MIOLO 2.0
UserGuide

Vilson Gartner
vilson@miolo.org.br

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

versão do documento: 1.2
10/01/2008

Sumário

1.Introdução

O que é o MIOLO?

O MIOLO é um framework para criação de sistemas de informação acessíveis via WEB escrito em PHP5. O MIOLO utiliza scripts javascript e conceitos de POO (Programação Orientada a Objetos), gerando páginas HTML e arquivos PDF. Com um projeto modular, baseado em componentes, e uma arquitetura em camadas, o MIOLO atua como "kernel" de todos os sistemas criados. Favorecendo a reultilização, os vários sistemas 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 uma metodologia de codificação para que os resultados esperados sejam obtidos de forma simples e rápida.

O pré-requisito para uso do MIOLO é o conhecimento de programação com PHP e de POO. Os sistemas desenvolvidos com o MIOLO seguem algumas regras básicas que devem ser conhecidas, como as definições de classes/forms/handlers dos módulos, definição dos arquivos de configuração (localização dos programas, componentes, bancos de dados, temas, etc...), 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.

Resumidamente, 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 uma arquitetura em camadas (figura 1), possibilitando a implementação do padrão MVC (Model-View-Controller) (figura 2).

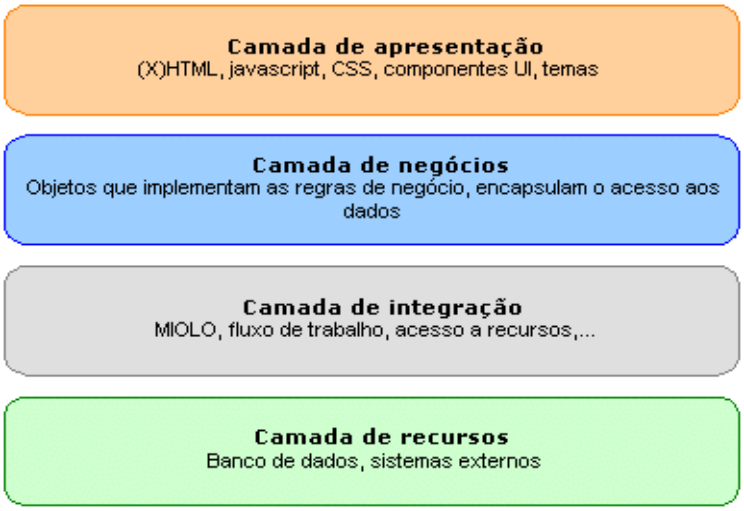


Figura 1 - Camadas

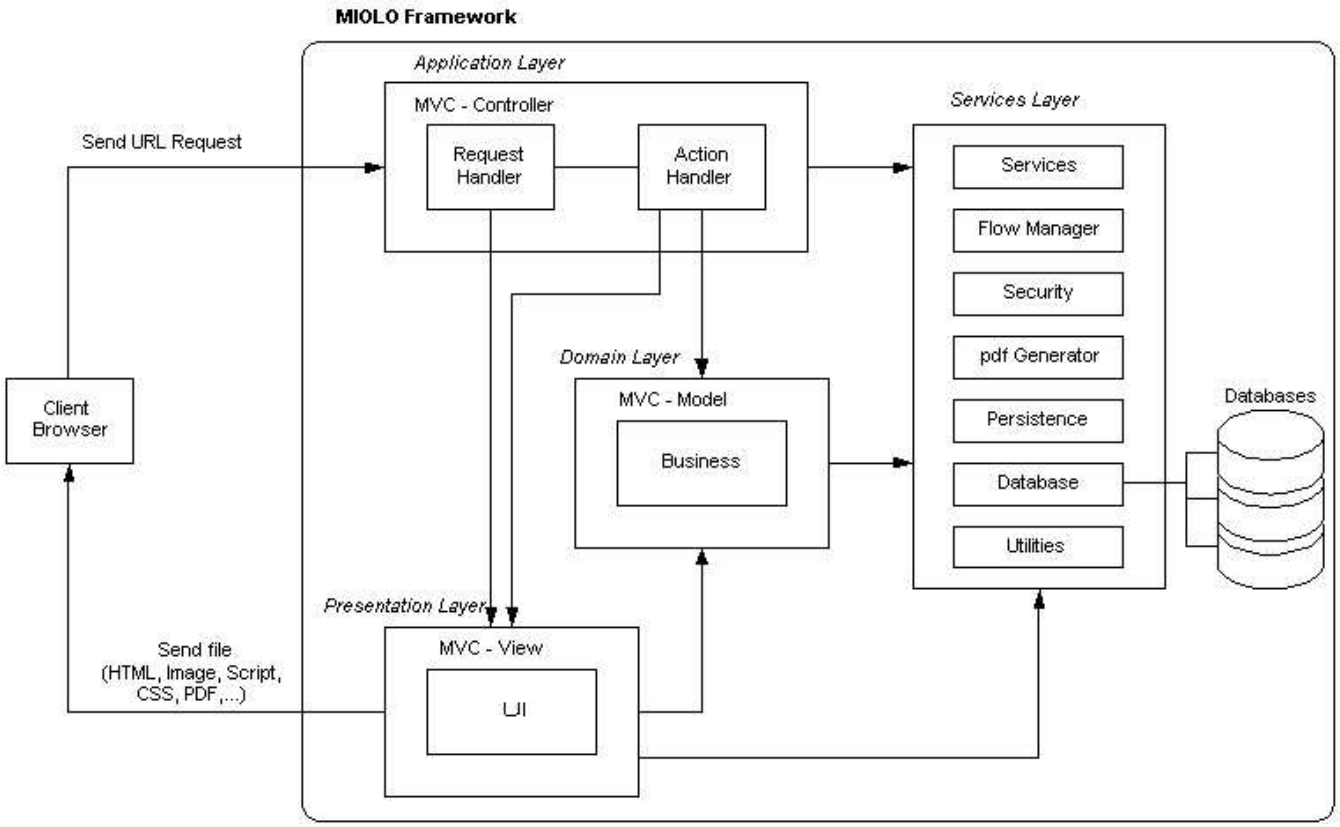


Figura 2 – Padrão MVC

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

Classes do framework responsáveis pela geração de arquivos PDF, 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). Representam a camada View do MVC.

Handlers – camada de integração

Classes que representam a parte funcional da aplicação, criadas pelo desenvolvedor para fazer o tratamento dos dados enviados pelo cliente. Definem o fluxo de execução e implementam os casos de uso. Os handlers podem acessar a camada de negócios para obter/gravar dados e usam a camada UI para definir a saída para o cliente. Estão localizadas no diretório handlers de cada módulo. Representam a camada Controller do MVC.

Business – camada de negócios

Classes criadas pelo desenvolvedor para representar o domínio da aplicação (as regras de negócio). São usadas pelas camadas UI e pelos handlers. A definição do nível de granularidade dos objetos de negócios deve ser feita pelo desenvolvedor. Estes objetos podem usar outros objetos de negócio ou acessaram o banco de dados através da camada de Persistência ou camada DAO. Representam a camada Model do MVC.

Database – camada de recursos

Classes do framework responsáveis por abstrair o acesso às bases de dados, tornando as classes da camada Business independentes do SGBD usado. Implementa uma camada DAO (Data Access Objects) e uma camada de Persistência de Objetos.

Utils e Services – camada de recursos

Classes do framework responsáveis por oferecer recursos e funcionalidades necessárias tanto ao framework quanto às aplicações dos usuários. Encapsulam o acesso a recursos da linguagem ou do sistema operacional. Seguindo a filosofia de reutilização, diversas bibliotecas e componentes open-source são encapsulados dentro do framework.

3.Diretórios

Após a instalação do MIOLO (ver arquivo INSTALL, na distribuição) são criados os vários subdiretórios (listados abaixo) que armazenam as classes do framework.

```
-----<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
         |         |                +--- miolo2
         |         |          +--- system
         |         |          +--- clean
         |         |          +--- .....
         |         +--- utils
         |
         +--- docs
         +--- etc
         |         +--- miolo.conf
         |         +--- passwd.conf
         |         +--- mkrono.conf
         +--- html
         |         +--- downloads
         |         +--- images
         |         +--- reports
         |         +--- scripts
         +--- locale
         +--- modules
         |         +--- admin
         |         +--- tutorial
         |         +--- helloworld
         |         +--- hangman
         |         +--- modulo1
         |         +--- ....
         +--- 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.
- etc/passwd.conf – arquivo para armazenamento de senhas para acesso aos bancos de dados.
- Etc/mkrono.conf – arquivo de configuração da biblioteca Mkrono.
- 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 (padrão Front Controller), 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/*.js – Diretórios e arquivos com as funções/bibliotecas 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 Facade) 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 (module.conf - 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 MTheme) 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 em <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>/handlers) 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.
Exemplo do código de um handler:

```
<?php
    // Get access to the User Interface classes/methods.
    // We need getUI to be able to call getForm
    $ui = $MIOLO->GetUI();

    // instantiate the form frmMain located in
    // the modules/helloworld/helloworld
    $form = $ui->GetForm('helloworld','frmMain');

    // set the $form as the theme content
    $theme->SetContent($form);
?>
```

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

O arquivo de configuração do Miolo é um arquivo XML e está dividido em várias seções:

- home: localização de arquivos e urls
- namespace: definição lógica da localização de arquivos
- theme: definições relativas ao tema usado
- options: opções gerais
- i18n: definições relativas à internacionalização
- mad: Miolo Administration Module (MAD)
- login: definições relativas à autenticação de usuários
- session: definições relativas ao tratamento da sessão
- db: definições dos DataSources
- logs: definições relativas à geração de arquivos de logs

A seguir é apresentado um arquivo de configuração como exemplo:

```
<?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>
▪   URL base para execução de relatórios com JasperReports
       <url_jasper>http://127.0.0.1:8080/MioloJasper/Report</url_jasper>
    </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>
```
- *Definição das ações possíveis nas janelas*

```
        <options>
            <close>true</close>
            <minimize>true</minimize>
            <help>true</help>
            <move>true</move>
        </options>
    </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>
```
  - *Opções gerais*

```
    <options>
```
- *Define o modulo que sera usado por default (quando não for informado na URL)*

```
        <startup>common</startup>
```
- *Define se os parâmetros na URL serão criptografados e com qual senha*

```
        <scramble>0</scramble>
        <scramble.password>password</scramble.password>
```
- *Define o script base para execução*

```
        <dispatch>index.php</dispatch>
```
- *Define o formato da URL:*
  - *0: >&...*
  - *1: ...>*

```
        <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 (Tigra menu)*
  - *3: utiliza um menu "suspenso" com DHTML (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 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 exibe imagem no browser durante o carregamento da página*

```
        <loading>
            <show>true</show>
            <generating>true</generating>
        </loading>
```
- *Define se serão usadas url relativas (false) ou diretas (true) para acesso a imagens e arquivos do*

tema (folhas de estilo CSS)

```
        <performance>
            <uri_images>true</uri_images>
            <uri_themes>true</uri_themes>
        </performance>
    </options>
```

- *Opções de internacionalizaçao*

```
  <i18n>
    <locale>c:/miolo/locale</locale>
    <language>pt_BR</language>
  </i18n>
```

- *Define qual o módulo será usado para administração do MIOLO (MAD) 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, mensagens e comandos SQL*

```
        <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, mssql, odbc*

```
            <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>
```

- *Definições para autenticação de usuários:*
  -

```
    <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 no banco de dados*
- *MauthDbMD5: senha "criptografada" com MD5 no banco de dados*

```
        <class>MAuthDb</class>
```

