MIOLO 2.0

Overview

Ely Edison Matos

ely.matos@ufjf.edu.br

versão do documento: 1.1
16/05/2005

Índice

1. Introdução

O que é o MIOLO?

Podemos definir o MIOLO como sendo 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. 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.

Como pré-requisito para utilizar o MIOLO é necessário ter conhecimento de programação com PHP 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:

- Controles de interface com o usuário, escritos em PHP e renderizados em HTML
- Autenticação de usuários
- Controle de permissão de acesso
- Camada de abstração para acesso a bancos de dados
- Camada de persistência transparente de objetos
- Gerenciamento de sessões e estado
- Manutenção de logs
- Mecanismos de trace e debug
- Tratamento da página como um webform, no modelo event-driven
- Validação de entrada em formulários
- Customização de layout e temas, usando CSS
- Geração de arquivos PDF

2. Arquitetura

O MIOLO adota a arquitetura em camadas (figura 1), implementando o padrão MVC (Model-View-Controller) (figura 2).

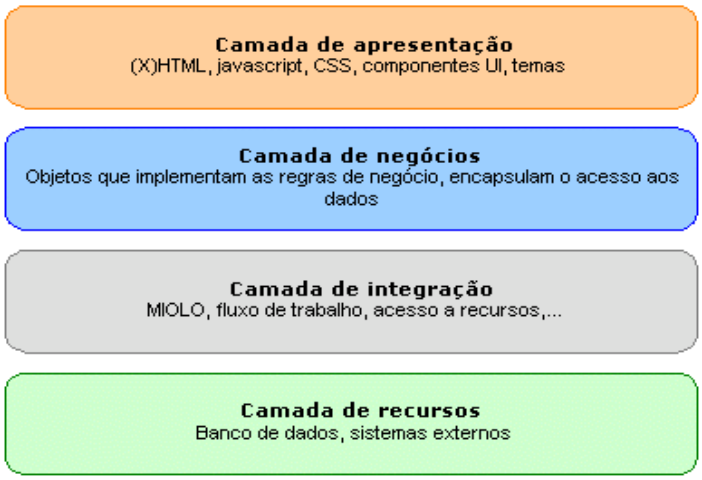

Figura 1 - Camadas

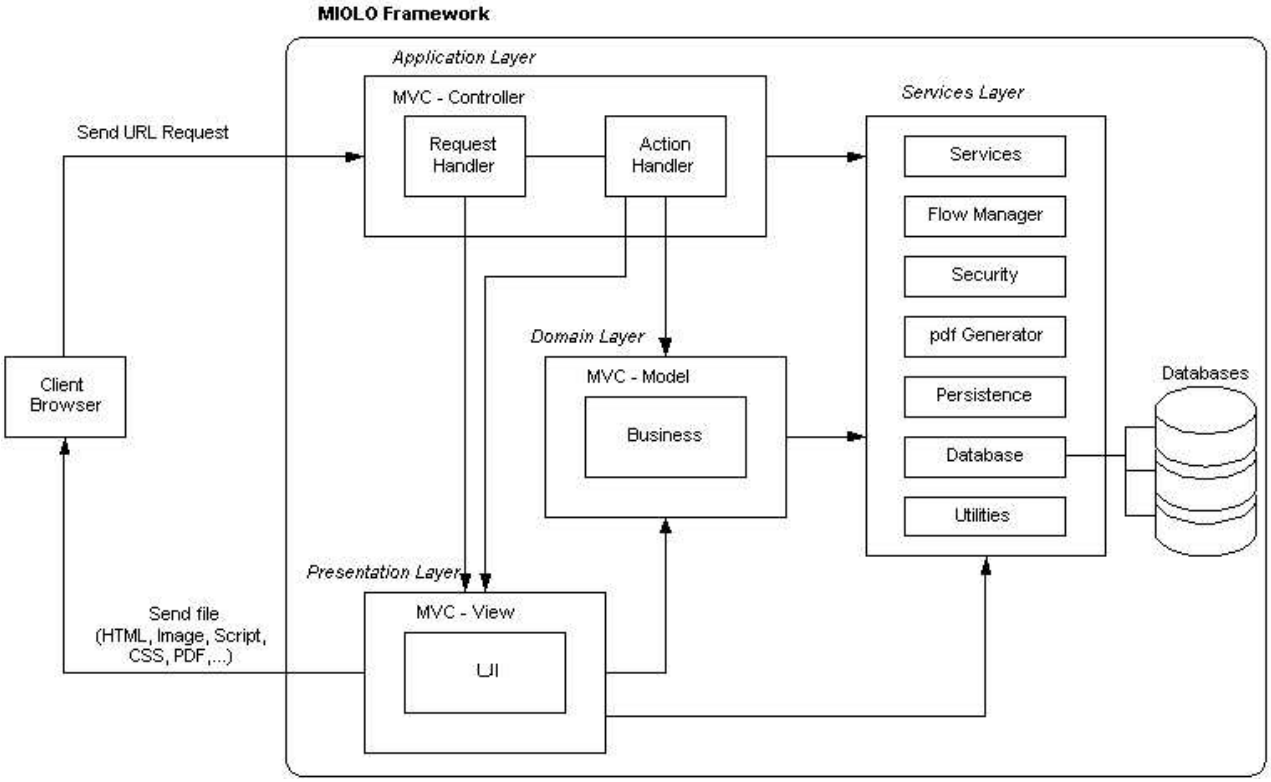

Figura 2 – Padrão MVC

Implementação das camadas

MIOLO – camada de integração

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

User Interface (UI) – camada de apresentação

São as classes do framework responsáveis pela geração de arquivos, renderização dos controles HTML e da 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).

Handlers – camada de integração

