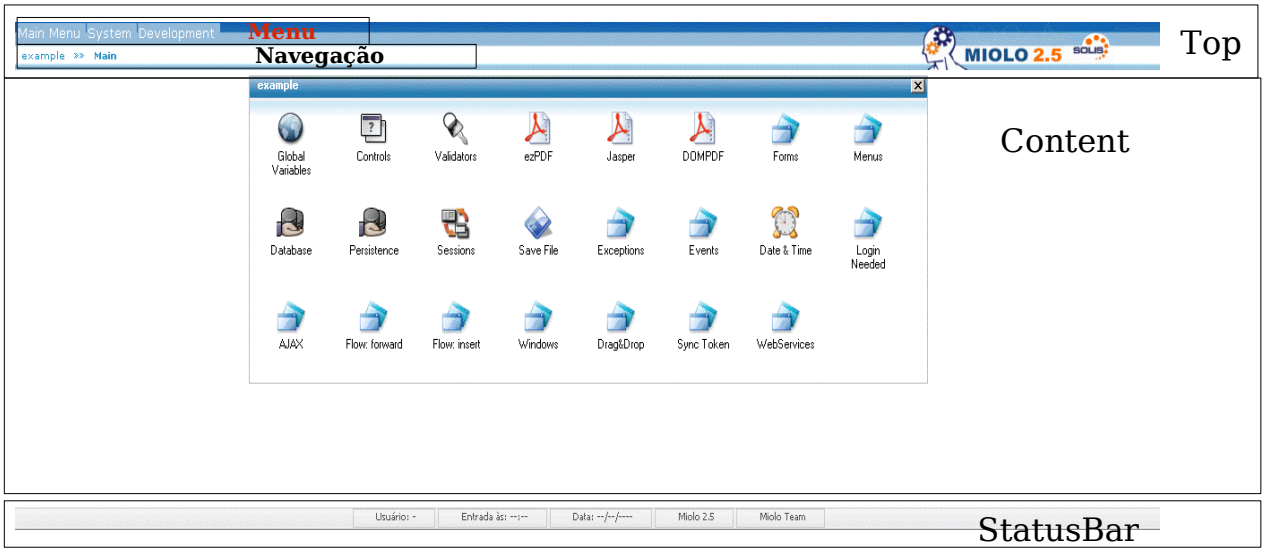

Figura 4 – Estrutura default do tema

Cada uma destas áreas é definida por um elemento do Tema (classe MThemeElement) e manipulada através dos métodos expostos pela classe Theme. A figura 5 mostra a organização lógica da estrutura do tema. Cada ThemeElement é renderizado como um controle HTML Div, com um atributo "id" ou "class", definido no tema.

Os handlers são responsáveis por gerar o conteúdo de cada uma das áreas visíveis. A definição do tema a ser usado é feita no arquivo miolo.conf (ou no module.conf, no caso dos módulos).

