Tutorial - Programando com o MIOLO

Conceitos Básicos

Antes de iniciar a programação com o Miolo é fundamental compreender alguns conceitos básicos:

- Uma aplicação no Miolo é constituída de um ou mais **módulos**, dependendo de sua complexidade.
- Cada **módulo** possui um **nome**, uma **estrutura de diretórios** padronizada e um **arquivo de configuração**.
- A interação com o usuário é feita através da execução de **handlers**. Os handlers são pequenos trechos de código, responsáveis por receber a entrada do usuário, executar alguma ação e preparar o conteúdo que será exibido. Os handlers são armazenados em arquivos com a extensão .inc.
- Um módulo possui geralmente muitos handlers, sendo cada handler responsável por uma ação específica.
- A URL gerada pelo Miolo indica qual handler de qual módulo deve ser executado. Assim a URL

  http://host.miolo/index.php?module=exemplo&handler=main

  indica que deve ser executado o **handler** *main* do **módulo** *exemplo*.
- Os handlers podem ser encadeados, ou seja, um handler pode passar o controle para outro handler que, por sua vez, pode passar o controle para outro handler. Isto permite que cada handler cuide de uma tarefa bem específica na aplicação. Por exemplo, a URL

  http://host.miolo/index.php?module=exemplo&handler=main:curso:find

  indica que será executado o handler *main* do módulo *exemplo*. O handler *main* deverá executar o handler *curso*. Finalmente o handler *curso* deverá executar o handler *find*.
- Os handlers podem receber parâmetros através da URL (método GET) ou através de formulários (método POST). Por padrão, no Miolo o 3º parâmetro na URL é a variável **item**, que pode ser acessada dentro dos handlers. Outras variáveis podem ser passadas normalmente, como na URL

  http://host.miolo/index.php?module=exemplo&handler=main:curso:find&item=10&form=dados

- O conteúdo a ser exibido ao usuário deve ser colocado na área de conteúdo (**content**) do tema. Geralmente isto é feito pelo handler executando o método `$theme->setContent($conteúdo)`, onde $conteúdo é uma variável que contém o controle (ou conjunto de controles) a ser exibido.
- O Miolo incentiva a programação seguindo o padrão MVC (Modelo-Visão-Controle):
  - Modelo: representado pelas classes do domínio, armazenadas no diretório "classes" de cada módulo. As regras de negócio e o acesso a base de dados devem ser implementados dentro das classes do domínio.
  - Visão: representado pelos controles visuais (widgets), tais como painéis, formulários, grids, relatórios, etc. Implementam não apenas o conteúdo visual, mas também a lógica de apresentação.
  - Controle: representado pelos handlers, fazem a interação entre o Modelo, a Visão e o Usuário.

Módulo "exemplo"

Este texto mostra a criação de um módulo no Miolo a partir do zero. Os arquivos apresentados estão disponíveis no diretório "modules/exemplo". O módulo, ainda que incompleto, mostra várias funcionalidades do Miolo, desde a criação do arquivo de configuração, implementação da classe de domínio com acesso ao banco de dados, construção dos formulários e tratamento dos eventos gerados pelo usuário.
Devemos ressaltar que a forma como a aplicação foi desenvolvida não é imposta pelo framework. Ela reflete a maneira como as aplicações são desenvolvidas na UFJF, mas o desenvolvedor tem liberdade de criar outros modelos de aplicação. O objetivo da interface apresentada é manter a programação visual também "orientada a objetos", como todo o resto do framework.

Passo 1 - Modelagem

Para este tutorial estaremos adotando um modelo simples. Trata-se de uma empresa que oferece cursos de informática. É feito um cadastro dos dados pessoais dos alunos e a matrícula dos mesmos em um determinado curso. Para simplificar, cada aluno pode estar matriculado em apenas um curso.