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 UI para definir a saída para o cliente. Estão localizadas no diretório handlers de cada módulo.

Business – camada de negócios

São as classes criadas pelo desenvolvedor para representar o domínio da aplicação (as regras de negócio). São usadas pelas camadas UI e handlers, acessando o banco de dados através da camada BD.

BD – camada de recursos

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

Utils e Services – camada de recursos

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

A figura 3 mostra a organização destas camadas na estrutura do framework.

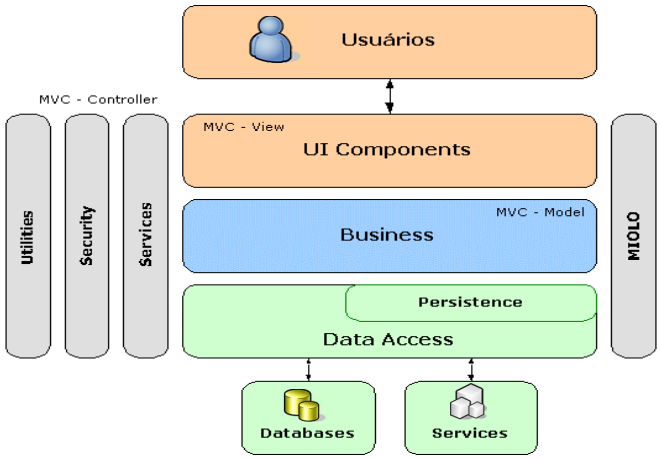

Figura 3 – Implementação das camadas

3. Instalação

O MIOLO deve ser instalado em um diretório acessível pelo servidor web. Por padrão, o MIOLO é instalado no diretório /usr/local/miolo, dentro do qual são criados os subdiretórios (listados abaixo). Instalados os arquivos do MIOLO, é necessário definir, no Apache, o diretório root dos arquivos ou criar um VirtualHost que aponte para esse diretório (/usr/local/miolo/html). Dessa forma, ao digitar o endereço no browser, o mesmo acessará o arquivo index.html deste diretório. Naturalmente o Apache deve estar configurado para executar scripts PHP5.

Exemplo de configuração do Apache:

```
<VirtualHost miolo.domínio.com.br>
    ServerAdmin admin@dominio.com.br
    DocumentRoot /usr/local/miolo/html
    ServerName miolo.domínio.com.br
    ErrorLog /var/log/apache/miolo-error.log
    CustomLog /var/log/apache/miolo-access.log common
</VirtualHost>
```

O diretório /usr/local/miolo/var (e seus subdiretórios) devem ter permissão de escrita pelo Apache.