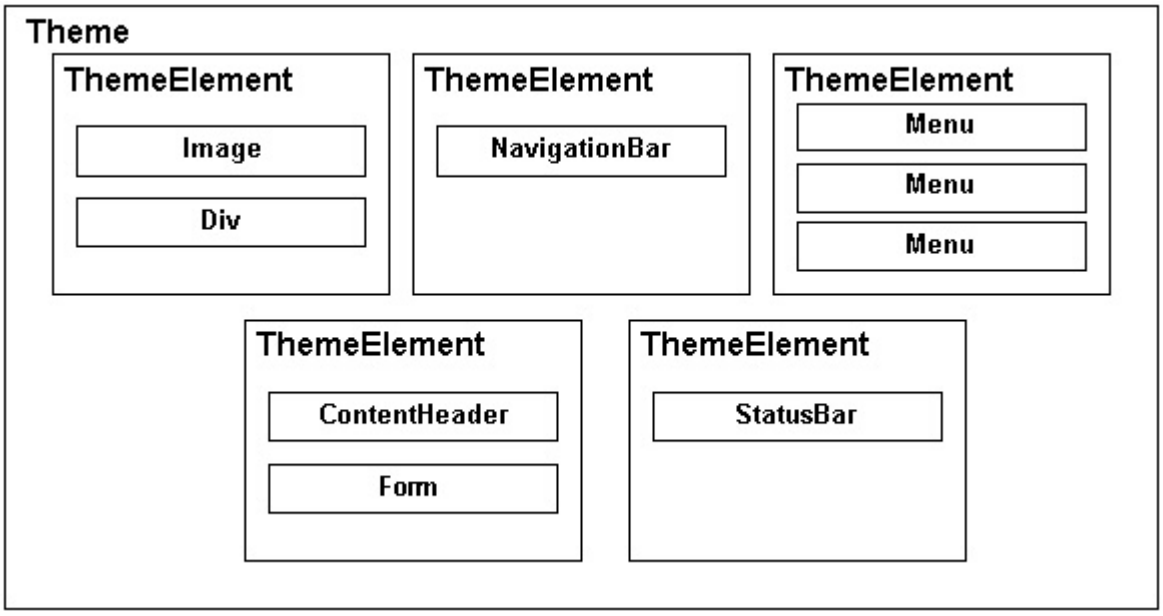

Figura 5 – Estrutura lógica do tema

A renderização do conteúdo do tema em uma página html é feita pelo próprio framework, através da chamada ao método $page->Generate(), no método MIOLO::Handler. Para esta renderização são utilizados a classe Theme<tema> (definida no arquivo theme.class.php) para os elementos do tema e o arquivo CSS (miolo.css), que devem ser definidos para cada tema (no diretório <miolo>/html/themes/<nome_do_tema>) e que serão usados para fazer a renderização de acordo com um tema específico. Assim, a geração de uma área específica do tema (top, navbar, menus, content, statusbar) pode ser customizada ou mesmo omitida.

O conteúdo HTML de cada elemento do tema é definido através de templates. Os templates são localizados no diretório <miolo>/html/themes/<nome_do_tema>/templates. Cada tema deve definir também os seguintes templates:

- base.php : usado para a renderização da página base
- default.php: usado para renderização do conteúdo completo (incluindo todos os elementos do tema) do form __mainForm
- window.php: usado para rendrização do conteúdo das janelas (geralmente apenas o elemento "content")

Da mesma forma, dentro da classe Theme<tema> devem ser definidos os métodos para geração dos layouts específicos (base, default, dynamic, window e lookup) e para cada um dos elementos do tema. Cada um destes métodos utiliza os templates definidos.

WebForms

Como visto, cada url será tratada pelo framework, gerando um formulário HTML. Esta funcionalidade está encapsulada na classe MPage. Cada "página" é gerada pelo processamento da seqüência de handlers. É importante observar, portanto, que os controles colocados na página devem ter nomes distintos, mesmo que estejam em objetos diferentes (por exemplo, os botões de submit de duas entradas de dados diferentes). Isto é válido mesmo para os objetos que serão renderizados em janelas (uma vez que as janelas são renderizadas como tags dentro da mesma página).

Como padrão, os formulários usados pelos handlers são armazenados em arquivos chamados <nome_do_form>.class.php, que contém a definição da classe do formulário, com os métodos dos formulário e os tratadores dos eventos (tipicamente as funções para tratar os eventos OnClick dos botões de submit).

Três métodos são executados automaticamente quando um formulário é instanciado: CreateFields, GetFormFields e OnLoad, nesta seqüencia. O método CreateFields é responsável pela definição dos campos do formulário e dos botões de ação. Os botões de ação podem ser do tipo SUBMIT (que gera um evento do tipo PostBack), RESET, PRINT (hard-copy da página), REPORT, uma URL (http://...) ou um método Ajax (:metodo). O método GetFormFields é chamado quando a página é submetida e é responsável por transferir os valores dos

dados enviados via browser para os campos do formulário que está sendo instanciado. O método OnLoad pode ser usado para criar um código qualquer de inicialização do formulário. É também definida a propriedade booleana *defaultButton* , usada para indicar se o formulário deve apresentar ou não um botão de submit, quando nenhum botão for definido.

Para testar se a página está sendo chamada a primeira vez, ou se ocorreu um "post" do formulário, podemos usar o atributo de formulário $this->page->isPostBack. A propriedade $this->page é uma instância da classe MPage, que representa as definições para a página atualmente sendo executada.