Atributos:

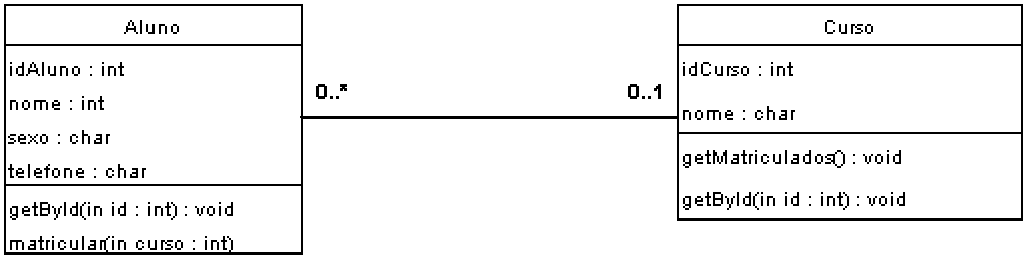
- Aluno: Código do aluno, Nome, Sexo, Telefone
- Curso: Código do Curso, Nome do Curso

Diagrama de Entidades e Relacionamentos (DER):

Tabelas (Banco de Dados SQLite):

```
CREATE TABLE Curso (
       IdCurso                      NUMBER         NOT NULL,
       Nome                         VARCHAR2(100));

ALTER TABLE Curso ADD CONSTRAINT PK_Curso PRIMARY KEY(IdCurso);

CREATE TABLE Aluno (
       IdAluno                      NUMBER         NOT NULL,
       Nome                         VARCHAR2(100),
       Sexo                         CHAR(1),
       Telefone                     CHAR(20),
       IdCurso                      NUMBER);

ALTER TABLE Aluno ADD CONSTRAINT PK_Aluno PRIMARY KEY(IdAluno);
ALTER TABLE Aluno ADD CONSTRAINT FK_Aluno1_Curso FOREIGN KEY(IdCurso) REFERENCES Curso;
```

Diagrama de classes (UML):



Passo 2 – Estrutura de diretórios do módulo exemplo

O módulo **exemplo** está contido no diretório <miolo>/modules/exemplo. Dentro deste diretório, estão os subdiretórios:

- **classes**: classes de domínio do módulo (aluno e curso)
- **classes/map**: mapeamento xml das classes, para a camada de persistência
- **forms**: formulários que serão apresentados ao usuário
- **grids**: grids para exibição do conteúdo das consultas ao banco de dados
- **handlers**: handlers do módulo
- **etc**: arquivo de configuração do módulo (module.conf)
- **html**: arquivos que devem ser acessíveis via web
- **html/images**: imagens usadas pelos controles visuais do módulo
- **sql**: scripts SQL e bancos de dados usados pelo módulo

Passo 3 – Arquivo de configuração do módulo

O arquivo modules/exemplo/module.conf contém as configurações para o módulo exemplo. Estas configurações sobrepõem as configurações globais do arquivo <miolo>/etc/miolo.conf.

É definida a configuração para acesso ao banco de dados (db.exemplo). Não será usada a autenticação de usuários (login.check=0). O módulo "exemplo" será o módulo inicial (não será executado o módulo padrão do Miolo).

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<conf>
    <options>
       <startup>exemplo</startup>
    </options>
    <db>
        <exemplo>
            <system>sqlite</system>
            <host>localhost</host>
            <name><miolo>/modules/exemplo/sql/curso0.db</name>
            <user>miolo</user>
            <password>miolo</password>
        </exemplo>
    </db>
```

```
    <login>
        <check>0</check>
    </login>
</conf>
```

Passo 4 – Classes de Domínio

As classes de domínio implementam as regras de negócio e encapsulam o acesso à base de dados. As regras de negócio são implementadas através dos métodos do objeto, geralmente através da interação com outros objetos. O acesso à base de dados pode ser feito através da camada DAO do Miolo (através das classes MDatabase, MSQL, MQuery entre outras) ou através da camada de persistência de objetos. Todas as classes de domínio herdam de MBusiness e são persistentes. As definições de classes ficam no diretório exemplo/classes. Os mapas XML usados pela camada de persistência ficam em exemplo/classes/map.
O código a seguir mostra a definição da classe Curso.