- *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 miolo_user*

- *true false false usuário deve estar cadastrado em miolo_user*
- *false true false não é necessário cadastro no miolo_user*
- *true true true usuario pre-definido deve existir no miolo_user*
- *false true true usuario pre-definido não é necessário no miolo_user*

```
        <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

Este processo é ilustrado nos módulos de exemplo distribuídos junto com o Miolo (módulos helloworld e hangman).

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 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:

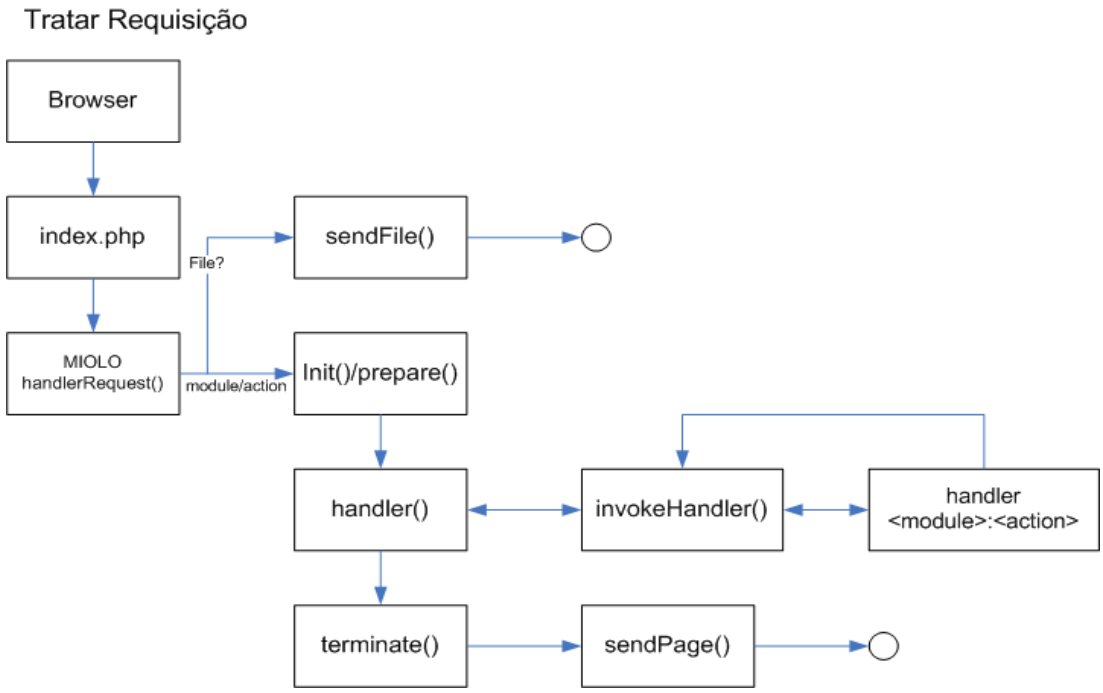
```
Ex: http://host.dominio/index.php/common/html/images/save.png
```

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) possui uma estrutura de diretórios pré-determinada:

```
-----<diretório-base>(p.ex. /usr/local/miolo)
       |
      +--- modules
        |       +--- module1
        |       |              +--- classes (ou db)
```

```
|       |             +--- 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, podendo ser usadas diretamente em todos os handler, ou dentro dos métodos das classes através da declaração **global**.

- **$MIOLO**: acesso a instância da classe principal do framework
- **$page**: acesso ao objeto MPage
- **$context**: acesso ao objeto MContext
- **$theme**: acesso ao objeto Mtheme em uso
- **$auth**: acesso ao objeto MAuth (autenticação)
- **$perms**: acesso ao objeto MPerms (permissões)
- **$session**: acesso ao objeto MSession (sessão atual)
- **$state**: acesso ao objeto MState (variáveis de estado)
- **$menu**: acesso ao objeto Menu (menu principal)
- **$log**: acesso ao objeto MLog (módulo admin)
- **$navbar**: acesso ao objeto navbar (barra de navegacao)
- **$module**: nome do módulo do handler em execução – ex: tutorial
- **$action**: path do handler em execução – ex: main:varglobais
- **$item**: campo item da url atual
- **$self** (*deprecated*): path do handler em execução – ex: main:varglobais
- **$url**: url completa do handler em execução:
  - http://miolo.org.br/index.php?module=tutorial&action=main:varglobais

Menu Principal

O arquivo <miolo>/modules/main_menu.inc define um menu principal, acessível por todos os módulos. Este menu pode ser acessado em qualquer ponto da execução através da atribuição:

```
$menu = $theme->getMainMenu();
```

Módulo MAD

O módulo MAD (Miolo Administration Module) é definido no arquivo de configuração miolo.conf. Toda instalação do Miolo deve possuir um módulo MAD. Seu objetivo é centralizar as taferas administrativas dos sistemas, evitando que elas sejam replicadas em cada módulo. O gerenciamento do cadastro de usuários, grupos, transações, direitos de acesso e logs é feito através deste módulo.

Na distribuição do Miolo é oferecido o módulo ADMIN com esta finalidade. Naturalmente, para o desenvolvimento de aplicações mais complexas, o desenvolvedor deve construir seu próprio módulo de administração e indicar este módulo no arquivo miolo.conf.

O Miolo oferece acesso às classes básicas de administração, através do método

```
$MIOLO->getBusinessMAD('<className>');
```

Para permitir que o desenvolvedor crie suas prórpias classes e ao mesmo tempo utilize a funcionalidade do Miolo, os nomes das classes básicas devem ser definidos no arquivo miolo.conf: access, group, log, session, transaction, user. Assim, independente do nome da classe de usuários definido pelo desenvolvedor, ela pode ser acessada com

```
$MIOLO->getBusinessMAD('user');
```

Base de dados MAD

A base de dados MAD (localizada no módulo ADMIN em <miolo>/modules/admin/sql/admin.sqlite) é fornecida como um banco de dados SQLite. Com visto anteriormente, ela pode ser customizada em cada instalação (ou seja, cada instalação pode optar por desenvolver sua própria base de dados de administração), desde que seja respeitada a interface definida pelo framework (as interfaces estão disponíveis em <miolo>/classes/interfaces).

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);
```

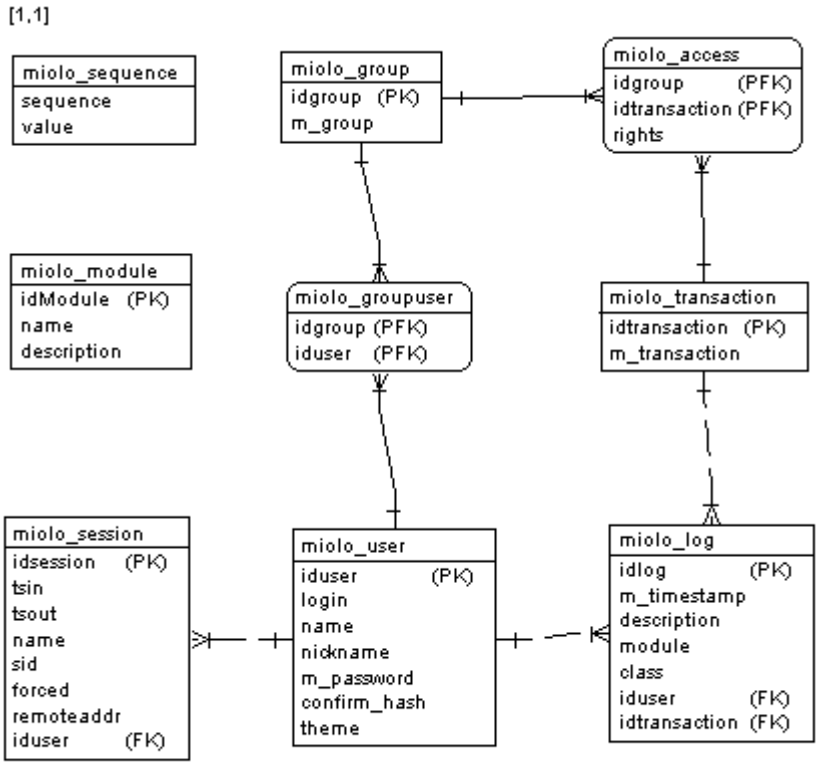


Figura 4 – Base de dados MAD

Módulo COMMON

O módulo COMMON é, por default, o módulo a ser executado caso nenhum seja especificado explicitamente na URL. O objetivo do módulo COMMON é justamente permitir o acesso aos demais módulos do sistema.

Logs

A geração de arquivos de logs é feita automaticamente pelo framework, de acordo com as configurações no arquivo miolo.conf. O registro das mensagens pode ser feito de 3 maneiras:

- file: as mensagens são armazenadas em arquivos em disco
- db: as mensagens são armazenadas no banco de dados MAD
- socket: as mensagens são enviadas via rede (TCP/IP) para um socket especificado

No caso de armazenamento em disco, os arquivos ficam localizados em

```
<miolo>/var/log
```

É possível definir 3 níveis de log:

- nenhum log é gerado (level = 0)
- log apenas das mensagens de erros (level = 1)
- log de mensagens, erros e comandos SQL (level = 2)

O desenvolvedor pode registrar as mensagens no log através dos métodos da classe Miolo:

- *Registra a execução de um comando SQL; $force=true obriga a geração da mensagem em disco; $conf é usado como prefixo do arquivo*

```
$MIOLO->logSQL($sql,$force=false,$conf='?')
```

- *Registra uma mensagem de erro; $conf é usado como prefixo do arquivo*

```
$MIOLO->logError($error, $conf = 'miolo')
```

- *Registra uma mensagem qualquer no log*

```
$MIOLO->logMessage($msg)
```

Trace (debug)

O mecanismo de trace permite ao programador acompanhar a execução do programa, facilitando a detecção e correção de bugs. A mensagem de trace é armazenada no arquivo de log. Os seguintes métodos podem ser usados pelo programador:

- *Registra a mensagem de trace; $file é um nome de arquivo para aparecer na mensagem e $line é um número de linha*

```
$MIOLO->trace($msg,$file='',$line=0)
Exemplo: $MIOLO->trace(“mensagem”,__FILE__,__NUMBER__);
```

- *Exibe a stack de execução do script*

```
$MIOLO->trace->traceStack()
```

5.Interface com o usuário

Temas

Um "tema", no contexto do framwork MIOLO, corresponde 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. O tema portanto deve definir não apenas como a página será "dividida", mas também como os controles html serão renderizados.

A localização das folhas de estilo e a classe Theme é definida no arquivo miolo.conf e pode estar em:

- tema global – opção perfomance.uri_themes = false

```
<miolo>/classes/ui/themes/<tema>/theme.class
<miolo>/classes/ui/themes/<tema>/*.css
<miolo>/classes/ui/themes/<tema>/images/*
```

- tema global – opção perfomance.uri_themes = true

```
<miolo>/classes/ui/themes/<tema>/theme.class
<miolo>/html/themes/<tema>/*.css
<miolo>/html/themes/<tema>/images/*
```

- tema específico para o módulo

```
<miolo>/modules/ui/themes/<tema>/theme.class
<miolo>/modules/ui/themes/<tema>/*.css
<miolo>/modules/ui/themes/<tema>/images/*
```

Por exemplo, uma estrutura de tema é a seguinte (tema "system"):

Figura 5 – Estrutura do tema "system"

Cada uma das áreas (top, menu, navigation, content e statusbar) 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.

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).

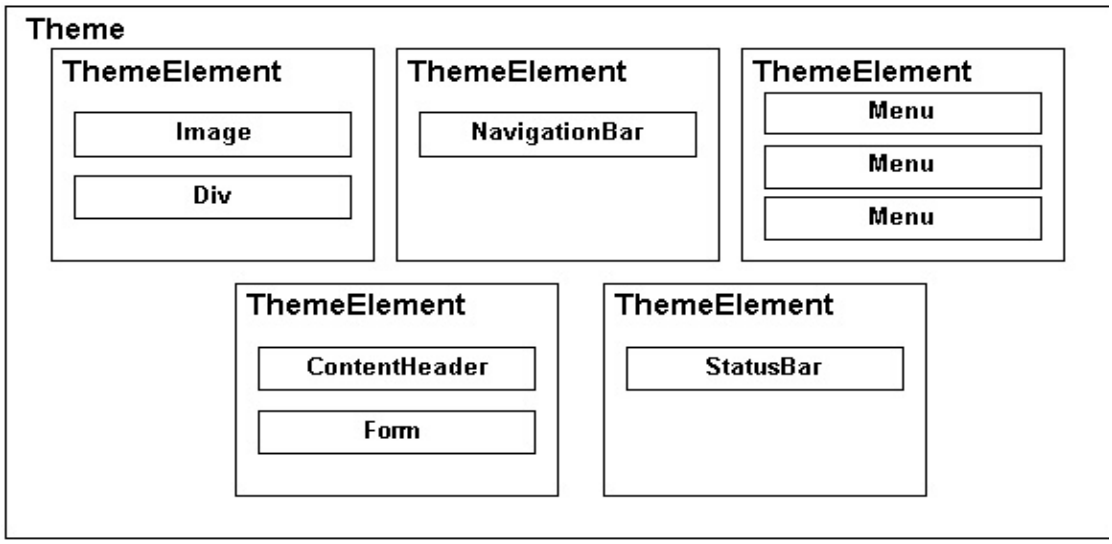


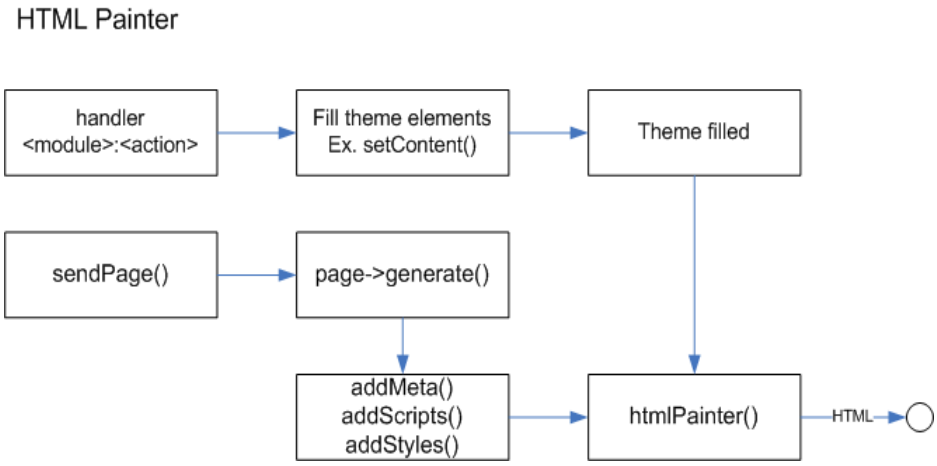
Figura 6 – Estrutura lógica do tema

Além dos elementos do tema, existe o conceito de "layout". Um layout define quais os elementos do tema serão renderizados para uma página específica. Dentro da classe Theme devem ser definidos os métodos para geração de layouts específicos (default, lookup, popup, DOMPdf). Cada um destes métodos define quais elementos serão renderizados. A estrutura da classe Theme é a seguinte (tema "system"):