Para usar as variáveis cujo estado foi mantido entre round-trips, usamos os métodos do objeto State.

Os botões do tipo "submit" são programados para gerar eventos quando clicados. O nome do método que vai tratar o evento tem, por default, o formato <nome_do_botão>_click. Pode-se usar o método attachEventHandler, para definir um outro nome para o método que vai tratar o evento. No processamento do formulário, quando a página é submetida, deve-se usar o método EventHandler da classe MForm para que o manipulador do evento (um método do formulário que estejamos tratando) seja executado. O método EventHandler também trata eventos "forçados" através da URL. Um evento na URL é definido através da variável *event* (ex: http://.../index.php?module=...&action=...&event=btnPost:click). Neste caso podem ser passados parâmetros para os eventos.

4. Camada de Acesso a Dados

DAO – Data Access Objects

A camada DAO no Miolo tem por objetivo fazer a abstração do acesso a bancos de dados relacionais, encapsulando a camada de acesso fornecida pelo PHP, permitindo usar uma única interface de programação, independente do banco de dados sendo utilizado. Embora existam várias soluções para este problema, com frameworks específicos (tais como o AdoDB e as classes PEAR), o Miolo tem implementado sua própria versão de DAO.

A implementação atual tem as seguintes características:

1. O mecanismo de acesso a dados fornecido pelo PHP é encapsulado, permitindo uma interface única de programação.
2. São fornecidos mecanismos básicos para geração automática de código SQL adaptado ao banco (inclusive joins, offsets e número máximo de linhas retornadas), uso de geradores (sequences) e conversão de datas e horários para um formato padrão.
3. Uso de transações, utilizando recursos nativos do banco de dados sendo acessado.
4. Abstração de resultados de consultas (queries) em ResultSets, permitindo operações como travessia (browse), paginação, filtragem e ordenação do resultado de uma consulta.

As principais classes da camada DAO são:

- Class MDatabase
  - Define uma interface padrão para acesso a banco de dados, encapsulando as diversas extensões do PHP
  - Suporte a transações, conversão de tipos DATE/TIME, Generators/Sequences e geração de arquivos CSV/XML
- Classe MSQL
  - Encapsula a criação de comandos SQL, inclusive joins e ranges
- Classe MQuery
  - Usa o conceito de "ResultSet"
  - $query->result : array com linhas e colunas do resultado
  - Métodos para navegação no resultado das queries : MoveFirst(), MoveLast(), MoveNext(), etc.

A camada DAO encapsula o acesso aos seguintes SGBDs : Oracle, PostgreSQL, MySQL, SQLite, MSSQL, Firebird e ODBC.

Persistência

Um dos grandes objetivos do MIOLO é a criação de aplicações "realmente" OO . Devido à impedância dos modelos OO-Relacional , é necessário implementar uma camada intermediária. A implementação da camada de Persistência no MIOLO toma a proposta de Scott Ambler e acrescenta várias extensões.

A camada de Persistência permite:

- Mapeamento (via XML) das classes de negócio
- Mapeamento (via XML) das associações/herança entre classes (1:1, 1:N, N:N)
- Realização de queries utilizando a MOQL (Miolo Object Query Language), com o uso de "criterias"
- Conversão automática de tipo/conteúdo de atributos
- Indexação de atributos
- Utilização de subqueries
- Manipulação de campos BLOB
- Operações de conjuntos (UNION, INTERSECT)
- Uso de OUTER JOINS

Como está implementada acima da camada DAO, a camada de persistência tem suporte nativo à geração de OIDs, transações e acesso a múltiplos SGBDs.

5. Camada de Lógica de Negócio

As regras de negócio devem ser encapsuladas em objetos que representam as classes do domínio da aplicação. No MIOLO, estas classes devem estender da classe Mbusiness. A classe MBusiness herda da classe de persistência, tornando os objetos de negócio virtualmente persistentes. Estão disponíveis métodos tais como **save**, **delete** e **retrieve** que tratam automaticamente o acesso ao banco de dados, bem como métodos para fazer o controle de transações.