```
<?php

// As classes de dominio herdam de MBusiness e sao persistentes
// O nome da classe de dominio segue o padrao Business<modulo><classe>
// para permitir que classes com o mesmo nome sejam usadas em modulos diferentes
class BusinessExemploCurso extends MBusiness
{
    // atributos
    public $idCurso;
    public $nome;
    public $alunos;

    // método construtor
    // No Miolo as classes sao instanciadas através
    // de $MIOLO->getBusiness(modulo,classe,parametro)
    // se o parametro for um objeto, o construtor chama o metodo setData(parametero)
    // se o parametro for um valor, o construtot chama o metodo getById(parametro)
    function __construct($data=NULL)
    {
        // executa o construtor da classe pai; 'exemplo' é o nome da
        // configuracao do banco de dados
        // definida no arquivo miolo.conf
        parent::__construct('exemplo',$data);
        // exemplo de inicializacao de um atributo
        $this->nome = '';
    }

    // acessa o banco de dados para recuperar o objeto com id fornecido
    // e retorna o proprio objeto, já com os atributos preenchidos
    function getById($id)
    {
        $this->idCurso = $id;
        $this->retrieve();
        return $this;
    }

    // exemplos de metodos setter/getter
    function setNome($nome)
    {
        $this->nome = $nome;
    }

    function getNome()
    {
        return $this->nome;
    }

    // recebe um "transfer object" e preenche os atributos
    function setData($data)
    {
        $this->idCurso = $data->idCurso;
        $this->nome = $data->nome;
    }

    // recupera os registros no banco de dados e retorna um objeto MQuery
    function listAll()
    {
        $criteria =  $this->getCriteria();
        $criteria->addOrderAttribute('nome');
        return $criteria->retrieveAsQuery();
```

```
    }

    // recupera os registros no banco de dados segundo um criterio
    // e retorna um objeto MQuery
    function listByNome($nome)
    {
        $criteria =  $this->getCriteria();
        $criteria->addCriteria('nome','LIKE', "$nome");
        $criteria->addOrderAttribute('nome');
        return $criteria->retrieveAsQuery();
    }

    // recupera um array de os objetos "aluno" associados a este curso
    // e preeche o atributo $alunos com este array
    function getMatriculados()
    {
        $this->retrieveAssociation('alunos');
    }
}
?>
```

Passo 5 – Classe Handler

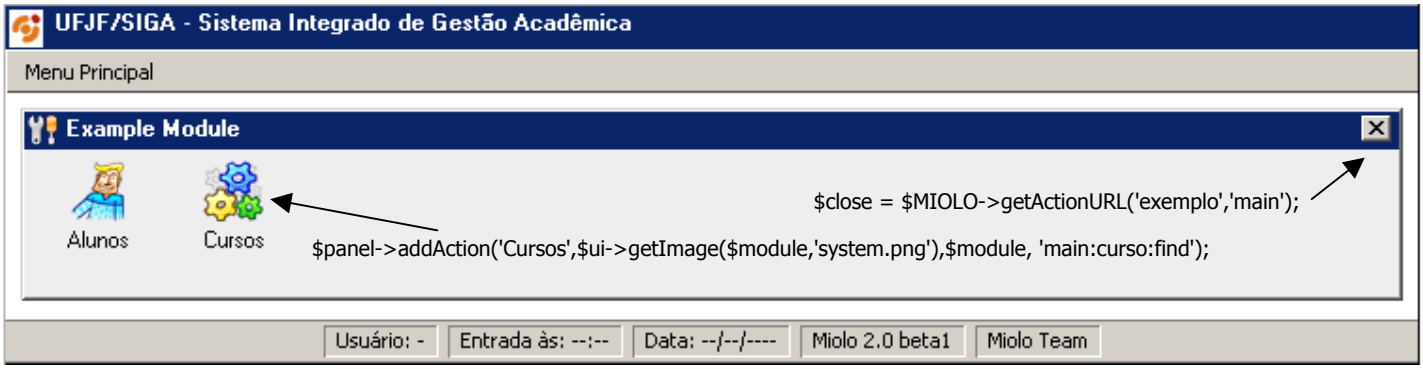
Todos os módulos têm uma classe que herda da classe MHandler e é nomeada como Handler<nome_do_módulo>. O Miolo usa os métodos desta classe para executar os handlers, ou seja, os códigos de programa responsáveis por tratar a requisição feita pelo usuário. Neste exemplo, não há nenhuma particularidade de tratamento para o módulo **exemplo**, então ele simplesmente herda da classe MHandler. Esta classe deve obrigatoriamente existir em todos os módulos.
O arquivo "handler.class" está localizado dentro do diretório <miolo>/modules/exemplo/handlers:

```
<?php
class HandlerExemplo extends MHandler
{
}
?>
```

Passo 6 – Handler Principal

Obrigatoriamente todos os módulos têm um arquivo "main.inc" que é o handler principal do módulo. Geralmente a função deste handler é mostrar um menu ou um painel de ícones que apontam para outros handlers.
É importante entender que **sempre** o handler main.inc do módulo vai ser executado, mesmo que ele não apareça explicitamente na variável *action* da URL.
O handler main.inc do módulo exemplo tem o seguinte conteúdo:

```
<?php
// obtem o objeto UI para acesso a metodos de interface (forms, grids, reports, images, etc.)
    $ui = $MIOLO->getUI();
// limpa a area de conteudo do tema
    $theme->clearContent();
// link a ser chamado se o usuario fechar o panel abaixo
    $close = $MIOLO->getActionURL('exemplo','main');
// instancia um novo panel, onde sao colocados os icones de acao
// a variavel global $module se refere ao modulo corrente (exemplo)
    $panel = new MActionPanel('pnlExemplo','Example Module','', $close, $ui->getImage(
$module,'tools1.png'));
// adiciona os icones de acao ao panel criado
    $panel->addAction('Alunos',$ui->getImage($module,'user.png'),$module, 'main:aluno:find');
    $panel->addAction('Cursos',$ui->getImage($module,'system.png'),$module,
'main:curso:find');