Estrutura de diretórios

```
-----<diretório-base>(p.ex. /usr/local/miolo)
       |
       +--- 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 e scripts. 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_common.js - Esse arquivo contém as principais funções javascript utilizadas pelos componentes e pelo framework.

<miolo>/classes/miolo.class - 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.

4. Programando com o MIOLO

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 desenvolver 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.

Módulo

Um módulo é um componente de uma aplicação. De forma geral, um módulo reflete um sub-domínio da aplicação, agregando as classes de negócio que estão fortemente relacionadas e provendo o fluxo de execução (handlers) e a interface com o usuário (forms, grids, reports) para se trabalhar com tais classes. Um módulo é caracterizado por um nome, usado como subdiretório do diretório <miolo>/modules. Cada módulo tem uma estrutura de diretórios padrão, que é usada pelo framework para localizar os recursos, e possui seu próprio arquivo de configuração (que pode redefinir as configurações globais feitas no miolo.conf).

Controles (Widgets)

Os controles são componentes de interface com o usuário, usados na renderização das páginas html. Um controle pode agregar outros controles e tem propriedades e eventos associados a ele.

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 Theme) que trabalha como um container para os controles que vão ser renderizados na página HTML. Cada tema define "elementos", e cada elemento agrega controles visuais específicos. Um tema é associado a uma (ou várias) folha de estilos (um arquivo CSS) que define o posicionamento, as dimensões e a aparência dos controles a serem renderizados. Podem ser definidos vários temas (cada um com seu próprio diretório, no diretório <miolo>/classes/ui/themes), embora geralmente cada módulo utilize apenas um tema.

Handler

Um handler é uma instância da classe MHandler. Sua função é tratar a solicitação feita pelo usuário através do browser. Em cada módulo (no diretório <miolo>/modules/<modulo>/handler) é definida uma classe Handler<Modulo>, que é instanciada pelo MIOLO quando é feita a análise da solicitação do usuário. O controle é então passado para esta classe, que inclui o handler específico para tratar a solicitação. O handler atua, assim, no papel de controller, fazendo a integração entre as regras de negócio e a interface com o usuário.

Namespace

Namespaces são apenas aliases para diretórios. O objetivo do uso de namespaces é a possibilidade de mudança 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.

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>/usr/local/miolo</miolo>
```

- *Diretório classes*

```
        <classes>/usr/local/miolo/classes</classes>
```

- *Diretório base dos módulos*

```
        <modules>/usr/local/miolo/modules</modules>
```

- *Diretório base da configuração*

```
        <etc>/usr/local/miolo/etc</etc>
```

- *Diretório para arquivos de log*

```
        <logs>/usr/local/miolo/var/log</logs>
```

- *Diretório para arquivos de debug*

```
        <trace>/usr/local/miolo/var/trace</trace>
```

- *Diretório da base de dados de execução*

```
        <db>/usr/local/miolo/var/db</db>
```

- *Diretório base para as páginas*

```
        <html>/usr/local/miolo/html</html>
```

- *Diretório base dos temas disponíveis no framework*

```
        <themes>/usr/local/miolo/classes/ui/themes</themes>
```

- *Diretório para armazenamento de arquivos PDF (relatórios)*

```
        <reports>/usr/local/miolo/var/reports</reports>
```

- *Diretório da imagens usadas pelo framework*

```
        <images>/usr/local/miolo/html/images</images>
```

- *URL base (conforme configurado no servidor web)*

```
        <url>http://miolo.dominio.com.Br</url>
```

- *URL base para obter arquivos dos temas (p.ex. folhas de estilo)*

```
<url_themes>/themes</url_themes>
```

- *URL base para obter arquivos PDF (relatórios)*

```
<url_reports>/reports</url_reports>
```

- *Diretório base dos temas disponíveis no módulo*

```
<module.themes>/ui/themes</module.themes>
```

- *Diretório base dos arquivos internos ao módulo, acessíveis via browser*

```
<module.html>/html</module.html>
```

- *Diretório base das imagens usadas pelo módulo, acessíveis via browser*

```
<module.images>/html/images</module.images>
</home>
```

- *Definição dos namespaces usados pelo framework*

```
<namespace>
   <core>/classes</core>
   <service>/classes/services</service>
   <ui>/classes/ui</ui>
   <themes>/ui/themes</themes>
   <extensions>/classes/extensions</extensions>
   <controls>/ui/controls</controls>
   <database>/classes/database</database>
   <util>/classes/util</util>
   <modules>/modules</modules>
</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 /usr/local/miolo/classes/ui/themes, caso contrário, deve ser definido em /usr/local/miolo/modules/<module>/ui/themes*

```
<module></module>
<main>kenobi</main>
<lookup>kenobi</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>Universidade Federal de Juiz de Fora</company>
<system>SIGA - Sistema de Gestão Acadêmica</system>
<logo>logonet.gif</logo>
<email>siga@ufjf.edu.br</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, 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>admin</startup>
```

- *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 criptografadas*

```
<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*

```
<mainmenu>2</mainmenu>
```

- *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 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>
</options>
```

- *Define qual o módulo será usado para administração do MIOLO 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/trace*

```
            <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/miolo/var/db/miolo.sqlite</name>
```

- *Usuário e senha para acesso*

```
               <user>miolo</user>
               <password>miolo</password>
           </miolo>
           <admin>
               <system>oracle8</system>
```