Para que um objeto Mbusiness possa ser utilizado em módulos diferentes daqueles em que foi definido, eles devem ser nomeados da seguinte forma:

```
class Business<modulo><classe> extends Mbusiness
```

Assim, a classe Pessoa, definida no módulo Common deve ser nomeada

```
class BusinessCommonPessoa extends Mbusiness
```

6. Segurança

Base de dados ADMIN

A base de dados ADMIN (<miolo>/modules/admin/sql/admin.sqlite) fornecida junto com o framework é usada na administração de usuários, transações e grupos. Seu principal objetivo é fornecer relativa independência para o framework, embora ela possa ser customizada em cada instalação (ou seja, cada instalação pode optar por desenvolver sua própria base de dados de administração, desde que sejam respeitados os campos usados pelo framework).

Os usuários são armazenados na tabela miolo_user. Podem ser usados hashes MD5, evitando que a senha seja armazenada em texto claro no banco de dados. Cada usuário pode estar associado a um ou vários grupos (tabela miolo_group). Cada grupo tem direitos de acesso a uma ou mais transações (tabela miolo_transaction). Os direitos de acesso estão representados na tabela miolo_access, como um valor inteiro representado em PHP da seguinte forma:

```
define('A_ACCESS',    1); // 000001
define('A_QUERY',     1); // 000001

define('A_INSERT',    2); // 000010
define('A_DELETE',    4); // 000100
define('A_UPDATE',    8); // 001000
define('A_EXECUTE',  15); // 001111
```

```
define('A_SYSTEM',   31); // 011111
define('A_ADMIN',    31); // 011111

define('A_DEVELOP',  32); // 100000
```

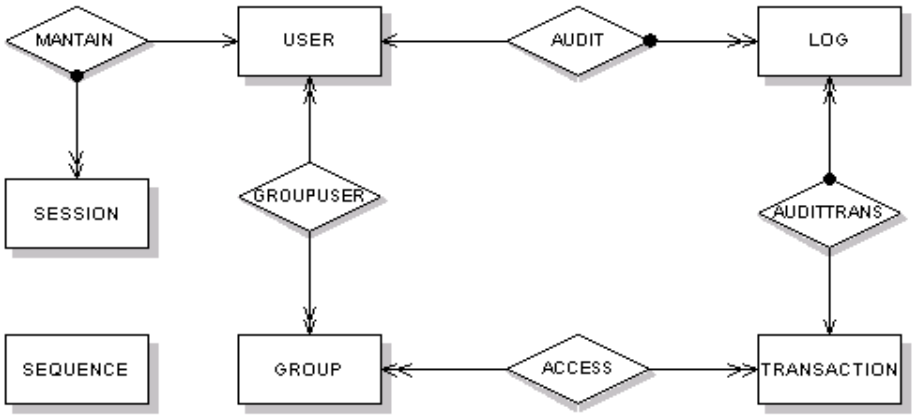

Figura 6 – Base de dados ADMIN

Tabelas

```
CREATE TABLE miolo_sequence (
       sequence                     CHAR(20)       NOT NULL,
       value                        INTEGER);

CREATE TABLE miolo_user (
       iduser                       INTEGER        NOT NULL,
       login                        CHAR(25),
       name                         VARCHAR(80),
       nickname                     CHAR(25),
       m_password                   CHAR(40),
       confirm_hash                 CHAR(40),
       theme                        CHAR(20));

CREATE TABLE miolo_transaction (
       idtransaction                INTEGER        NOT NULL,
       m_transaction                CHAR(30));

CREATE TABLE miolo_group (
       idgroup                      INTEGER        NOT NULL,
       m_group                      CHAR(50));

CREATE TABLE miolo_access (
       idgroup                      INTEGER        NOT NULL,
       idtransaction                INTEGER        NOT NULL,
       rights                       INTEGER);

CREATE TABLE miolo_session (
       idsession                    INTEGER        NOT NULL,
       tsin                         CHAR(15),
       tsout                        CHAR(15),
       name                         CHAR(50),
       sid                          CHAR(40),
       forced                       CHAR(1),
       remoteaddr                   CHAR(15),
       iduser                       INTEGER        NOT NULL);

CREATE TABLE miolo_log (
       idlog                        INTEGER        NOT NULL,
       m_timestamp                  CHAR(15),
       description                  VARCHAR(200),
       module                       CHAR(25),
       class                        CHAR(25),
       iduser                       INTEGER        NOT NULL,
       idtransaction                INTEGER        NOT NULL);

CREATE TABLE miolo_groupuser (
       iduser                       INTEGER        NOT NULL,
      idgroup                        INTEGER        NOT NULL);
```