// o codigo seguinte obtem qual o proximo handler a ser executado na sequencia definida
// pelo parametro "action" da URL
    $a = $context->shiftAction();
// chama o proximo handler - $handled é falso se nenhum handler for executado
    $handled = $MIOLO->invokeHandler($module,$a);
// se nenhum outro handler foi executado, insere o panel na area de conteudo do tema
// se outro handler for executado, assume que esse outro handler vai preecher a area de
conteudo
    if (! $handled)
    {
        $theme->insertContent($panel);
    }
// inclui o menu principal da aplicacao (geralmente com links para outros modulos)
    include_once($MIOLO->GetConf('home.modules') .'/main_menu.inc');
?>
```



Passo 7 – Estrutura de handlers do módulo

Como dito anteriormente, o controle da execução das ações do módulo é feita através dos handlers. Cada handler pode ser visto como sendo a implementação de uma transação da aplicação ou de um determinado caso de uso. A idéia é que cada ação que o usuário possa fazer no sistema deve ter um handler correspondente.

O módulo **exemplo** apresentado neste tutorial possibilita que o usuário faça a manutenção do cadastro dos cursos. Assim, o usuário deve poder encontrar um curso específico, editar os dados de um curso, remover um curso ou criar um novo curso. O modelo de interface adotado agrupa estas ações em 3 handlers:

- Para editar ou remover um curso, o usuário deve primeiramente localizar (escolher) o curso. Esta ação é implementada através do handler *find.inc.*
- A inclusão de um novo curso é implementada através do handler *new.inc.*
- A edição/remoção de um curso, após este ter sido localizado, é implementada através do handler principal do curso, chamado *main.inc.*

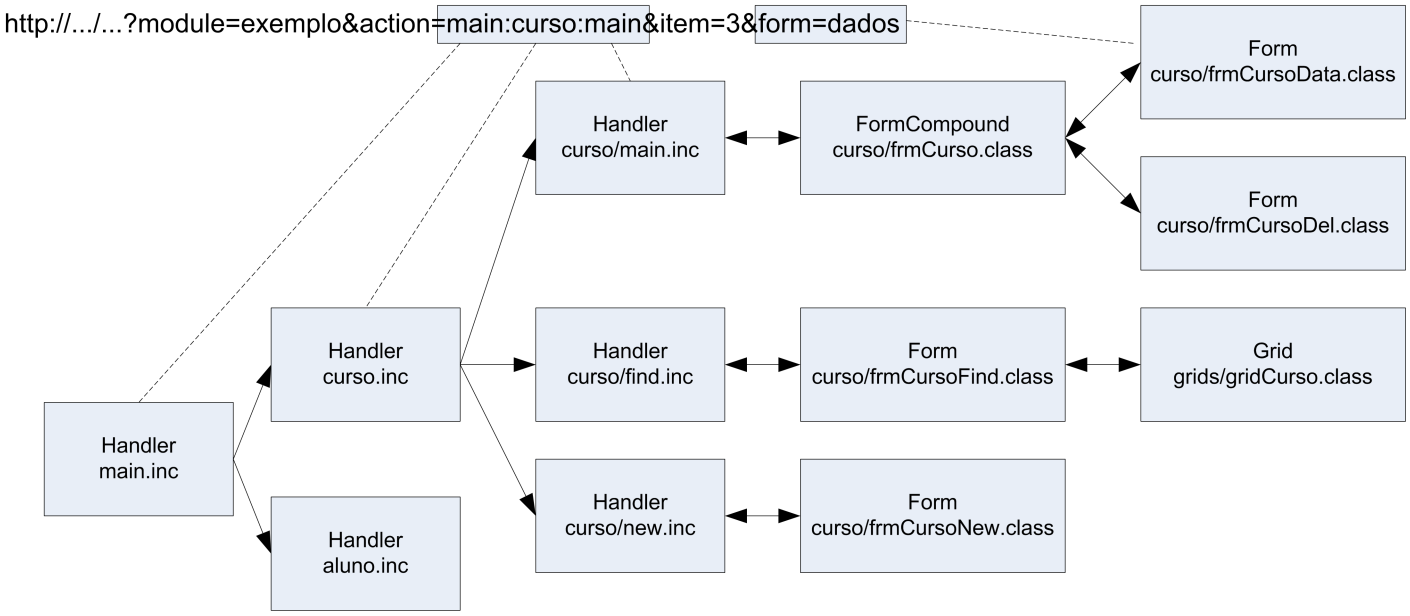
Observe que, como essas ações são comuns para praticamente todos os objetos de um sistema, foi criado um diretório **curso** abaixo do diretório **handlers**, a fim de agrupar os handlers específicos do objeto curso. Isto melhora a legibilidade, evitando que, em um sistema grande, tenhamos dezenas de handlers dentro do diretório **handlers**.

Para facilitar o acesso a estes handlers específicos de um objeto, por padrão é criado um handler como o nome do objeto (por exemplo, *curso.inc*) colocado dentro do diretório **handlers**. A única função deste handler é redirecionar para o handler de uma ação específica.

Na URL, a seqüência de ações é definida no parâmetro *action*, por exemplo:

> http://host.miolo/index.php?module=exemplo&handler=main:curso:main&item=3&form=dados
>
> Esta URL define que será executado o handler *main* do módulo *exemplo*. Este handler (visto no Passo 4) possui o código necessário para chamar o handler seguinte (neste caso, o handler *curso*). A única ação do handler *curso* será executar o handler *main*, localizado no diretório exemplo/handlers/curso.

Todos os handlers têm acesso às variáveis passadas como parâmetros. A figura a seguir mostra parcialmente a estrutura de handlers do módulo **exemplo**. O código dos handlers pode ser visto no diretório <miolo>/modules/exemplo/handlers.



Passo 8 – Formulários

Como visto na figura anterior (Passo 5) cada handler vai instanciar um formulário (form). O objetivo de um formulário é agrupar os controles visuais que são usados pelo usuário para executar determina ação. No caso deste tutorial, as ações são referentes à manutenção do cadastro de cursos (inclusão, remoção, edição, pesquisa). Para cada ação, geralmente haverá um formulário associado.

O Miolo possui 3 tipos principais de formulários:

- MForm: formulário básico, com um grupo de controles visuais e métodos para tratamento de eventos.
- MFormAjax: formulário que possua controles visuais usando Ájax.
- MCompoundForm: formulário que é composto por diversos tipos de objetos (Labels, Panels, Forms).

Na estrutura do módulo **exemplo** os formulários são colocados abaixo do diretório **forms**, em um subdiretório correspondente ao objeto a que eles se referem (por exemplo, **curso**). Como em relação aos handlers, o objetivo é melhorar a legibilidade, evitando que, em um sistema grande, tenhamos dezenas de formulários dentro do diretório **forms**. Um exemplo de formulário básico é o formulário para inclusão de um novo curso (exemplo/forms/curso/frmCursoNew.class):

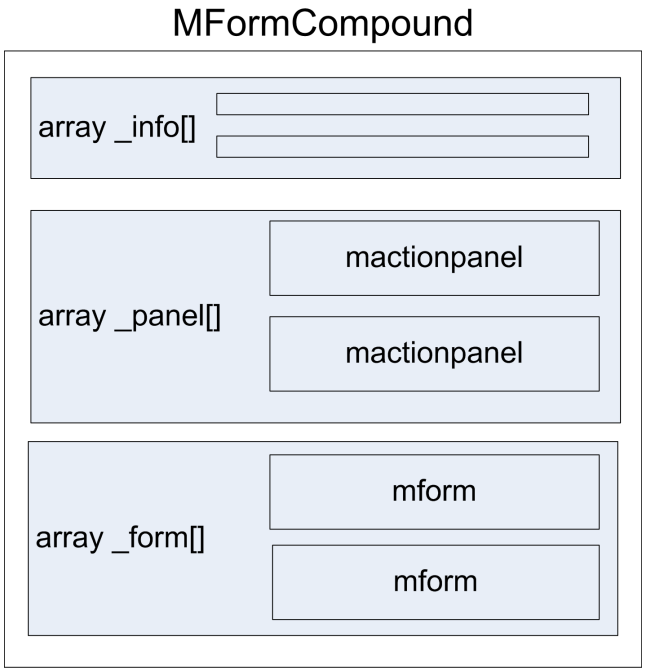
```
<?php