```
                <host>alpha</host>
                <name>dev</name>
                <user>miolo</user>
                <password>xxxxxxxx</password>
            </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 autentição:*
  - *MauthDb: senha em texto claro*
  - *MauthDbMD5: senha "criptografada" com 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>
```

Desenvolvimento

Tendo em vista os conceitos apresentados, podemos dizer que o processo de desenvolvimento de aplicações com o MIOLO possui as seguintes etapas:

- Modelagem das classes e do banco de dados
- Criação da estrutura do módulo, com seus subdiretórios
- Definição do tema a ser utilizado (um já existente, ou a criação de um novo tema)
- Criação de controles específicos para o módulo, caso seja necessário
- Criação do arquivo de configuração do módulo em <miolo>/modules/<modulo>/etc/module.conf
- Criação da classe handler em <miolo>/modules/<modulo>/handler/handler.class
- Criação do handler principal em <miolo>/modules/<modulo>/handler/main.inc
- Criação das classes de negócio em <miolo>/modules/<modulo>/classes – caso existam

- Criação dos formulários em <miolo>/modules/<modulo>/forms – caso existam

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 generalizara para acessar:

A - Handlers
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 – Arquivos

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

C – Templates (por enquanto usado só para customizaçao de temas)

```
Ex: http://host.dominio/index.php?module=tutorial3&action=themes:mystica:blue.tpl
```

O módulo igual a "miolo" indica que o arquivo é do core e não de algum módulo específico.

Além disso, está implementada a escrita de URLs amigáveis:
Ex: http://host.dominio/index.php/common/html/images/save.png

Com o uso do mod_rewrite do Apache será possível evitar o index.php.

Ciclo de vida da página

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.

Módulos

No MIOLO, todos os sistemas 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/miolo)
      |
      +--- modules
      |           +--- module1
      |           |           +--- classes
      |           |           +--- forms
      |           |           +--- menus
      |           |           +--- sql
      |           |           +--- handlers
      |           |           +--- grids
      |           |           +--- reports
      |           |           +--- inc
      |           |           +--- etc
      |           |           +--- html
      |           |           |    +--- images
      |           |           |    +--- files
      |           |           +--- ui
      |           |                +--- controls
      |           |                +--- themes
      |           |
      |           +--- 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 page
- $**context**: acesso ao objeto context
- $**module**: nome do módulo do handler em execução (ex: 'admin')
- $**action**: url completa do handler em execução
- $**item**: campo item da url atual
- $**self**: path do handler em execução
- $**history**: objeto com histórico das urls acessadas

5. 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).

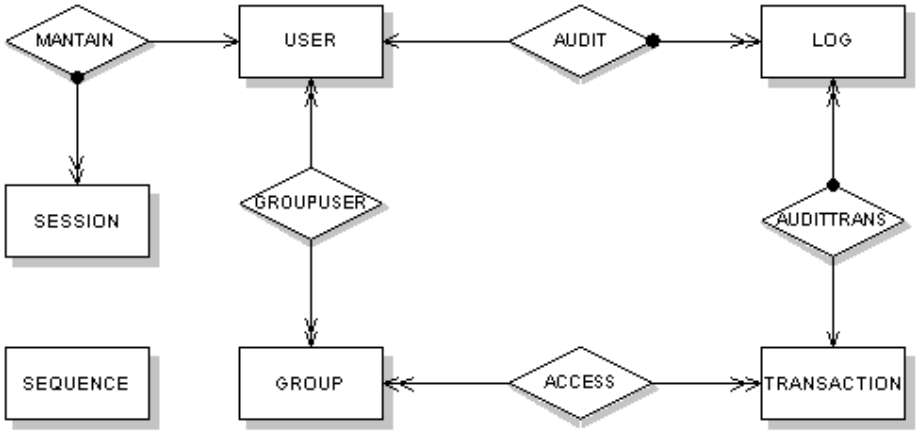

Figura 4 – Base de dados ADMIN

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);
```

6. Interface com o usuário

Temas

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.

Por default, a estrutura do tema é a seguinte:

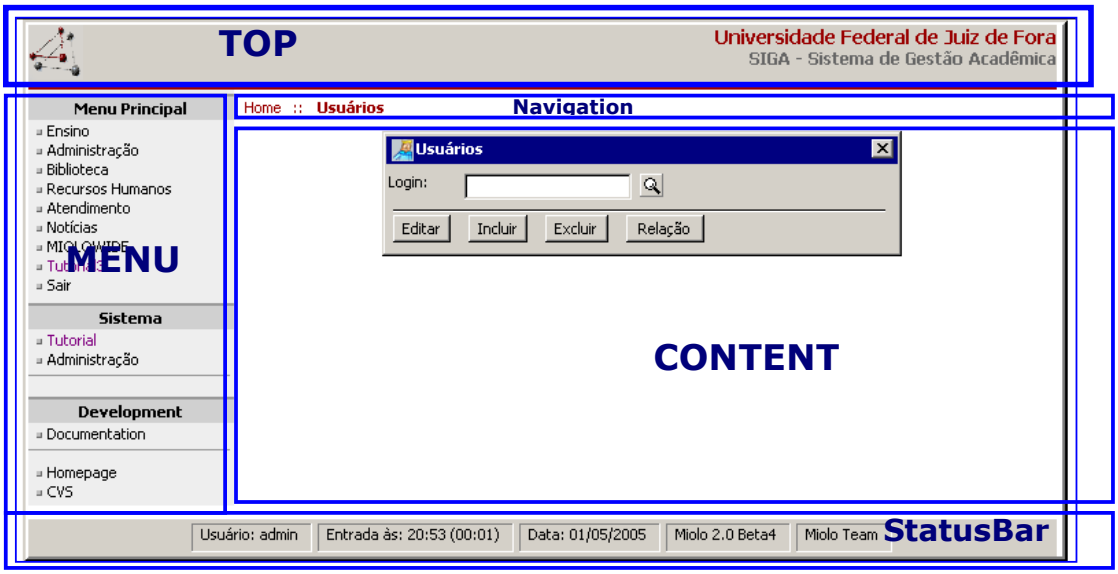

Figura 5 – Estrutura default do tema

Cada uma destas áreas é definida por um elemento do Tema (classe MThemeElement) e manipulada através dos métodos expostos pela classe Theme. A figura 6 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 arquivo m_themeelement.css relativo ao 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).

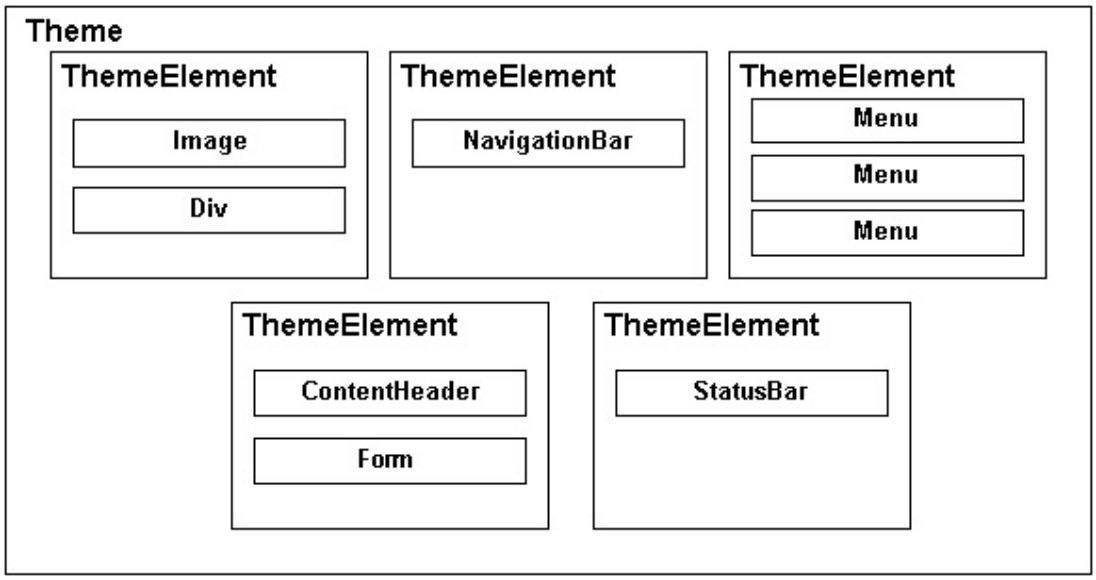

Figura 6 – Estrutura lógica do tema

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 (theme.class) para os elementos do tema e o arquivos com a folhas de estilo CSS, que devem ser definidos para cada tema (no diretório <miolo>/classes/ui/themes/<nome_do_tema> ou em <miolo>/modules/<modulo>/ui/themes/<nome_do_tema>) e que serão usadas para fazer a renderização de acordo com um tema específico. Assim, a geração de uma área específica do tema (title, navbar, menus, content, statusbar) pode ser customizada ou mesmo omitida.

Dentro da classe Theme, devem ser definidos os métodos para geração dos layouts específicos (default, lookup, popup, htmlarea). Cada um destes métodos define quais elementos serão renderizados.

WebForms

Como visto, cada url será tratada pelo framework, gerando uma página html. A funcionalidade é encapsulada em um objeto da classe MPage. Cada página é gerada pelo processamento da seqüência de handlers e constitui-se em um único formulário html, mesmo que vários controles estejam presentes na página. 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). O objeto MPage também é responsável pela renderização dos elementos não-visíveis (CSS styles, metas, scripts) que são gerados pelo framework.

Como padrão, os formulários usados pelos handlers são armazenados em um arquivo chamado <nome_do_form>.class, 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 ou ainda uma URL (http://...) que será chamada através da função javascript GotoURL (do arquivo m_common.js). 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, pode-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 (quando é usado o método GET, ao invés do POST). 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.