```
Arquivo: <miolo>/classes/ui/themes/system/theme.class
class ThemeSystem extends MTheme
{
        function __construct()
        {
            parent::__construct('system');
        }

        function init()
        function generate()
        function generateDefault()
        function generateLookup()
        function generatePopup()
        function generateDOMPdf()
}
```

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 definidos para cada tema. Assim, a geração de uma área específica do tema (title, navbar, menus, content, statusbar) pode ser customizada ou até mesmo mesmo omitida, de acordo com a programação da classe Theme.



WebForms

Como visto, cada url será tratada pelo framework, gerando uma página html ou um arquivo PDF. A geração da página é encapsulada em várias classes (Mpage, Mtheme, HTMLPainter). Cada página é gerada após o processamento da seqüência de handlers e constitui-se em um único formulário html (tag <form>), mesmo que vários controles estejam presentes na página. É importante observar, portanto, que os controles colocados na página devem ter nomes (id) 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 definição 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 do formulário e os tratadores dos eventos (eventHandlers - tipicamente as funções para tratar os eventos OnClick dos botões de submit). Este arquivo deve estar localizado eno diretório

```
<miolo>/modules/<module>/forms
```

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, 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, usa-se os métodos do objeto MState.

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 (quando é usado o método GET, ao invés do POST). Um evento na URL é definido através da variável event (neste caso podem ser passados parâmetros para os eventos):

```
http://.../index.php?module=....&action=....&item=....&event=btnPost:click
```

Eventos a serem tratados no lado servidor podem ser gerados em javascript através do método

```
miolo.doPostBack(eventTarget, eventArgument, formSubmit)
```

O código a seguir apresenta a definição básica de um formulário:

```
<?php
class frmMain extends MForm
{
    function __construct()
    {
        // call the parent constructor
        parent::__construct( 'Hello World!' );

        // call the event's handler
        $this->eventHandler();
    }

    function createFields()
    {
        // array of fields
        $fields  = array(
           new MTextField( 'myMsg', '', _M('Message'), 30, _M('Your message to the world') ),
           new MTextField( 'myName', '', _M('Name'), 20, _M('Your name') )
```

```
        );
        // define buttons

        $buttons = array(
           new MButton( 'btnHello', _M('Click Me!') ),
           new MButton( 'btnReset', _M('Reset') ),
        );

        // insert the components on the form
        $this->setFields( $fields );
        $this->setButtons( $buttons );
    }

    // this method is called when the form is submitted (click on btnHello button)
    function btnHello_click( $sender )
    {
        // set the visible attribute of the fields to false
        $this->setFieldAttr( 'myMsg', 'visible', false );
        $this->setFieldAttr( 'myName', 'visible', false );
        $this->setFieldAttr( 'btnHello', 'visible', false );

        // add fields (labels) to the form
        $this->addField( new MLabel( _M('Hello World!') ) );

        // this labels contains the text entered in the input fields
        $this->addField( new MLabel( $this->myMsg->GetValue() ) );
        $this->addField( new MLabel( $this->GetFieldValue('myName') ) );
    }
}

?>
```

Controles (widgets)

Os controles visuais são classes programadas em PHP e que encapsulam controles html (ou controles javascript). Cada controle tem um método generate(), responsável por gerar o código html/javscript correspondente. A renderização está encapsulada na classe MHTMLPainter (<miolo>/classes/ui/painter/mhtmlpainter.class).
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
+----+----+----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
+----+----+----MDHTMLMenu2
+----+----+----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
+----+----+----+----+----+----MLookupFieldValue
+----+----+----+----+----MMultiTextField2
+----+----+----+----+----+----MMultiTextField3
+----+----+----+----+----+----MmultiTextField4
+----+----+----+----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
+----+----+----+----MTextTable
+----+----+----MGridNavigator
+----+----+----MIFrame
+----+----+----MOptionListItem
+----+----+----MOptionList
+----+----+----+----MMenu
+----+----+----+----+----MNavigationBar
+----+----+----MPrompt
+----+----+----MStatusBar
+----+----+----MTabbedFormPage
+----+----+----MTheme
+----+----+----MThemeBox
+----+----+----MThemeElement
+----+----+----MTreeMenu
+----+----+----Mpage
+----+----+----Option
+----+----+----OptionGroup
+----+----+----MOptionGroup
+----+----+----MTable
+----+----+----MSimpleTable
+----+----+----+----MTableRaw
+----+----+----+----+----MTableXml
+----+----+----TableCell
+----+----+----TableRow
```

6.DAO - Camada de abstração de acesso a dados

A camada DAO (Data Access Objects) no Miolo tem por objetivo fazer a abstração do acesso a bancos de dados relacionais, encapsulando as diversas extensões fornecidas 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, que é descrita neste documento.
A implementação atual tem as seguintes características:

- O mecanismo de acesso a dados fornecido pelo PHP é encapsulado, permitindo uma única interface de programação (API).
- 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.
- Suporte a transações, utilizando recursos nativos do banco de dados sendo acessado.
- Abstração de resultados de consultas (queries) em ResultSets, permitindo operações como travessia (browse), paginação, filtragem e ordenação do resultado de uma consulta.

A figura 1 fornece uma visão geral da estrutura de classes da camada DAO. As classes MConnection, MQuery, MTransaction, MIdGenerator e MSQLJoin são especializadas para cada banco de dados (PostgreSql, MySQL, Oracle, SQLite, Interbase, etc) a fim de implementar as particularidades de sintaxe e encapsular as funções PHP de acesso aos dados.

Configuração

As bases de dados que serão acessadas pelos módulos do MIOLO são configuradas no arquivo XML <miolo>/etc/miolo.conf. As entradas neste arquivo seguem a seguinte sintaxe:

```
<db>
   <base_name>
       <system>dbms</system>
       <host>address</host>
       <name>db_name</name>
       <user>username</user>
       <password>password</password>
   </base_name>
</db>
```

<base_name>: nome da base de dados, conforme será referenciada pelos módulos do MIOLO. Um mesmo banco de dados pode ser referenciado por nomes diferentes, para implementar restrições de segurança, por exemplo.
<dbms>: o Sistema Gerenciador de Banco de Dados. O MIOLO implementa drivers para: postgres, mysql, oracle, firebird, sqlite, mssql, odbc
<address>: endereço do servidor executando o DBMS (geralmente um endereço IP).
<db_name>: nome do banco de dados, conforme referenciado pelo DBMS, ou pelo software cliente.
<username>: identificação do usuário para acesso ao banco de dados.
<password>: senha do usuário para acesso ao banco de dados.

Classes DAO

Embora constituída por várias classes, o usuário necessita de acesso a apenas alguns poucos métodos oferecidos pela camada DAO. Nesta seção apresentamos a definição dos principais métodos usados.
Alguns conceitos são importantes para compreensão do funcionamento da camada DAO:

- **Campo:** corresponde a uma coluna de uma tabela (ou visão) no banco de dados.
- **Registro:** corresponde a uma linha de uma tabela (ou visão) no banco de dados.
- **ResultSet:** corresponde a um array bidimensional, retornado como resultado de uma consulta (query) SQL. Cada linha corresponde a um registro e cada coluna corresponde a um campo. As colunas têm índice numérico a partir de 0, e as linhas são ordenadas de acordo com o comando SQL SELECT.
- **DataSet:** corresponde a uma instância da classe MDataSet, usada internamente pela camada de acesso a dados, para encapsular o acesso a um ResultSet.

O procedimento básico para acesso a uma base de dados tem a seguinte seqüência de ações:

1. Instanciar um objeto MDatabase (via $MIOLO->getDatabase), que encapsula a conexão e o acesso ao banco de dados;
2. Instanciar um objeto MSql, que encapsula um comando SQL;
3. Realizar a consulta, criando um objeto MQuery, ou executar o(s) comando(s) SQL;
4. Trabalhar com o ResultSet, através do objeto MQuery.

Exemplos

Para efeito dos exemplos, será usada a base de dados "admin", as tabelas miolo_acesso, miolo_transacao e miolo_sistema e a sequence seq_miolo_transacao. Estas tabelas têm a seguinte estrutura:

miolo_acesso (idgrupo (PK), idtrans(PK), direito)
miolo_transacao (idtrans (PK), transacao, idsistema (FK))
miolo_sistema (idsistema (PK), sistema)

Exibir todos os campos de todos os registros

Código:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('admin'); // instancia um objeto Database
$sql = new sql('*','miolo_transacao'); // instancia um objeto SQL com os dados para a consulta
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
$n = $query->GetRowCount(); // obtem o número de registros retornados
$result = $query->result; // trabalha com o ResultSet
for ($i=0; $i < $n; $i++)
{
   echo "#$i - " . $result[$i][1] . '<br>'; // acessa o campo 'transacao'
}
```

Consulta com critério de seleção (WHERE)

Código:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('admin'); // instancia um objeto Database
$sql = new sql('*','miolo_transacao', 'idtrans = 100'); // instancia um objeto SQL
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
$n = $query->GetColumnCount(); // obtem o número de colunas
$result = $query->result; // trabalha com o ResultSet
for ($i=0; $i < $n; $i++)
{
   echo $result[0][$i] . '<br>'; // acessa, no primeiro registro, cada campo retornado
}
```

Consulta com critério de seleção composto

Código:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('admin'); // instancia um objeto Database
$sql = new sql('*','miolo_transacao', "(idtrans = 100) AND (transacao LIKE 'S%'");
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Código alternativo:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('admin'); // instancia um objeto Database
$sql = new sql('*','miolo_transacao'); // instancia um objeto SQL
$SQL->SetWhere("(idtrans = 100) AND (transacao LIKE 'S%'");
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Código alternativo:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('admin'); // instancia um objeto Database
$sql = new sql('*','miolo_transacao'); // instancia um objeto SQL
$SQL->SetWhere('idtrans = 100');
$SQL->SetWhereAnd("transacao LIKE 'S%'");
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Consulta com parâmetro único

Código:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('admin'); // instancia um objeto Database
$sql = new sql('*','miolo_transacao', 'idtrans = ?'); // instancia um objeto SQL
$sql->SetParameters(100); // indica o valor do parâmetro
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Consulta com múltiplos parâmetros

Código:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('admin'); // instancia um objeto Database
$sql = new sql('*','miolo_transacao', “(idtrans = ?) AND (transacao LIKE ?)”);
$sql->SetParameters(100, ‘A%’); // indica os valores dos parâmetros
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Código alternativo:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('admin'); // instancia um objeto Database
$sql = new sql('*','miolo_transacao', “(idtrans = ?) AND (transacao LIKE ?)”);
$sql->SetParameters(array(100, ‘A%’)); // indica os valores dos parâmetros como array
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Consulta com ordenação

Código:

```
$db = $MIOLO->GetDatabase('admin');
$sql = new sql('*','miolo_transacao','','transacao');
$query = $db->GetQuery($sql);
```

Consulta com inner join entre 2 tabelas e uso de aliases

Código:

```
$db = $MIOLO->GetDatabase('admin');
$sql = new sql('s.sistema, t.transacao','','','s.sistema,t.transacao');
$sql->SetJoin('miolo_sistema s','miolo_transacao t','(s.idsistema=t.idsistema)');
$query = $db->GetQuery($sql);
```

Consulta com left join entre 3 tabelas, uso de aliases, função de grupo e cláusula GROUP BY

Código:

```
$db = $MIOLO->GetDatabase('admin');
     $sql = new sql('s.sistema, t.transacao, count(a.idgrupo)','','', 's.sistema,t.transacao',
's.sistema,t.transacao');
$sql->SetLeftJoin('miolo_sistema s','miolo_transacao t','(s.idsistema=t.idsistema)');
$sql->SetLeftJoin('miolo_transacao t','miolo_acesso a','(t.idtrans=a.idtrans)');
$query = $db->GetQuery($sql);
```

Consulta com uso da cláusula HAVING

Código:

```
$db = $MIOLO->GetDatabase('admin');
  $sql = new sql('s.sistema, count(t.idtrans)','','','s.sistema','s.sistema', '(count(t.idtrans) >
3)');
$sql->SetLeftJoin('miolo_sistema s','miolo_transacao t','(s.idsistema=t.idsistema)');
$query = $db->GetQuery($sql);
```

Consulta com subquery

Código:

```
$db = $MIOLO->GetDatabase('admin');
$sql = new sql('a.idgrupo','miolo_acesso a');
$sql->SetWhere('a.idtrans IN ?');
$sqlx = new sql('t.idtrans','miolo_transacao t','(t.idsistema = 1)'); // Sql para a subquery
$sql->SetParameters(':(' . $sqlx->Select() . ')'); // gera o SQL SELECT como parâmetro
$query = $db->GetQuery($sql);
```

Observar neste exemplo o uso do caracter ‘:’ antes do valor do parâmetro. O uso do ‘:’ impede que sejam acrescentadas as aspas ao valor do parâmetro.