Autenticação

Para autenticação são usados métodos da classe MAuth. A autenticação pode ser feita contra um banco de dados ou uma base LDAP (os parâmetros para autenticação são definidos no arquivo de configuração). Os principais métodos são:

- $auth->checkLogin() : verifica se há algum usuário logado, redirecionando para o formulário de login, se não houver.
- $auth->authenticate($uid, $pwd) : autentica o usuário $uid, com a senha $pwd
- $auth->authenticate($uid, $challenge, $response) : autentica o usuário $uid, que possui a senha armazenada em MD5, usando o mecanismo challenge-response.

Depois de autenticado, os dados do usuário (e os grupos aos quais ele pertence) são armazenados na sessão e registrados no objeto MLogin.

Autorização (Permissões)

Para autorização são usados métodos da classe Mperms:

- $perms->checkAccess($transaction, $access, $deny = false) : verifica se o usuário logado tem, no mínimo, o direito $access (um número inteiro) na transação $transaction. $deny = true indica que o processamento será interrompido caso o usuário não tenha direito de acesso e $deny = false faz com que o método retorne FALSE, sem interromper o processamento.
- $perms->isAdmin() : retorna TRUE se o usuário logado for administrador (se pertence ao grupo ADMIN).

Logs

Para geração dos logs são usados métodos da classe MLog. Nos logs podem ser armazenadas mensagens genéricas, mensagens de erro e comandos SQL. As mensagens podem ser armazenadas em arquivos (no diretório <miolo>/var/log), no banco de dados ADMIN (tabela miolo_log) ou serem enviadas via rede (socket). São também estabelecidos os seguintes níveis de log: 0 – nenhum log; 1 – somente erros; 2 – erros e mensagens. As configurações relativas à geração de logs estão no arquivo miolo.conf.

Os principais métodos da classe MLog são:

- $log->logSQL($sql,$force=false,$conf='?') : registra todos os comandos SQL enviados para o banco de dados.
- $log->logError($error,$conf='miolo') : registra as mensagens de erro (p.ex. Na execução de um comando SQL).
- $log->logMessage($msg) : registra uma mensagem genérica.
- $log->isLogging(): retorna TRUE se o nível de log > 0.

7. Debug e Tratamento de Exceções

Trace

Para facilitar o mecanismo de debug, o MIOLO possui métodos de trace, que permitem enviar para o log mensagens genéricas:

- $MIOLO->trace($msg, $file = '', $line = 0) : registra uma mensagem genérica no log. $file e $line podem ser usados para prover informações adicionais
- $MIOLO->traceStack() : registra no log um "stack trace", que permite acompanhar quais métodos foram executados até o instante.

Exceções

O MIOLO define uma série de exceções customizadas que podem ser usadas pelos desenvolvedores:

- EmioloException : exceção genérica. Estende a classe Exception do PHP5.

- EInOutException : acesso a arquivos
- EDatabaseException : acesso a banco de dados
- EDatabaseExecException : execução de comandos DML (insert, update, delete) no banco de dados
- EDatabaseQueryException : execução de comandos SELECT no banco de dados
- EDataNotFoundException : exceção genérica de acesso a dados
- EDatabaseTransactionException : execução de transações no banco de dados
- EControlException : execução dos handlers
- EUsesException : inclusão de arquivos
- EFileNotFoundException : acesso a arquivos
- ESessionException : acesso a sessão
- EBusinessException : exceção genérica associada a objetos MBusiness
- ETimeOutException : exceção associada à expiração da sessão
- ELoginException : falha na autenticação
- ESecurityException : falha na autorização