Controles (widgets)

Os controles visuais são classes programadas em PHP e que encapsulam controles html (ou controles contruídos em javascript). Cada controle tem um método getRender, responsável por gerar o código html/javscript correspondente. A renderização está encapsulada na classe MHTMLPainter (<miolo>/classes/ui/painter). Cada método desta classe recebe um objeto e gera o código html correspondente.
Os controle do framework podem ser correspondentes a um único controle Html, ou podem ser construídos através da composição de outros controles. Os usuários também podem construir seus próprios controles com base nos já existentes, através de herança.

Árvore de controles

+----MComponent
+----+----MControl
+----+----+----M_SPAW_Wysiwyg
+----+----+----MSpan
+----+----+----MDiv
+----+----+----+----MSpacer
+----+----+----+----MHr
+----+----+----+----MBoxTitle
+----+----+----+----MBox
+----+----+----MBaseGrid
+----+----+----+----MGridColumn
+----+----+----+----+----MGridHyperlink
+----+----+----+----+----+----MDataGridHyperlink
+----+----+----+----+----+----MObjectGridHyperlink
+----+----+----+----+----MGridControl
+----+----+----+----+----+----MDataGridControl
+----+----+----+----+----+----MObjectGridControl
+----+----+----+----+----+----MPDFReportControl
+----+----+----+----+----MDataGridColumn
+----+----+----+----+----MObjectGridColumn
+----+----+----+----+----MPDFReportColumn
+----+----+----+----MGridAction
+----+----+----+----+----MGridActionIcon
+----+----+----+----+----MGridActionText
+----+----+----+----+----MGridActionSelect
+----+----+----+----+----MDataGridAction
+----+----+----+----+----MObjectGridAction
+----+----+----+----MGridFilter
+----+----+----+----+----MGridFilterText
+----+----+----+----+----MGridFilterSelection
+----+----+----+----+----MGridFilterControl
+----+----+----+----MGrid
+----+----+----+----+----MActiveGrid
+----+----+----+----+----+----MActiveLookupGrid
+----+----+----+----+----MDataGrid
+----+----+----+----+----+----MDataGrid2
+----+----+----+----+----+----+----MLookupGrid
+----+----+----+----+----MObjectGrid

+----+----+----+----+----MPDFReport
+----+----+----MAreaContainer
+----+----+----MContent
+----+----+----+----MFileContent
+----+----+----MContentHeader
+----+----+----MDHTMLMenu
+----+----+----MError
+----+----+----MForm
+----+----+----+----MCompoundForm
+----+----+----+----MCSSForm
+----+----+----+----MCSSPForm
+----+----+----+----MIndexedForm
+----+----+----+----MTabbedForm
+----+----+----+----MTabbedForm2
+----+----+----MFormControl
+----+----+----+----MButton
+----+----+----+----+----MInputButton
+----+----+----+----+----+----MButtonFind
+----+----+----+----+----MButtonWindow
+----+----+----+----MChoiceControl
+----+----+----+----+----MCheckBox
+----+----+----+----+----MRadioButton
+----+----+----+----MImage
+----+----+----+----+----MImageFormLabel
+----+----+----+----MTextField
+----+----+----+----+----MPasswordField
+----+----+----+----+----MHiddenField
+----+----+----+----+----MMultiLineField
+----+----+----+----+----+----MHtmlArea
+----+----+----+----+----MFileField
+----+----+----+----+----MCalendarField
+----+----+----+----+----MCurrencyField
+----+----+----+----+----MLookupField
+----+----+----+----+----+----MLookupTextField
+----+----+----+----+----+----+----MActiveLookup
+----+----+----+----+----+----MLookupFieldValue
+----+----+----+----+----MMultiTextField2
+----+----+----+----+----+----MMultiTextField3
+----+----+----+----MBaseLabel
+----+----+----+----+----MPageComment
+----+----+----+----+----MSeparator
+----+----+----+----+----MLabel
+----+----+----+----+----MFieldLabel
+----+----+----+----+----MTextHeader
+----+----+----+----+----MText
+----+----+----+----+----+----MTextLabel
+----+----+----+----MLink
+----+----+----+----+----MButtonClose
+----+----+----+----+----MLinkBack
+----+----+----+----+----MOpenWindow
+----+----+----+----+----MGridHeaderLink
+----+----+----+----+----MLinkButton
+----+----+----+----+----MActionHyperLink
+----+----+----+----+----MImageLink
+----+----+----+----+----+----MImageLinkLabel
+----+----+----+----+----+----MImageButton
+----+----+----+----MListControl
+----+----+----+----+----MSelection
+----+----+----+----+----+----MComboBox
+----+----+----+----+----MMultiSelection
+----+----+----+----+----MMultiSelectionField
+----+----+----+----+----MOrderedList