Acessando uma linha específica

Código:

```
$db = $MIOLO->GetDatabase('admin');
$sql = new sql('*','miolo_transacao');
$query = $db->GetQuery($sql);
$n = $query->GetRowCount();
$query->MoveTo(4);
```

Navegação (browse) no ResultSet

Código:

```
$db = $MIOLO->GetDatabase('admin');
$sql = new sql('*','miolo_transacao');
$query = $db->GetQuery($sql);
```

```
      echo 'Move to First: <br>';
//
// acessa o primeiro registro
//
      $query->MoveFirst();
      $row = $query->GetRowValues();
      foreach($row as $f)
      {
         echo $f . ' - ';
      }
      echo '<br><br>';
//
// acessa o último registro
//
      echo 'Move to Last: <br>';
      $query->MoveLast();
      $row = $query->GetRowValues();
      foreach($row as $f)
      {
         $text .= $f . ' - ';
      }
      echo '<br><br>';
//
// browse
//
      echo 'Transverse: <br>';
      $query->MoveFirst();
      while (!$query->eof())
      {
         echo $query->fields('transacao') . '<br>';
         $query->MoveNext();
      }
```

Contando registros

Código:

```
      $db = $MIOLO->GetDatabase('admin');
      $sql = new sql('*','miolo_transacao');
      $n = $db->Count($sql);
      $command = $sql->Select();
      $echo "$command => #records: " . $n . '<br><br>';
      $sql = new sql('*','miolo_transacao','(idtrans > 100)');
      $n = $db->Count($sql);
      $echo "$command => #records: " . $n . '<br><br>';
```

Usando range para paginação

Código (com objeto QueryRange):

```
      $db = $MIOLO->GetDatabase('admin');
      $sql = new sql('*','miolo_transacao');
      $n = $db->Count($sql);  // total de registros
      $totalpages = (int) $n / 5;  // cada página com 5 registros
      for($page=1; $page <= $totalpages; $page++)
      {
        $range = new MQueryRange($page,5);
        $result = $db->QueryRange($sql->Select(), &$range);
        $n = $range->total;
        echo "Page: $page  [$n records]: <br>";
        echo '          #rec - transação' . '<br>';
        for ($i=0; $i < $n; $i++)
        {
           echo "         #$i - " . $result[$i][1] . '<br>';
        }
        echo "<br>";
      }
```

Código (com objeto Sql):

```
      $db = $MIOLO->GetDatabase('admin');
      $sql = new sql('*','miolo_transacao');
      $n = $db->Count($sql); // total de registros
      $totalpages = (int) $n / 5; // cada página com 5 registros
      for($page=1; $page <= $totalpages; $page++)
      {
        $sql->SetRange($page,5);
        $query = $db->GetQuery($sql);
        $result = $query->result;
        $n = $query->GetRowCount();
        echo "Page: $page  [$n records]: <br>";
```

```
        echo '          #rec - transação' . '<br>';
        for ($i=0; $i < $n; $i++)
        {
           echo "         #$i - " . $result[$i][1] . '<br>';
        }
        echo "<br>";
      }
```

Paginação com objeto Query

Código:

```
      $db = $MIOLO->GetDatabase('admin');
      $sql = new sql('*','miolo_transacao');
      $query = $db->GetQuery($sql);
      $query->SetPageLength(10);  // indica o tamanho de cada página
      $n = $query->GetPageCount(); // obtém a quantidade páginas
      echo "<b>$n pages</b><br><br>";
      for($page=1; $page <= $n; $page++)
      {
          $result = $query->GetPage($page); // subconjunto do ResultSet, correspondente a pagina
          $m = count($result); // a pagina pode não estar completa
          echo "Page: $page  [$m records]: <br>";
          echo '          #rec - transação' . '<br>';
          for ($i=0; $i < $m; $i++)
          {
             echo "         #$i - " . $result[$i][1] . '<br>';
          }
          echo "<br>";
      }
```

Geração automática de ID (sequences)

Código:

```
      $db = $MIOLO->GetDatabase('admin');
      $id = $db->GetNewId('seq_miolo_transacao');
```

Transações

Código:

```
      $db = $MIOLO->GetDatabase('admin');
//
// Exemplo com Commit
//
      $id = $db->GetNewId('seq_miolo_transacao');  // inserir nova transacao ‘Teste’
      $sql = new sql('idtrans, transacao, idsistema','miolo_transacao');
      $args = array($id,'Teste',1);
      $cmd = array(); // array com os comandos SQL
      $cmd[] = $sql->Insert($args);
      $sql->sql('idtrans, idgrupo, direito','miolo_acesso'); // reconstrói o objeto Sql
      $cmd[] = $sql->Insert(array($id, 1, 1));
      $cmd[] = $sql->Insert(array($id, 2, 1));
      $cmd[] = $sql->Insert(array($id, 3, 1));
      $cmd[] = $sql->Insert(array($id, 4, 1));
      $cmd[] = $sql->Insert(array($id, 5, 1));
      $ok = $db->Execute($cmd); // como $cmd é array, vai gerar uma transação
      echo "id = $id <br>";
      echo ($ok ? 'Transaction Ok' : 'Transaction Fail');
      echo '<br>';
//
// Exemplo com RollBack
//
      $id = $db->GetNewId('seq_miolo_transacao');
      $sql = new sql('idtrans, transacao, idsistema','miolo_transacao');
      $args = array($id,'Teste2',1);
      $cmd = array();
      $cmd[] = $sql->Insert($args);
      $sql->sql('idtrans, idgrupo, direito','miolo_acesso');
      $cmd[] = $sql->Insert(array($id, 1, 1));
      $cmd[] = $sql->Insert(array($id, 2, 1));
      $cmd[] = $sql->Insert(array('aaa', 3, 1)); // este comando gerará um erro!
      $cmd[] = $sql->Insert(array($id, 4, 1));
      $cmd[] = $sql->Insert(array($id, 5, 1));
      $ok = $db->Execute($cmd);
      echo "id = $id <br>";
      echo ($ok ? 'Transaction Ok' : 'Transaction Fail ' . $db->GetError());
```

Ordenando o resultset

É possível ordenar o resultado de uma consulta (resultset). Não se trata de indicar uma cláusula ORDER BY para

um comando SQL SELECT, mas sim ordenar o array resultante de uma consulta já realizada (usando funções do PHP).

Código:

```
$db = $MIOLO->GetDatabase('admin');
$sql = new sql('*','miolo_transacao',"transacao like 'A%'");
$query = $db->GetQuery($sql);
$n = $query->GetRowCount();
echo 'Results - before order' . '<br>';
for ($i=0; $i < $n; $i++)
{
   echo "         #$i - " . $result[$i][0] . ' – ' . $result[$i][1] . '<br>';
}
echo "<br>";
$query->SetOrder('transacao');
echo 'Results - after order' . '<br>';
for ($i=0; $i < $n; $i++)
{
   echo "         #$i - " . $result[$i][0] . ' – ' . $result[$i][1] . '<br>';
}
echo "<br>";
```

Filtrando o resultset

É possível filtrar o resultado de uma consulta (resultset). Não se trata de indicar uma cláusula WHERE para um comando SQL SELECT, mas sim filtrar o array resultante de uma consulta já realizada. Como o filtro modifica o resultset, não é possível voltar ao resultset original depois de aplicado o filtro. Podem ser usados os operadores '=', '!=','like' e 'regex'.

Código com operador 'like':

```
$db = $MIOLO->GetDatabase('admin');
$sql = new sql('*','miolo_sistema');
$query = $db->GetQuery($sql);
$query->AddFilter('sistema','like','C%');
$query->ApplyFilter();
```

Código com operador 'regex':

```
$db = $MIOLO->GetDatabase('admin');
$sql = new sql('*','miolo_sistema');
$query = $db->GetQuery($sql);
$query->AddFilter('sistema','regex','^(.*?)A(.*)');
$query->ApplyFilter();
```

7.Camada de persistência de objetos

A camada de persistência do Miolo está baseada nos trabalhos de Ambler e Artyom Rudoy (v. Referências). A implementação atual tem as seguintes características:

- O mecanismo de persistência está encapsulado. A classe MBusiness passa a herdar da classe de persistência (PersistentObject), 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.
- Ações sobre múltiplos objetos. São fornecidos mecanismos para recuperação e remoção de múltiplos objetos. Os mecanismos de recuperação permitem retornar objetos MQuery (com acesso ao ResultSet via camada DAO do Miolo) ou Cursores (um cursor está implementado como um array de objetos).
- Suporte a "lazy read" através do uso de proxies. Um objeto proxy permite recuperar apenas alguns atributos do objeto, evitando o overhead de recuperar todos os atributos. Isto é útil em situações tais como a exibição de listas de seleção, por exemplo.
- Suporte a associações. Quando um objeto é recuperado, removido ou atualizado, a mesma ação pode ser realizada nos objetos associados, se desejado. As associações do tipo ManyToMany podem ser tratadas automaticamente pela camada de persistência.
- Suporte a herança, tornando possível mapear uma árvore de herança para um esquema no banco de dados.
- Suporte a transações, geração automática de identificadores (OID), geração automática do comando SQL, acesso paginado e acesso a diferentes bancos de dados, que são características implementadas pela camada DAO do Miolo.
- Suporte a conversão de valores de atributos, através de classes de conversão.
- Suporte a recuperação simultânea de objetos distintos através de SQL Joins.

Neste tutorial o seguinte modelo de classes será usado (fig. 1), com o esquema do banco de dados correspondente (fig.2). Estão indicados os atributos auxiliares relativos à navegação das associações (geralmente não exibidos nos diagramas UML) para melhor compreensão do processo de mapeamento.