8. Configuração

As configurações do MIOLO, como localização dos arquivos, bases de dados, temas, entre outros, estão definidas no arquivo **<miolo>/etc/miolo.conf.** Cada módulo pode definir sua própria configuração (ou redefinir alguma configuração global) no arquivo **<miolo>/modules/<nome_modulo>/etc/module.conf**

Arquivo miolo.conf

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<conf>
    <home>
Diretório base do framework
          <miolo>path_miolo</miolo>
Diretório das classes
          <classes>path_miolo/classes</classes>
Diretório dos módulos
          <modules>path_miolo/modules</modules>
Diretório dos arquivos de configuração
          <etc>path_miolo/etc</etc>
Diretório dos logs
          <logs>path_miolo/var/log</logs>
Diretório do trace
          <trace>path_miolo/var/trace</trace>
Diretório do banco de dados interno ao MIOLO
          <db>path_miolo/var/db</db>
Diretório HTML (acessível via browser)
          <html>path_miolo/html</html>
Diretório dos temas (acessíveis via browser)
          <themes>path_miolo/html/themes</themes>
Diretório classes de extensão
          <extensions>path_miolo/classes/extensions</extensions>
Diretório dos relatórios gerados
          <reports>path_miolo/var/reports</reports>
Diretório das imagens (acessíveis via browser)
          <images>path_miolo/html/images</images>
URL base (conforme configurado no servidor web)
          <url>http://miolo.domain</url>
URL dos temas
          <url_themes>/themes</url_themes>
URL dos relatórios
          <url_reports>/reports</url_reports>
Diretório base dos temas disponíveis no módulo
          <module.themes>/ui/themes</module.themes>
Diretório interno ao módulo, acessível via browser
          <module.html>/html</module.html>
Diretório de imagens, interno ao módulo, acessível via browser
          <module.images>/html/images</module.images>
Diretório da instalação do Java (usado em relatórios JasperReports)
          <java>E:\java\jdk6</java>
    </home>

Definição dos namespaces usados pelo framework
```

```
  <namespace>
    <core>/classes</core>
    <service>/classes/services</service>
    <ui>/classes/ui</ui>
    <themes>/html/themes</themes>
    <extensions>/classes/extensions</extensions>
    <controls>/ui/controls</controls>
    <database>/classes/database</database>
    <utils>/classes/utils</utils>
    <modules>/modules</modules>
    <business>/classes</business>
  </namespace>

Definição do tema base a ser usado. Indica a localização do tema, que pode estar na estrutura
do framework ou interno a algum módulo
    <theme>
Se <module> estiver vazio, o tema deve ser definido em <path_miolo>/html/themes
          <module></module>
          <main>blue</main>
          <lookup>blue</lookup>
Título usado na janela do browser
          <title>Miolo Web Application</title>
Company, system, logo e email podem ser usados em customizações do tema, e se referem a uma
instalação
         <company>Company</company>
         <system>System</system>
         <logo>logo.gif</logo>
         <email>miolo@miolo.org</email>
    </theme>


Definição do gerenciamento de sessões
    <session>
Indica como os dados das sessões serão armazenados:
- files: somente em arquivos no lado do servidor (usando os serviços do PHP)
- db: armazenados também no banco de dados de execução
          <handler>db</handler>
Define o tempo de inatividade da sessão (em minutos), antes que ela seja encerrada
          <timeout>20</timeout>
    </session>

    <options>
Define o modulo que sera usado por default (quando não for informado na URL)
          <startup>common</startup>
Define o modulo que contem dados comuns aos demais módulos
          <common>common</common>
Define se os parâmetros na URL serão criptografados
          <scramble>0</scramble>
Define o script base para execução
          <dispatch>index.php</dispatch>
Define o formato da URL:
- 0: http://<miolo>/<dispatch>?module=<module>&action=<action>&...
- 1: http://<miolo>/<dispatch>/<module>/<action>/<...>
          <url.style>0</url.style>
Define se a autenticação usa senhas com hashes MD5
          <authmd5>false</authmd5>
Define como o menu principal vai ser exibido:
- 0: não exibe o menu
- 1: exibe o menu na posição definida pelo tema (geralmente à esquerda)
- 2: utiliza um menu "suspenso" com DHTML – biblioteca Tigra
- 3: utiliza um menu "suspenso" com DHTML – biblioteca JsCookMenu
    <mainmenu>3</mainmenu>
    <mainmenu.style>office2003</mainmenu.style>
    <mainmenu.clickopen>false</mainmenu.clickopen>