+----+----+----+----+----MUnOrderedList
+----+----+----+----MContainer
+----+----+----+----+----MBaseGroup
+----+----+----+----+----+----MCheckBoxGroup
+----+----+----+----+----+----MRadioButtonGroup
+----+----+----+----+----+----MLinkButtonGroup
+----+----+----+----+----MVContainer
+----+----+----+----+----MHContainer
+----+----+----+----+----MBasePanel
+----+----+----+----+----+----MPanel
+----+----+----+----+----+----+----MActionPanel
+----+----+----+----MIndexedControl
+----+----+----+----MInputGridColumn
+----+----+----+----MInputGrid
+----+----+----+----MValidator
+----+----+----+----+----MRequiredValidator
+----+----+----+----+----MMASKValidator
+----+----+----+----+----MEmailValidator
+----+----+----+----+----MPasswordValidator
+----+----+----+----+----MCEPValidator
+----+----+----+----+----MPHONEValidator
+----+----+----+----+----MTIMEValidator
+----+----+----+----+----MCPFValidator
+----+----+----+----+----MCNPJValidator
+----+----+----+----+----MDATEDMYValidator
+----+----+----+----+----MDATEYMDValidator
+----+----+----+----+----MCompareValidator
+----+----+----+----+----MRangeValidator
+----+----+----+----+----MRegExpValidator
+----+----+----+----+----+----MIntegerValidator
+----+----+----+----MOption
+----+----+----MGridNavigator
+----+----+----MIFrame
+----+----+----MMioloStatus
+----+----+----MModuleHeader
+----+----+----MOptionListItem
+----+----+----MOptionList
+----+----+----+----MMenu
+----+----+----+----+----MNavigationBar
+----+----+----MPrompt
+----+----+----MStatusBar
+----+----+----MTabbedFormPage
+----+----+----MTheme
+----+----+----+----Theme
+----+----+----+----Theme
+----+----+----+----Theme
+----+----+----MThemeBox
+----+----+----MThemeElement
+----+----+----MTreeMenu
+----+----+----MWindow
+----+----+----+----MPage
+----+----+----Option
+----+----+----OptionGroup
+----+----+----MOptionGroup
+----+----+----MTable
+----+----+----MSimpleTable
+----+----+----+----MTableRaw
+----+----+----+----+----MTableXml
+----+----+----TableCell
+----+----+----TableRow

7. Guia de Upgrade para usuários da versão 1.0

Compatibilidade

Uma das grandes preocupações que sempre estiveram presentes durante as modificações e implementações de novas funcionalidades, foi a questão de compatibilidade. É ponto pacífico que os sistemas existentes devam ser compatíveis com a nova versão do MIOLO, porém, em função das profundas mudanças não é possível garantir 100% de compatibilidade sem, eventualmnete, serem necessárias algumas alterações.

Para compatibilizar um sistema existente, é necessário informar essa situação no arquivo de configuração do módulo, especificando o nível de compatibilidade:

<compatibility>**v1**</compatibility> =>compatibilidade com MIOLO 1.0
<compatibility>**b3**</compatibility> => compatibilidade com MIOLO 2 Beta3

O que mudou?

Nesta versão, houve uma convergência de 2 versões do MIOLO: a oficial da Solis/Univates e a versão da UFJF, que já vem trabalhando com o framework desde a época do treinamento ocorrido em final de 2002.

Diferente do que acontecia nas versões anteriores do MIOLO, a versão 2.0 já começa a implementar a proposta de que não tenhamos mais uma variável global $MIOLO. Este objetivo ainda não foi plenamente alcançado, mas já houve um grande progresso nesse sentido.

Com a versão, boa parte da programação passa a ser feita diretamente na classe de formulários.

O arquivo startup.inc passa a ser support.inc, pois a partir de agora, a funcionalidade de inicializar o $MIOLO não está mais neste arquivo. Este arquivo, além da função de tradução _M(), possui também a função Import() e a função __autoload(). Essas funções são responsáveis pela inclusão automática de arquivos de classes conforme namespaces definidos no arquivo <miolo>/classes/etc/autoload.xml

Arquivos de configuração

A partir esta versão, os arquivos de configuração (tanto do MIOLO quanto dos módulos) passa a ser definido em formato XML. Os arquivos de configuração dos módulos deixa de ser chamado nome_modulo.conf e passa a se chamar module.conf.

As configurações a serem definidas continuam sendo as mesmas, mas como acontecia antes, não se limitando a elas.

Para acessar uma configuração, utilize o método $MIOLO->GetModuleConf.