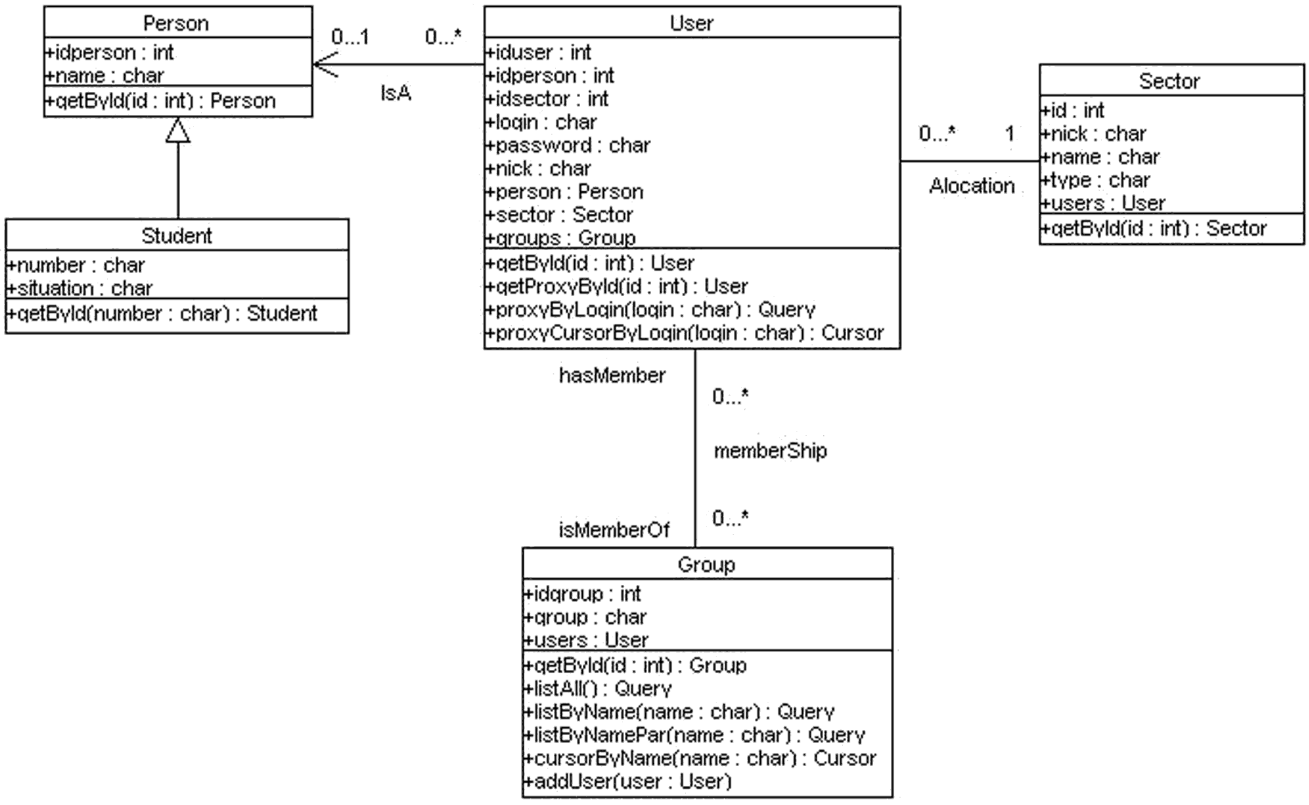


Figura 1 – Modelo de Classes

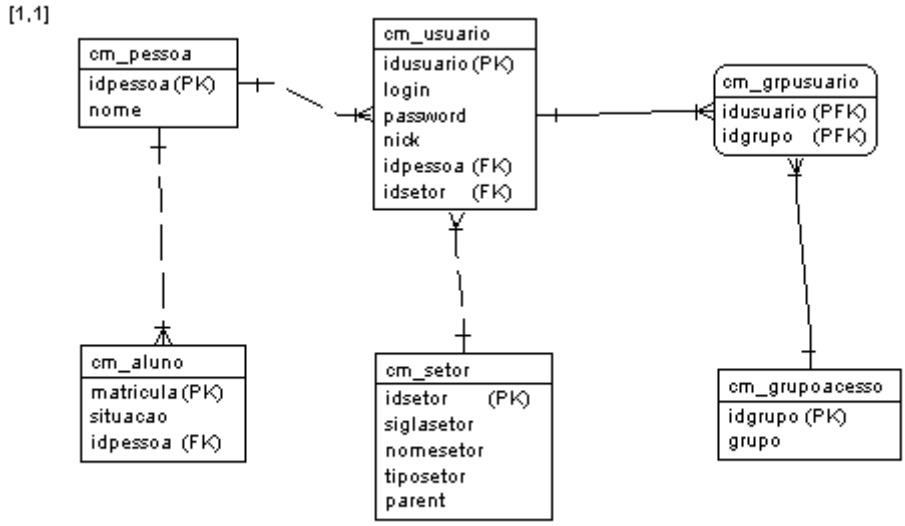


Figura 2 – Esquema do banco de dados relacional

Mapeamento

No contexto da camada de persistência, o mapeamento diz respeito a como são representados os objetos, seus atributos/propriedades e suas associações, em termos do modelo relacional.
Nesta implementação estamos usando arquivos XML para representar o mapeamento das classes persistentes e das classes associativas (classes que são usadas dentro da camada de persistência para representar as associações muitos-para-muitos - ManyToMany). São criados mapas para as classes, para os atributos, para as associações, para as colunas e para as tabelas, permitindo uma grande flexibilidade no processo de mapeamento.
Os mapas XML são criados com o nome <nome_da_classe>.xml e ficam localizados no subdiretório /map dentro do diretório <modulo>/classes.
Os exemplos seguintes mostram os mapas XML criados para cada uma das classes.

Classe USER

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
   <moduleName>persistence</moduleName>
   <className>user</className>
   <tableName>cm_usuario</tableName>
   <databaseName>admin</databaseName>
   <attribute>
      <attributeName>iduser</attributeName>
      <columnName>idusuario</columnName>
      <key>primary</key>
      <idgenerator>seq_cm_usuario</idgenerator>
   </attribute>
   <attribute>
      <attributeName>login</attributeName>
      <columnName>login</columnName>
      <proxy>true</proxy>
   </attribute>
   <attribute>
      <attributeName>password</attributeName>
      <columnName>password</columnName>
   </attribute>
   <attribute>
      <attributeName>nick</attributeName>
      <columnName>nick</columnName>
   </attribute>
   <attribute>
      <attributeName>idperson</attributeName>
      <columnName>idpessoa</columnName>
      <proxy>true</proxy>
```

```
   </attribute>
   <attribute>
       <attributeName>idsector</attributeName>
       <columnName>idsetor</columnName>
       <proxy>true</proxy>
   </attribute>
   <attribute>
       <attributeName>person</attributeName>
   </attribute>
   <attribute>
       <attributeName>groups</attributeName>
   </attribute>
   <attribute>
       <attributeName>sector</attributeName>
   </attribute>

   <association>
       <toClassModule>persistence</toClassModule>
       <toClassName>person</toClassName>
       <cardinality>oneToOne</cardinality>
       <target>person</target>
       <retrieveAutomatic>true</retrieveAutomatic>
       <saveAutomatic>true</saveAutomatic>
       <entry>
              <fromAttribute>idperson</fromAttribute>
              <toAttribute>idperson</toAttribute>
      </entry>
   </association>

   <association>
       <toClassModule>persistence</toClassModule>
       <toClassName>group</toClassName>
       <associativeClassModule>persistence</associativeClassModule>
       <associativeClassName>groupuser</associativeClassName>
       <cardinality>manyToMany</cardinality>
       <target>groups</target>
       <retrieveAutomatic>false</retrieveAutomatic>
       <saveAutomatic>false</saveAutomatic>
       <direction>
              <fromAttribute>users</fromAttribute>
              <toAttribute>groups</toAttribute>
       </direction>
   </association>

   <association>
       <toClassModule>persistence</toClassModule>
       <toClassName>sector</toClassName>
       <cardinality>oneToOne</cardinality>
       <target>sector</target>
       <retrieveAutomatic>true</retrieveAutomatic>
       <saveAutomatic>false</saveAutomatic>
       <entry>
              <fromAttribute>idsector</fromAttribute>
              <toAttribute>id</toAttribute>
       </entry>
   </association>
</map>
```

Classe GROUP

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
   <moduleName>persistence</moduleName>
   <className>group</className>
   <tableName>cm_grupoacesso</tableName>
   <databaseName>admin</databaseName>
   <attribute>
```

```
      <attributeName>idgroup</attributeName>
      <columnName>idgrupo</columnName>
      <key>primary</key>
   </attribute>
   <attribute>
      <attributeName>group</attributeName>
      <columnName>grupo</columnName>
   </attribute>
   <attribute>
      <attributeName>users</attributeName>
   </attribute>

   <association>
      <toClassModule>persistence</toClassModule>
      <toClassName>user</toClassName>
      <associativeClassModule>persistence</associativeClassModule>
      <associativeClassName>groupuser</associativeClassName>
      <cardinality>manyToMany</cardinality>
      <target>users</target>
      <retrieveAutomatic>false</retrieveAutomatic>
      <saveAutomatic>true</saveAutomatic>
      <deleteAutomatic>true</deleteAutomatic>
      <direction>
            <fromAttribute>groups</fromAttribute>
            <toAttribute>users</toAttribute>
      </direction>
   </association>
</map>
```

Classe SECTOR

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
   <moduleName>persistence</moduleName>
   <className>sector</className>
   <tableName>cm_setor</tableName>
   <databaseName>admin</databaseName>
   <attribute>
      <attributeName>id</attributeName>
      <columnName>idsetor</columnName>
      <key>primary</key>
   </attribute>
   <attribute>
      <attributeName>nick</attributeName>
      <columnName>siglasetor</columnName>
   </attribute>
   <attribute>
      <attributeName>name</attributeName>
      <columnName>nomesetor</columnName>
   </attribute>
   <attribute>
      <attributeName>type</attributeName>
      <columnName>tiposetor</columnName>
   </attribute>
   <attribute>
      <attributeName>users</attributeName>
   </attribute>

   <association>
      <toClassModule>persistence</toClassModule>
      <toClassName>user</toClassName>
      <cardinality>oneToMany</cardinality>
      <target>users</target>
      <retrieveAutomatic>true</retrieveAutomatic>
      <saveAutomatic>false</saveAutomatic>
      <deleteAutomatic>true</deleteAutomatic>
      <inverse>true</inverse>
      <entry>
```

```
            <fromAttribute>idsector</fromAttribute>
            <toAttribute>id</toAttribute>
        </entry>
    </association>
</map>
```

Classe PERSON

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
    <moduleName>persistence</moduleName>
    <className>person</className>
    <tableName>cm_pessoa</tableName>
    <databaseName>admin</databaseName>
    <attribute>
        <attributeName>idperson</attributeName>
        <columnName>idpessoa</columnName>
        <key>primary</key>
    </attribute>
    <attribute>
        <attributeName>name</attributeName>
        <columnName>nome</columnName>
        <converter>
            <converterName>CaseConverter</converterName>
                <parameter>
                    <parameterName>case</parameterName>
                    <parameterValue>upper</parameterValue>
                </parameter>
            </converter>
    </attribute>
</map>
```

Classe STUDENT

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
    <moduleName>persistence</moduleName>
    <className>student</className>
    <tableName>ga_aluno</tableName>
    <databaseName>admin</databaseName>
    <extends>
        <moduleName>tutorial3</moduleName>
        <className>person</className>
    </extends>
    <attribute>
        <attributeName>number</attributeName>
        <columnName>matricula</columnName>
        <key>primary</key>
    </attribute>
    <attribute>
        <attributeName>situation</attributeName>
        <columnName>idsituacao</columnName>
    </attribute>
    <attribute>
        <attributeName>idperson</attributeName>
        <columnName>idpessoa</columnName>
        <reference>idperson</reference>
    </attribute>
</map>
```

Classe associativa GROUPUSER

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
    <moduleName>persistence</moduleName>
```

```
   <className>groupuser</className>
   <tableName>cm_grpusuario</tableName>
   <databaseName>admin</databaseName>
   <attribute>
       <attributeName>iduser</attributeName>
       <columnName>idusuario</columnName>
   </attribute>
   <attribute>
       <attributeName>idgroup</attributeName>
       <columnName>idgrupo</columnName>
   </attribute>

   <association>
       <toClassModule>persistence</toClassModule>
       <toClassName>user</toClassName>
       <cardinality>oneToOne</cardinality>
       <target>users</target>
       <retrieveAutomatic>true</retrieveAutomatic>
       <saveAutomatic>true</saveAutomatic>
       <entry>
              <fromAttribute>iduser</fromAttribute>
              <toAttribute>iduser</toAttribute>
       </entry>
   </association>

   <association>
       <toClassModule>persistence</toClassModule>
       <toClassName>group</toClassName>
       <cardinality>oneToOne</cardinality>
       <target>groups</target>
       <retrieveAutomatic>true</retrieveAutomatic>
       <saveAutomatic>true</saveAutomatic>
       <entry>
              <fromAttribute>idgroup</fromAttribute>
              <toAttribute>idgroup</toAttribute>
       </entry>
   </association>
</map>
```

O elemento principal do mapa XML é denominado <map>.

<moduleName>	nome do módulo onde a classe de negócios está definida
<className>	nome da classe de negócios, dentro do módulo <moduleName>
<tableName>	nome da tabela que mapeia a classe. Nesta versão uma classe pode ser mapeada para apenas uma tabela.
<databaseName>	nome da configuração do banco de dados no arquivo miolo.conf
<extends>	Usado no suporte à herança entre classes. <moduleName> e <className> indicam qual é a superclasse.

Atributos

Os atributos são definidos dentro da tag <attribute>.

<attributeName>	nome do atributo conforme definido na classe
<columnName>	nome da coluna que representa o atributo na tabela <tableName>. Atributos que são usados apenas para representar a associação entre duas classes geralmente não têm colunas associadas (p.ex. o atributo person na classe User).Opcional.
<key>	indica, quando este atributo é uma chave da tabela, qual o tipo de chave. Pode assumir dois valores: primary e foreign. Opcional.
<idgenerator>	quando o atributo é chave, esta tag pode indicar qual o nome do IdGenerator a ser passado para a camada DAO do Miolo, para a geração automática de OID. Opcional.
<proxy>	indica que este atributo será recuperado com objetos proxy. O valor default é true. Atributos chave são sempre recuperados com objetos proxy.

<reference>	Usado no suporte à herança entre classes. Indica o nome do atributo da superclasse que é referenciado por este atributo. É usado para a construção do join entre as tabelas da superclasse e da classe sendo mapeada.
<attributeIndex>	na criação de atributos indexados (arrays), pode-se informar qual será o índice associado à coluna descrita em <columnName>. Isto permite agrupar vários campos da tabela em um único atributo do objeto.