Define se vai ser feito um log da sessão do usuário
          <dbsession>0</dbsession>
Define se a autenticação vai utilizar "criptografia" ou não
          <authmd5>0</authmd5>
Define se vai ser feito o debug ou não (trace)
          <debug>1</debug>
Define o charset usado na geração da página HTML e na comunicação via AJAX
```

```
          <charset>ISO-8859-1</charset>
Define a extensão dos arquivos dos módulos:
- 2: não utiliza a extensão .php
- 25: utiliza a extensão .php
          <fileextension>25</fileextension>

Define os parâmetros para caso de um dump, durante o processo de debug
         <dump>
             <peer>127.0.0.1</peer>
             <profile>false</profile>
             <uses>false</uses>
             <trace>false</trace>
             <handlers>false</handlers>
        </dump>
Define se será exibida a mensagem de 'loading' durante ao acesso via AJAX
           <loading>
               <show>false</show>
               <generating>false</generating>
           </loading>
    </options>

Define qual o módulo será usado para administração do MIOLO (MAD – Miolo Administration
Module) e quais os nomes das classes a serem usadas pelo framework
    <mad>
       <module>admin</module>
       <classes>
             <access>access</access>
             <group>group</group>
             <log>log</log>
             <session>session</session>
             <transaction>transaction</transaction>
             <user>user</user>
       </classes>
    </mad>

Define parâmetros para os arquivos de log
    <logs>
Nível de log:
- 0: nenhum log
- 1: somente erros
- 2: erros e mensagens
        <level>2</level>
Handler do Log de debug (trace):
- socket: as mensagens são enviadas via tcp/ip para o host indicado "peer" e para a porta
indicada em "port"
- db: as mensagens são armazenadas no banco de dados de execução (no diretório
<miolo>/var/db)
- file: as mensagens são armazenadas em arquivos no diretório <miolo>/var/log
        <handler>socket</handler>
        <peer>127.0.0.1</peer>
          <port>9999</port>
    </logs>

Configuração das bases de dados
    <db>
Banco de dados de execução <miolo> – faz parte do framework
       <miolo>
DBMS: firebird, mysql, postgres, sqlite, oracle8
           <system>sqlite</system>
Servidor do banco de dados
           <host>localhost</host>
Nome do banco de dados
           <name>/usr/local/miolo2/var/db/miolo.sqlite</name>
Usuário e senha para acesso
           <user>miolo</user>
           <password>miolo</password>
       </miolo>
       <admin>
           <system>postgres</system>
           <host>alpha</host>
           <name>dev</name>
           <user>miolo</user>
```

```
            <password>xxxxxxxx</password>
Definições para uso do Java, nos relatórios com JasperReports
           <jdbc_driver>org.postgresql.Driver</jdbc_driver>
            <jdbc_db>jdbc:postgresql://localhost/dev</jdbc_db>
        </admin>
    </db>

Definição de parâmetros para autenticação:
- se vai checar o login ou não
- se o login é automático (qual login) ou não

check    shared    auto     result
-----------------------------------------------------------------
true     true      false    usuário deve estar cadastrado em cm_usuario
true     false     false    usuário deve estar cadastrado em cm_usuario
false    true      false    não é necessário cadastro no cm_usuario
true     true      true     usuario pre-definido deve existir no cm_usuario
false    true      true     usuario pre-definido não é necessário no cm_usuario
    <login>
Define em qual módulo está o formulário para login
        <module>admin</module>
Define qual a classe será usada para processar a autenticação:
- MauthDb: senha em texto claro
- MauthDbMD5: senha "criptografada" com hash MD5
        <class>MAuthDb</class>
        <check>1</check>
        <shared>1</shared>
        <auto>user1</auto>
        <user1>
            <id>teste</id>
            <password>pass</password>
            <name>Usuario Teste</name>
        </user1>
    </login>
</conf>
```