Estrutura dos diretórios

Basicamente, a estrutura do MIOLO continua sendo a mesma, porém, com a adição de novos diretórios para refletir a reestruturação e desmembramento das classes em arquivos novos.

A nova estrutura de diretórios, bem como a separação e organização dos arquivos visa organizá-los baseado em sua funcionalidade: serviços, persistência, segurança, utilitários,...

Veja também o tópico # Estrutura de diretórios

Novidades

Desde as versões iniciais do MIOLO sempre se teve em mente a idéia de criar uma hierarquia de objetos que facilitasse tanto a compreensão do framework quanto a própria programação, além de possibilitar a fácil importação de formulários criados em software de designers como Glade, QT Designer, entre outros. Nesse sentido uma das mudanças mais significativas está no fato de ter-se criado uma "componentização" dos objetos do MIOLO, o que começa a trazer esse objetivo mais próximo da realidade.

Graças a essa componentização, agora também é possível adicionar eventos aos objetos, de forma a ampliar as capacidades de programação no MIOLO.

Veja a nova hierarquia dos controles/classes no tópico #5.Hierarquia de controles

Também o tema traz substancias mudanças. Na versão anterior do MIOLO, a renderização sempre era feita em HTML e o responsável por isso era o método Generate da classe. A partir de agora, também é possível que a página seja renderizada em outros formatos. Veja mais informações no tópico #4.Interface com o usuário

Forms

- Ao enviar um formulário, é chamado o evento default nomeBotao_click
- Deve existir um método CreateFields, responsável pela criação/adicão dos campos ao formulário.
- SetIcon define o ícone do formulário

Formfields

De forma geral, a função e sistemática dos formfields continuam sendo os mesmos. Existem modificações a nível de framework que de uma ou outra forma acabam afetando os mesmos. Uma alteração importante, é que a ordem dos parâmetros para o construtor da classe foi modificada e programas escritos para a nova versão do MIOLO devem observar essas modificações.

Conforme descrito anteriormente, existe um "layer" de compatibilidade, que permite que os programas criados para a versão anterior funcione com a nova versão. Esse layer é o responsável por traduzir a solicitação de um formfield na forma antiga para a nova.

A ordem dos parâmetros

- HiddenField
- MultiTextField3
-
- ActionPanel
-

Validators

A partir desta versão, os validadores passam a ser classes e não mais métodos de uma classe única. Dessa forma, ao invés de termos o método MASKValidator, temos agora a classe MAKSValidator extends Validator. As classes de validadores estão localizadas no arquivo <miolo>/ui/controls/validator.class

Também foi modificada a forma como os validadores são adicionados aos campos. Ao invés de passar o mesmo como parâmetro, agora deve-se utilizar o método AddValidator, da seguinte forma:

```
$this->AddValidator( new RequiredValidator('edtFieldName') );
```

Outra maneira, é criar um array que contenha os validadores e adiconar utilizando o método SetValidators:

```
$validators = array( new RequiredValidator('edtFieldName'),
                     new MaskValidator('edtFieldName1','','99aa99aa','required'),
                     new EmailValidator('edtFieldName2','','required')
                     );
$this->SetValidators( $validators );
```

A nova versão do MIOLO traz também uma série de novos validadores. Veja abaixo os validadores disponíveis:

- RequiredValidator: para indicar que se trata de um campo necessário
- MASKValidator: validação de máscara
- EmailValidator: validação de e-mail
- PasswordValidator:
- CEPValidator: campo para máscara de CEP
- PHONEValidator: validador de telefone (somente caracters numéricos e ( ) -)
- TIMEValidator: validador de hora
- CPFValidator: validador de CPF
- CNPJValidator: validador de CNP
- DATEDMYValidator: data no formato Dia/Mês/Ano
- DATEYMDValidator: data no formato Ano/Mês/Dia
- CompareValidator: comparação de valores, por exemplo, >= 10
- RangeValidator: validação de intervalo de strings, números ou datas
- RegExpValidator: validação baseada numa expressão regular
- IntegerValidator: validação para apenas dígitos numéricos

Link Controls

São classes que fornecem componentes para a criação de links:

- Hyperlink: basicamente um link em formato de texto
- ImageLink: gera uma imagem com um link
- ImageLinkLabel: gera um link imagem com um texto

Label Controls

Componentes para gerar texto formatado ou não:

- PageComment
- Separator
- FieldLabel
- Label
- Text
- TextHeader
- TextLabel

Button Controls

- MButton:
- MLinkButton: botão tipo texto, com uma ação definida em javascript.
- MImageButton: botão que gera uma imagem com uma ação definida em javascript.
- MInputButton:

Listings X Grids

Business

GetData e SetData