Associações

As associações representam os relacionamentos existentes entre as classes. São definidas a partir do ponto de vista da classe sendo mapeada. Suas características estão dentro da tag <association>.

<toClassModule>	nome do módulo da classe associada
<toClassName>	nome da classe associada, dentro do módulo <toClassModule>
<cardinality>	cardinalidade da associação. Refere-se a quantos objetos da classe associada estão relacionados com a classe sendo mapeada. Os valores possíveis são: oneToOne, oneToMany e manyToMany.
<target>	indica qual o atributo vai receber o objeto (ou o array de objetos) que for recuperado através da associação. É também o nome usado para identificar a associação. Por exemplo, a associação com <target> igual a 'person', na classe User, indica que um objeto da classe Person, associado com este objeto User, vai ser recuperado e armazenado no atributo 'person'.
<retrieveAutomatic> <saveAutomatic> <deleteAutomatic>	indica se as operações de recuperação, armazenamento e remoção (respectivamente) serão realizadas automaticamente na classe associada. Estas tags são opcionais e o valor default é false. Cuidado especial deve ser tomado nesta definição pois a recuperação (ou armazenamento ou remoção) de um simples objeto pode ocasionar a recuperação automática de dezenas de outros, dependendo do valor destas tags. De certa forma é usada para mapear (e sobrepor) as definições de integridade referencial declaradas no esquema relacional.
<joinAutomatic>	Indica que o(s) objeto(s) associado(s) será(ao) recuperado(s) simultaneamente, através de um SQL Join. Os valores possíveis são 'inner','left' e 'right', indicando o tipo de join.
<orderAttribute>	Indica o(s) atributo(s) que serão usados para ordenar os objetos retornados da classe associada. <orderAttributeName> indica o nome do atributo e <orderAttributeDirecton> indica a direção ('ascend' ou 'descend').
<indexAttribute>	Indica, nas associações oneToMany e manyToMany, o atributo cujos valores serão usados como índices do array de objetos retornado pela recuperação na classe associada.
<entry>	Nas associações oneToOne e oneToMany descreve a associação entre os atributos da classe mapeada e os atributos da classe associada. Dentro da tag <entry>, <fromAttribute> indica o atributo da classe mapeada e <toAttribute> indica o atributo da classe associada. Podem existem várias tags <entry> (por exemplo, no caso de chaves compostas).
<inverse>	Indica se o sentido da associação é o mesmo sentido da referência entre as tabelas (FK -> PK) . Se <inverse> é false (default), o atributo <fromAttribute> é acessado na classe mapeada e o atributo <toAttribute> é acessado na classe <toClassModule>:<toClassName>. O sentido da associação é o mesmo sentido de FK->PK. Se <inverse> é true, o atributo <fromAttribute> é acessado na classe <toClassModule>:<toClassName> e o atributo <toAttribute> é acessado na classe mapeada. O sentido da associação é inverso ao sentido de FK->PK. Como exemplo do uso de <inverse> tomemos a associação entre as classes USER e SECTOR: a) <inverse> = false Classe mapeada: USER Classe associada: SECTOR

	<toClassModule>:<toClassName> : persistence:sector <fromAttribute>: idsector (atributo na classe USER) <toAttribute>: id (atributo na classe SECTOR) O sentido da associação (user->sector) coincide com o sentido foreignKey -> primaryKey b) <inverse> = true Classe mapeada: SECTOR Classe associada: USER <toClassModule>:<toClassName> : persistence:user <fromAttribute>: idsector (atributo na classe USER) <toAttribute>: id (atributo na classe SECTOR) O sentido da associação (sector->user) é inverso ao sentido foreignKey -> primaryKey
<associativeClassModule>	nome do módulo da classe associativa, usada no mapeamento de associações ManyToMany
<associativeClassName>	nome da classe associativa, dentro do módulo <toClassModule>, usada no mapeamento de associações ManyToMany. Esta classe **não existe** no modelo de classes de negócio. É criada internamente na camada de persistência para permitir o acesso à tabela de ligação no modelo relacional (uma vez que no modelo relacional os relacionamentos são sempre OneToOne ou OneToMany). Se no modelo de classes de negócio existir realmente uma classe associativa (uma classe relacionada a uma associação, geralmente com atributos próprios) ela deve ser mapeada normalmente como uma classe persistente.
<direction>	descreve a direção de uma associação com cardinalidade ManyToMany. Dentro da tag <direction>, <fromAttribute> e <toAttribute> indicam os nomes da associações definidas no mapa da classe associativa que simulam o relacionamento com a tabela de ligação no modelo relacional. A ordem em que os atributos são colocados define a direção da associação. Por exemplo, no mapa da classe USER, a associação com <target> igual a 'groups' (que indica a associação entre USER e GROUP) tem cardinalidade ManyToMany (um usuário pode estar em vários grupos, um grupo pode ter vários usuários). É definida a classe associativa GROUPUSER (interna à camada de persistência), em cujo mapa encontramos duas associações (groups e users). A tag <direction> em USER indica que para materializar a associação com GROUP, deve ser percorrida a associação USER<-GROUPUSER e depois GROUPUSER->GROUP.

Classes de conversão de valores

A camada de persistência permite definir métodos de conversão de valores, que são podem ser usados antes e/ou depois de um determinado valor ser gravado/acessado no banco de dados.
Estes métodos de conversão devem ser definidos em classes próprias, que implementem a interface IConverter (definida em <miolo>/classes/persistence/converter). A camada já fornece duas classes pré-definidas: TrivialConverter e CaseConverter. TrivialConverter é a classe default, usada quando não for definida nenhuma outra classe, e na realidade não realiza nenhuma conversão. A classe CaseConverter é fornecida como exemplo de implementação, e pode ser usada para converter os dados string para maiúsculas ou minúsculas.
Um exemplo da definição de conversão de atributos é dado no mapeamento da classe Person:

Classe PERSON

```
…
   <attribute>
```

```
      <attributeName>name</attributeName>
        <columnName>nome</columnName>
        <converter>
            <converterName>CaseConverter</converterName>
            <parameter>
                  <parameterName>case</parameterName>
                  <parameterValue>upper</parameterValue>
           </parameter>
        </converter>
   </attribute>
…
```

Os parâmetros para a conversão são definidos dentro da tag <converter>.

<converterName>	nome da classe de conversão. Deve implementar a interface IConverter.
<parameter>	Define os parâmetros que poderão ser usados como valores de propriedades na classe de conversão. Uma classe pode ter vários parâmetros associados. No caso do exemplo, o parâmetro 'case' define se a conversão será para maiúsculas ('upper') ou minúsculas ('lower').Opcional.
<parameterName> <parameterValue>	Define o nome e o valor do parâmetro a ser passado para a classe de conversão.

A listagem abaixo mostra a definição da classe CaseConverter.

```
class caseconverter implements IConverter
{
   private $case;

   function caseconverter()
   {
   }

   function init($properties)
   {
      $this->case = $properties['case'];
   }

   function convertFrom($object)
   {
       switch ($this->case)
       {
           case 'upper': $o = strtoupper((string)$object); break;
           case 'lower': $o = strtolower((string)$object); break;
       }
       return $o;
   }

   function convertTo($object)
   {
       return strtoupper((string)$object);
   }

   function convertColumn($object)
   {
       return $object;
   }

   function convertWhere($object)
   {
       return $object;
   }
}
```

Os parâmetros passados no mapeamento são armazenados no array $properties, que é passado para o método init(). O método convertFrom() converte o valor que foi lido do banco para ser armazenado em um atributo. O método convertTo() converte o valor do atributo em um valor a ser armazenado no banco. O método convertColumn é usado primariamente pelo Miolo para fazer conversões necessárias na construção do comando

SQL SELECT (por exemplo, na formatação de campos do tipo Date). O método convertWhere também é usado pelo Miolo para fazer conversões necessárias na construção da cláusula Where no comando SQL SELECT (também, por exemplo, na formatação de campos do tipo Date).

Objetos persistentes

Uma vez que a classe MBusiness passa a herdar da classe PersistentObject, todos os objetos de negócio são automaticamente persistentes, ou seja, podem usar os métodos de persistência. O acesso direto à camada DAO do Miolo continua válido, permitindo construir comandos SQL independente da camada de persistência.

Cursor

Um cursor é um objeto que mantém o ResultSet gerado por uma query no banco de dados, e conforme vai sendo percorrido, gera os objetos correspondentes. O método disponível para percorrer o cursor é getObject(), que obtém o objeto corrente e avança para o próximo, retornando NULL se o final do ResultSet foi alcançado.

Exemplo

```
        $cursor = $this->user->ProxyCursorByLogin('2146'); // retorna um cursor, depois de uma
                                                           // consulta ao banco
        while ($obj = $cursor->getObject())
        {
           $text .= "<br>Iduser: " . $obj->iduser;
           $text .= "<br>Login: " . $obj->login . "<br>";
        }
        echo $text;
```

Queries

As consultas ao banco de dados (queries) são realizadas através de um mecanismo chamado "query by criteria". Através deste mecanismo, definimos um "criteria" (um critério para a consulta), instanciando um objeto do tipo RetrieveCriteria. Um objeto RetrieveCriteria está associado a uma classe persistente que serve de base para a consulta (as consultas sempre são feitas sob a perspectiva de uma determinada classe persistente), sendo obtido através do método getCriteria().

Exemplos

Para efeito dos exemplos, $this->group, $this->user, $this->sector, $this->person, $this->student são objetos das respectivas classes.

Consulta simples

Código:

```
        $criteria = $this->group->getCriteria();
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso
```

Consulta com filtro

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->addCriteria('group','LIKE',"'SIGA%'");
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso WHERE
(cm_grupoacesso.grupo LIKE 'SIGA%')
```

Consulta com filtro composto

A cláusula WHERE é criada a partir de um objeto da classe CriteriaCondition, que normalmente é encapsulado pela camada de persistência. No caso de criação de filtros compostos, pode ser necessário instanciar um objeto CriteriaCondition separado e anexa-lo ao critério da consulta.

Código:

```
        $criteria = $this->group->getCriteria();
        $cc = new CriteriaCondition;
```

```
        $cc->addCriteria($criteria->getCriteria('group','LIKE',"'%A%'"));
        $cc->addOrCriteria($criteria->getCriteria('group','LIKE',"'%E%'"));
        $criteria->addCriteria('group','LIKE',"'C%'");
        $criteria->addCriteria($cc);
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso WHERE
((cm_grupoacesso.grupo LIKE 'C%') AND (((cm_grupoacesso.grupo LIKE '%A%') OR
(cm_grupoacesso.grupo LIKE '%E%')) ))
```

Consulta com parâmetro

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->addCriteria('group','LIKE',"?");
        $criteria->addOrCriteria('group','LIKE',"'C%'");
        $query = $criteria->retrieveAsQuery("A%");
```

SQL:

```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso WHERE
((cm_grupoacesso.grupo LIKE 'A%') OR (cm_grupoacesso.grupo LIKE 'C%'))
```

Consulta com ordenação

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->addOrderAttribute('group');
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso ORDER BY
cm_grupoacesso.grupo ASC
```

Consulta com join, via associação oneToOne, sem definição de colunas

Código:

```
        $criteria = $this->user->getCriteria();
        $criteria->addCriteria('sector.nick','LIKE',"'PROR%'");
        $query = $criteria->retrieveAsProxyQuery();
```

SQL:

```
SELECT cm_usuario.idusuario,cm_usuario.login,cm_usuario.idpessoa,cm_usuario.idsetor FROM
cm_usuario,cm_setor WHERE (cm_setor.siglasetor LIKE 'PROR%') and
(cm_usuario.idsetor=cm_setor.idsetor)
```

Consulta com join, via associação oneToOne, com definição de colunas

Código:

```
        $criteria = $this->user->getCriteria();
        $criteria->addCriteria('sector.nick','LIKE',"'PROR%'");
        $criteria->addColumnAttribute('login');
        $criteria->addColumnAttribute('sector.nick');
        $query = $criteria->retrieveAsProxyQuery();
```

SQL:

```
SELECT cm_usuario.login,cm_setor.siglasetor FROM cm_usuario,cm_setor WHERE
(cm_setor.siglasetor LIKE 'PROR%') and (cm_usuario.idsetor=cm_setor.idsetor)
```

Consulta com join, via associação manyToMany, sem definição de colunas

Código:

```
        $criteria = $this->user->getCriteria();
        $criteria->addCriteria('groups.group','LIKE',"'CDARA%'");
        $query = $criteria->retrieveAsProxyQuery();
```

SQL:

```
SELECT cm_usuario.idusuario,cm_usuario.login,cm_usuario.idpessoa,cm_usuario.idsetor FROM
cm_grpusuario,cm_usuario,cm_grupoacesso WHERE (cm_grupoacesso.grupo LIKE 'CDARA%') and
(cm_grpusuario.idusuario=cm_usuario.idusuario) and
(cm_grpusuario.idgrupo=cm_grupoacesso.idgrupo)
```

Consulta com join, via associação manyToMany, com definição de colunas e agrupamento

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->addColumnAttribute('group');
        $criteria->addColumnAttribute('count(users.iduser)');
```

```
        $criteria->addGroupAttribute('group');
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.grupo,count(cm_usuario.idusuario) FROM
cm_grpusuario,cm_grupoacesso,cm_usuario WHERE (cm_grpusuario.idgrupo=cm_grupoacesso.idgrupo)
and (cm_grpusuario.idusuario=cm_usuario.idusuario) GROUP BY cm_grupoacesso.grupo
```

Consulta com join, via associação manyToMany, com definição de colunas, agrupamento e cláusula having

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->addColumnAttribute('group');
        $criteria->addColumnAttribute('count(users.iduser)');
        $criteria->addGroupAttribute('group');
        $criteria->addHavingCriteria('count(users.iduser)','>','50');
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.grupo,count(cm_usuario.idusuario) FROM
cm_grpusuario,cm_grupoacesso,cm_usuario WHERE (cm_grpusuario.idgrupo=cm_grupoacesso.idgrupo)
and (cm_grpusuario.idusuario=cm_usuario.idusuario) GROUP BY cm_grupoacesso.grupo HAVING
(count(cm_usuario.idusuario) > 50)
```

Consulta com uso do operador de conjuntos IN

Código:

```
        $criteria = $this->group->getCriteria();
        $values = array('CDARA','CURRICULO','EXCECAO');
        $criteria->addCriteria('group','IN', $values);
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso WHERE
(cm_grupoacesso.grupo IN ('CDARA', 'CURRICULO', 'EXCECAO'))
```

Consulta com uso de alias

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->setAlias('G');
        $criteria->addColumnAttribute('G.idgroup');
        $criteria->addColumnAttribute('G.group');
        $criteria->addCriteria('group','LIKE', "'A%'");
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT G.idgrupo,G.grupo FROM cm_grupoacesso G WHERE (G.grupo LIKE 'A%')
```

Consulta com uso de alias para associação

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->setAlias('G');
        $criteria->setAssociationAlias('users','U');
        $criteria->addColumnAttribute('G.idgroup');
        $criteria->addColumnAttribute('G.group');
        $criteria->addColumnAttribute('U.iduser');
        $criteria->addCriteria('group','=', "'ADMIN'");
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT G.idgrupo,G.grupo,U.idusuario FROM cm_grpusuario,cm_grupoacesso G,cm_usuario U WHERE
(G.grupo = 'ADMIN') and (cm_grpusuario.idgrupo=G.idgrupo) and
(cm_grpusuario.idusuario=U.idusuario)
```

Consulta com auto-relacionamento

Código:

```
        $criteria = $this->sector->getCriteria();
        $criteria->setAutoAssociationAlias('S1','S2');
        $criteria->addColumnAttribute('S1.nick');
        $criteria->addColumnAttribute('S2.nick');
```

```
        $criteria->addCriteria('S1.nick', '=', 'S2.parent');
        $criteria->addCriteria('S1.parent','=', "'PROGRAD'");
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT S1.siglasetor,S2.siglasetor FROM cm_setor S1,cm_setor S2 WHERE ((S1.siglasetor =
S2.paisetor) AND (S1.paisetor = 'PROGRAD'))
```

Consulta com subquery

Código:

```
        $subCriteria = $this->user->getCriteria();
        $subCriteria->addCriteria('sector.nick','LIKE',"'PROR%'");
        $subCriteria->addColumnAttribute('iduser');
        $criteria = $this->user->getCriteria();
        $criteria->addCriteria('iduser','IN', $subCriteria);
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT
cm_usuario.idusuario,cm_usuario.login,cm_usuario.password,cm_usuario.nick,cm_usuario.idpessoa,
cm_usuario.idsetor FROM cm_usuario WHERE (cm_usuario.idusuario IN (SELECT cm_usuario.idusuario
FROM cm_usuario,cm_setor WHERE (cm_setor.siglasetor LIKE 'PROR%') and
(cm_usuario.idsetor=cm_setor.idsetor)))
```

Consulta com subquery referenciando a query externa

Código:

```
        $subCriteria = $this->user->getCriteria();
        $subCriteria->setReferenceAlias('S');
        $subCriteria->addColumnAttribute('count(iduser)');
        $subCriteria->addCriteria('idsector','=',"S.idsector");
        $criteria = $this->sector->getCriteria();
        $criteria->setAlias('S');
        $criteria->addCriteria($subCriteria, '>', '150');
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
      SELECT S.idsetor,S.siglasetor,S.nomesetor,S.paisetor,S.tiposetor FROM cm_setor S WHERE
((SELECT count(cm_usuario.idusuario) FROM cm_usuario WHERE (cm_usuario.idsetor = S.idsetor)) >
150)
```

Consulta utilizando "path join"

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->addColumnAttribute('group');
        $criteria->addColumnAttribute('users.login');
        $criteria->addColumnAttribute('users.sector.nick');
        $criteria->addCriteria('users.sector.nick','=',"'REITORIA'");
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.grupo,cm_usuario.login,cm_setor.siglasetor FROM
cm_grpusuario,cm_grupoacesso,cm_usuario,cm_setor WHERE (cm_setor.siglasetor = 'REITORIA') and
(cm_grpusuario.idgrupo=cm_grupoacesso.idgrupo) and
(cm_grpusuario.idusuario=cm_usuario.idusuario) and (cm_usuario.idsetor=cm_setor.idsetor)
```

Consulta com outer join

Código:

```
        $criteria = $this->sector->getCriteria();
        $criteria->addColumnAttribute('nick');
        $criteria->addColumnAttribute('count(users.iduser)');
        $criteria->addCriteria('nick','LIKE',"'PRO%'");
        $criteria->addGroupAttribute('nick');
        $criteria->addHavingCriteria('count(users.iduser)','=','0');
        $criteria->setAssociationType('users','right');
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_setor.siglasetor,count(cm_usuario.idusuario) FROM cm_setor,cm_usuario WHERE
(cm_setor.siglasetor LIKE 'PRO%') and (cm_setor.idsetor = cm_usuario.idsetor(+)) GROUP BY
cm_setor.siglasetor HAVING (count(cm_usuario.idusuario) = 0)
```

Consulta com herança entre classes

Código:

```
        $criteria = $this->student->getCriteria();
        $criteria->addColumnAttribute('number');
        $criteria->addColumnAttribute('name');
        $criteria->addCriteria('name','LIKE',"'FELIPE A%'");
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT ga_aluno.matricula,cm_pessoa.nome FROM ga_aluno,cm_pessoa WHERE (cm_pessoa.nome LIKE
'FELIPE A%') and (ga_aluno.idpessoa=cm_pessoa.idpessoa) and
(ga_aluno.idpessoa=cm_pessoa.idpessoa)
```

Referências

- Scott W. Ambler, The fundamental of Mapping Objects to Relational Databases. Disponível em: http://www.agiledata.org/essays/mappingObjects.html
- Scott W. Ambler, The design of a Robust Persistence Layer for Relational Databases. Disponível em: http://www.ambysoft.com/persistenceLayer.pdf
- Artyom Rudoy, Persistence Layer. Disponível em: http://artyomr.narod.ru

8.AJAX

As aplicações Web são tradicionalmente constituídas por um conjunto de páginas relacionadas entre si. A sua utilização (navegação) é feita através de ligações existentes entre as páginas (hyperlinks), ou através da submissão de dados presentes em formulários HTML. Neste último caso, o usuário faz a submissão do formulário e, depois deste ser processado no servidor, ele recebe uma resposta com o novo conteúdo (página). Quando se pretende atualizar apenas parte da interface, este modelo tem problemas como, por exemplo, tempo de espera sem qualquer conteúdo visível ou "flicker" resultantes da atualização total da página e a geração de tráfego desnecessário na rede.

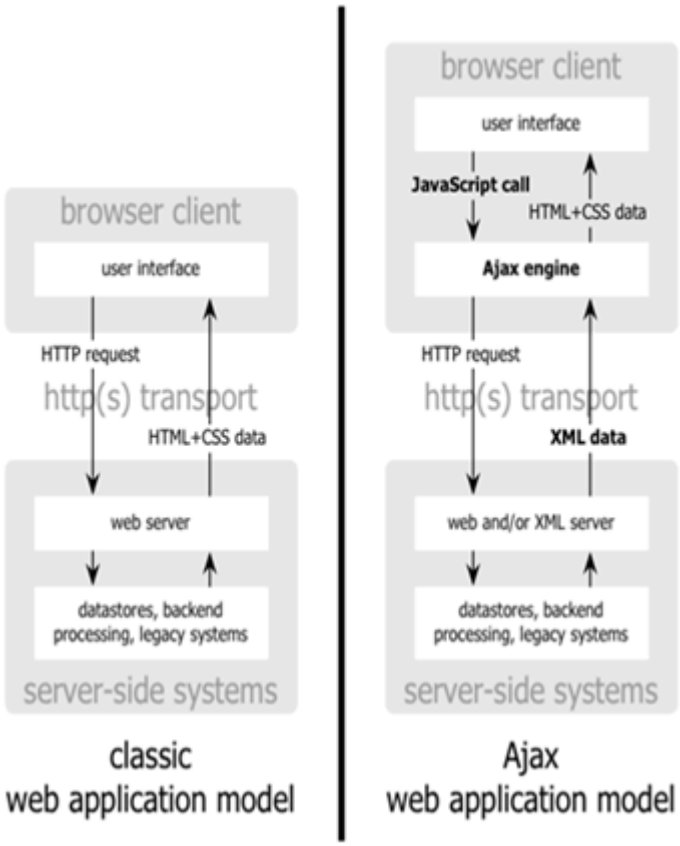
O Asynchronous JavaScript and XML (AJAX) é uma técnica para desenvolvimento de aplicações Web, através da qual as páginas são atualizadas de forma parcial e não total. A sua utilização melhora a interatividade e a experiência de utilização das aplicações Web, reduzindo também a largura de banda utilizada.

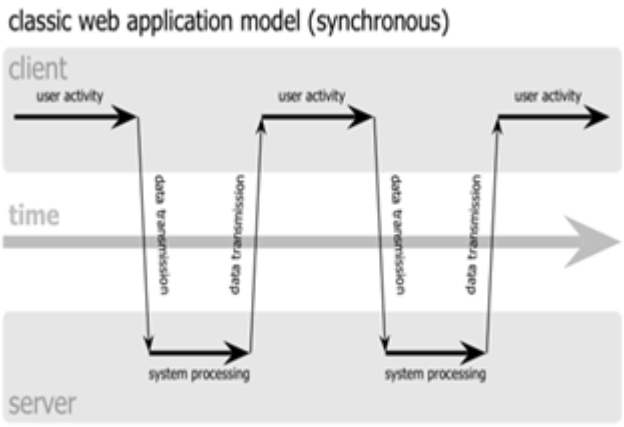
Segundo Jesse James Garret: "O AJAX não é uma tecnologia. São na realidade várias tecnologias, cada uma progredindo de forma independente, que se juntaram de forma a poder explorar formas de melhorar a interação com os usuários em aplicações Web.". De forma simplificada, podemos dizer que AJAX = Asynchronous JavaScript + XML.

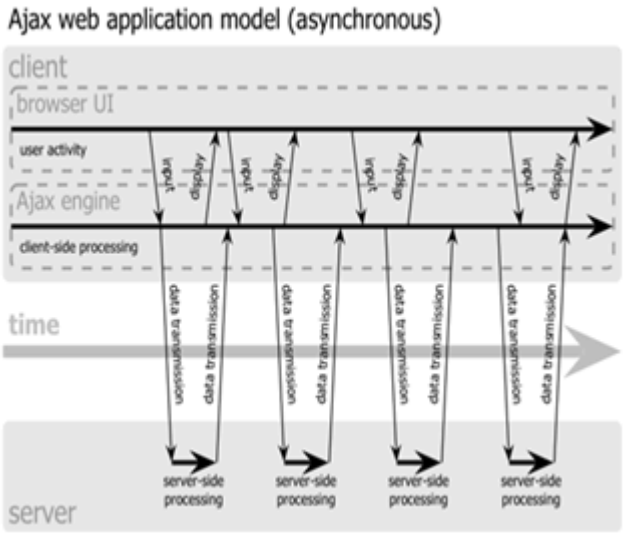
Algumas características do AJAX:

- Apresentação baseada em standards utilizando XHTML e CSS
- Interação e apresentação dinâmica utilizando o Documento Object Model
- Formato standard para troca e manipução de dados - XML
- Comunicação assíncrona com o servidor utilizando XMLHttpRequest
- Uso do Javascript como agregador de todas estas tecnologias

As figuras seguintes mostram uma comparação entre o modelo clássico e o uso do AJAX:







No Miolo o AJAX está implementado através do encapsulamento de duas bibliotecas:

- CPAINT http://cpaint.wiley14.com/
- Prototype http://www.prototypejs.org/

Funcionamento do AJAX

O funcionamento do AJAX depende de definições tanto no client-side, onde são definidos os parâmetros para a chamada a ser feita via javascript e o tratamento da resposta, quanto no server-side, onde são criados os métodos (em PHP) que retornarão o texto/XML/objeto para o client-side. O exemplo seguinte mostra como é feita esta associação:

Server side

Associação de uma chamada AJAX ao evento click de um botão:

```
new MButton('btnSel','[Select]','ajaxSelection.call();')
```

O objeto ajaxSelection é definido pelo desenvolvedor no client-side (código javascript).

Geração da resposta

HTML:
Para gerar o conteúdo HTML a ser colocado em uma área da página (geralmente dentro de um controle MDiv), deve-se ter response_type=TEXT definido no client-side. No server-side, no método que será chamado, o controle a ser gerado deve ser colocado no ThemeElement Ajax, conforme o exemplo abaixo:

```
      // create a instance of a Mselection control
      $sel2 = new MSelection("secondSelection","","Opções",$array);
      // response_type = TEXT : set the ajax area of theme
```

```
      $this->manager->getTheme()->setAjaxContent($sel2);
```

XML/OBJECT/JSON:
Para gerar uma resposta estruturada (um texto XML, que poderá ser tratado no client-side como um objeto XMLDocument ou um objeto Javascript comum, ou um objeto Javascript na notação JSON – através de response_type=XML ou response_type=OBJECT ou response_type=JSON, respectivamente), pode-se usar os próprios métodos do objeto Cpaint. Para a resposta estruturada, não é necessário preencher o ThemeElement Ajax.

```
      // create a instance of a Curso object
       $curso = $this->manager->GetBusiness('tutorial','curso',$idCurso);
      // retrieve alunos with name initial = 'E' inside a query object
      $query = $curso->listAlunos('E');
      // at Javascript was defined response_type = XML or OBJECT or JSON
      // so, we will build a xml object to send, using CPAINT library methods.
      // atribute $this->cp is the cpaint object (created by the parent).
      // creates a main node (aluno)
      $result_node  = $this->cp->add_node('aluno');
      $query->moveFirst();
      while (!$query->eof)
      {
          // foreach record, add a nome node to the main node
          $name_node = $result_node->add_node('nome'); // create a new node
          // put the data inside the node
          $name_node->set_data($query->fields("nome"));
          $query->moveNext();
      }
      // response_type = XML or OBJECT or JSON : it´s not necessary to set the ajax area of
theme
```

O código acima gera um xml com a seguinte estrutura, que será enviado para o cliente:

```
<?xml version="1.0" encoding="UTF-8"?>
<ajaxResponse>
   <aluno>
      <nome>...</nome>
      <nome>...</nome>
      <nome>...</nome>
      ...
   </aluno>
</ajaxResponse>
```

AjaxHandler
Um dos objetivos do uso do Ajax é minimizar o tempo de interação entre o browser e o servidor. No caso específico do Miolo, como os métodos a serem executados pelo Ajax estão dentro de formulários, que por sua vez são executados por handlers, que por sua vez são chamados pelo handler main, há uma interação muito grande com o framework até a resposta ser obtida.
Para minimizar este custo, pode ser definido um handler especial, chamado AjaxHandler, onde os métodos a serem chamados via ajax podem ser colocados. Para se evitar a execução do handler main (que é default no Miolo), deve-se prefixar o nome do handler com um "_".
O código abaixo mostra o AjaxHandler definido no módulo Tutorial (em ajaxhandler.inc):

```
// basic definition to use Ajax

// import the cpaint library
MIOLO::Import('extensions::cpaint2.inc.php','cpaint');

// create the object
$cp = new cpaint();

// insert the scripts
$page->addScript('x/x_core.js');
$page->addScript('x/x_dom.js');
$page->addScript('cpaint/cpaint2.inc.js');
$page->addScript('m_ajax.js');
```

```
// register the methods callable by ajax
$cp->register('ajax_sample');

// check if it is a ajax call
if (($page->request('cpaint_function')) != "")
{
    $theme->clearContent();
    $page->generateMethod = 'generateAJAX';
    $page->cpaint = $cp;
    $cp->start('ISO-8859-1');
}

// define the methods callable by ajax

function ajax_sample($arg)
{
    global $theme;
    $sample = new MRawText("<br>i've received the argument: $arg");
    $theme->setAjaxContent($sample);
}
?>
```

No código do formulário pode-se usar a seguinte estrutura para chamar o handler AjaxHandler:

```
      // an "ajax handler" let us to handler generics ajax calls
      // the "_" prefix escapes the handler "main"
      // and goes straight to handler "ajaxhandler" (without the "_")
      $urlAjaxHandler = $this->manager->getActionURL('tutorial','_ajaxhandler');
.. ..
      // set onclick to call ajaxhandler, after change the target url
      new MButton('btnSelSample','[ajaxhandler]',"ajaxHandlerSample.url = '$urlAjaxHandler';
ajaxHandlerSample.call();"),
.. ..
```

O objeto ajaxHandlerSample é definido pelo desenvolvedor no código javascript.Note que na definição do javascript do botão, o atributo "url" é alterado antes de se fazer a chamada Ajax.

Client side

Definição dos objetos usados na chamada AJAX

O objeto definido via javascript tem a seguinte sintaxe:

```
01: var ajaxObject = new Miolo.Ajax({
02:    updateElement: 'a_html_element',
03:    url: 'a_url',
04:    response_type: {'TEXT' |  'XML' | 'JSON' | 'OBJECT'},
05:    remote_method: 'a_php_method',
06:    parameters: function(){
07:       return {variable | js_object};
08:    }
09:    callback_function: function(result[, xmlText]) {
10:          ..some code ...
11:    }
12: });
```

Linha 01:
Criação do objeto `ajaxObject`, como uma instância da classe `Miolo.Ajax`. O construtor recebe um objeto com os atributos seguintes.

Linha 02:
`updateElement` (opcional): id do elemento HTML cujo conteúdo será atualizado automaticamente com o retorno da chamada AJAX, caso não seja definida uma função em `callback_function`.

Linha 03:
`url` (opcional): url a ser chamada pelo Ajax. Se não for definida, será usado o atributo action do formulário

corrente.

Linha 04:
`response_type` (obrigatório): define o tipo de resposta que deverá ser retornado pelo método do servidor:

- TEXT: um texto plano (sem estrutura)
- XML: um objeto XMLDocument, baseado no xml gerado no servidor
- JSON: um objeto Javascript, baseado no xml gerado pelo servidor
- OBJECT: um objeto Javascript, baseado no xml gerado pelo servidor

Linha 05:
`remote_method` (obrigatório): método php que será executado no lado do servidor. A localização do método é dada pelo atributo `url`.

Linha 06:
`parameters` (opcional): uma função anônima que deve retornar os parâmetros a serem passados ao `remote_method`. Se for um único parâmetro ele pode ser retornado imediatamente:

```
parameters: function(){
      sel = miolo.getElementById("selSample");
      return sel.options[sel.selectedIndex].text;
}
```

Se for um parâmetro estruturado, ele deve ser passado através de um objeto javascript, que será convertido em um objeto php:

```
parameters: function(){
      sel = miolo.getElementById("firstSelection");
      return {
          value: sel.value,
          option: sel.options[sel.selectedIndex].text
      };
}
```

Linha 09:
`callback_function` (opcional): uma função anônima que será executada no retorno da chamada Ajax, caso não seja definido o atributo `updateElement`. O parâmetro "result" é o retorno gerado pelo método php (dependendo do valor de `response_type`) e o parâmetro opcional "xmlText" é o texto xml gerado no caso de retorno de tipos estruturados (XML ou JSON).

9.Dialogs

Os formulários (forms) das aplicações Web são tradicionalmente constituídos por uma página HTML e algum código Javascript. Em contraste com as aplicações desktop, onde váriao forms podem ser apresentados simultaneamente através de diálogos abertos dentro da própria janela da aplicação, as aplicações Web geralmente abrem diversas janelas do browser para exibir as páginas (janelas pop-up).
Esta extensão do Miolo, baseada em objetos javascript, visa permitir que as aplicações desenvolvidas com o framework possam simular algumas das funcionalidades e vantagens de se usar diálogos, mantendo a estrutura da aplicação compatível com o Miolo.
Deve-se ressaltar que, embora não sendo obrigatório, várias destas funcionalidades exigem o uso de javascript junto com o código PHP.

Visão geral

O principal objetivo da extensão é permitir que os formulários criados no Miolo possam ser exibidos em caixas de diálogos. Estes diálogos podem ser modais (não permitir acesso a nenhuma outra parte da aplicação a não ser o diálogo) ou não-modais, podem ser arrastados pela tela (*draggable*) através da barra do título, podem fazer uso da extensão AJAX, o resultado de um submit (POST) é exibido dentro do próprio diálogo e, através de javascript, podem ser acessados campos de outros formulários dinamicamente.
Os diálogos são criados através de IFrames (Internal Frames). Todos os diálogos tem por base a janela principal (não são permitidos diálogos dentro de diálogos), mas um novo diálogo pode ser aberto através de um diálogo já existente (que passa a ser referido como diálogo-pai – *parent*). Os iFrames são criados dinamicamente e recebem um id também dinâmico.

Estrutura das classes

Server side:

MControl
+----MForm
+----+----MFormAjax
+----+----+----MFormDialog
+----MDialog

Client side:

Miolo.Dialog (definida no arquivo m_dialog.js)
Miolo.IFrame (definida no arquivo m_iframe.js)

Objeto miolo.IFrame

```
miolo.iFrame: {
                object: null,
                dialogs: new Array(),
                sufix: 0,
                dragElement: null,
                base: window,
                parent: null,
                getById: function(id) {
                      return this.parent.miolo.iFrame.dialogs[id];
                }
},
```

10.Window

A finalidade do controle MWindow é simular o uso de janelas pop-up dento da própria janela do browser, permitindo a execução de um outro handler do Miolo. A janela possui seu próprio controle para fechar, pode ser arrastada, pode ser modal (evitando a interação com a janela principal do browser) e possui uma barra de status.
No Miolo este controle está implementado através do encapsulamento da biblioteca Prototype Window, de Sebastien Gruhier:

- http://prototype-window.xilinus.com/

Estrutura das classes

Server side:
MControl
+----MWindow

Client side:
Miolo.Window (definida no arquivo m_window.js)
Window (Prototype Window - definida no arquivo window/window.js)

Exemplos

Acessando um formulário em uma window modal

```
// define a URL
$urlWindow = $MIOLO->getActionURL('tutorial','main:windows:window');
// instancia o objeto MWindow
$win = new MWindow('winWindow',array('url'=>$urlWindow));
// define o controle para a barra de status
$win->setStatusBar(new MLabel('Status Bar'));
// obtem o link para abrir a janela
$link = $ui->getWindow('winWindow', true, false));
```

Acessando um campo em um formulário de uma window

Este comando cria um link para alterar o valor do campo txtField na janela winModal2:

```
new MLink('lnk3','Change Value of TextField on Window 2: new value',
"javascript:miolo.getWindow('winModal2').field('txtField','new value')")
```

Botão no formulário para fechar a janela

```
new MButton('btnClose','Fechar', $this->getCloseWindow()),
```

11.Reports

A geração de documentos para impressão, que neste documento estaremos denominando genericamente de *reports,* é parte fundamental de qualquer aplicação.
No Miolo os reports são implementados através da geração de arquivos PDF que, sendo acessados via browser, podem ser impressos localmente. As classes do framework permitem que sejam utilizados diversos mecanismos para a geração de arquivos PDF. Este documento está focado no uso da biblioteca ezPDF.
O Miolo encapsula as funcionalidades da biblioteca ezPdf, desenvolvida por R&OS Ltd, e disponível em http://www.ros.co.nz/pdf/ (a versão em uso no Miolo é 0.09). O encapsulamento permite usar todos os métodos das classes da biblioteca. A documentação das classes está disponível no arquivo docs/ezpdf.pdf.

Estrutura das classes

MReport
+----MCrystalReport
+----MezPDFReport
+----MJasperReport

CPdf (biblioteca ezPDF)
+----MCPdf
CEzPdf (biblioteca ezPDF)
+----MCEzCPdf

MGridColumn
+----MPDFReportColumn
MGridControl
+----MPDFReportControl
MGrid
+----MPDFReport