// frmCursoNew estende (herda) de MForm
class frmCursoNew extends MForm
{

// metodo construtor - no Miolo um formulário é instanciado geralmente
// através de $ui->getForm(modulo,form,parametro,diretorio)
    function __construct()
    {
        // executa o construtor da classe MForm
        // durante a execução do construtor é executado o método createFields
        parent::__construct('Incluir Novo Curso');
        // executa o metodo para tratamento de evento (caso haja algum evento, por exemplo,
um clique de botao)
        $this->eventHandler();
    }

// o metodo createFields e' executado automaticamente quando o formulario e' instanciado
// sua funcao básica é definir quais os controles visuais (fields),
// botoes (buttons) e validadores (validators) serao instanciados
    function createFields()
    {
        // define os controles do formulario
        $fields = array(
            new MTextField('nome','','Nome','25','Máx. 30 caracteres'),
        );

        $this->setFields($fields);

        // define os botoes do formulario
             $buttons = array(
             new MButton('btnNew', 'Incluir Curso')
        );
             $this->setButtons($buttons);

        // define os validadores do formulario
             $validators = array(
                    new MRequiredValidator('nome')
             );
             $this->setValidators($validators);
    }

// metodo executado por eventHandler() para tratamento do clique no botao btnNew
// a sintaxe <nome_botao>_click é padrao no Miolo
    function btnNew_click()
    {
        // define o acesso as variaveis globais
        // $MIOLO: classe principal do framework, com vários métodos utilitarios
        // $page: representa a pagina que esta sendo construida
        // $module: modulo corrente (neste caso, 'exemplo')
        global $MIOLO, $page, $module;

        // $data é um "transfer object", um objeto usado para transferir dados entre camadas
        // neste caso, os dados são os valores fornecidos pelo usuário através do formulario
        $data = $this->getData();
        // instancia um novo objeto curso
        $curso = $MIOLO->getBusiness($module,'curso');
```

```
        // define os valores dos atributos do objeto curso, com base no transfer object $data
        $curso->setData($data);
        try
        {
            // persiste o objeto (neste caso, é uma inclusao)
            $curso->save();
            // caso tudo ok, redireciona para a página de edicao do objeto
            $go = $MIOLO->getActionURL('exemplo','main:curso',$curso->idCurso);
            $page->redirect($go);
        }
        catch (Exception $e)
        {
            // caso ocorra algum erro, adiciona uma caixa de mensagem no formulario
            $this->addError($e->getMessage());
        }
    }
 }
?>
```

Os formulários do tipo MCompoundForm são usados para representar o acesso a um objeto. O formulário é dividido em 3 partes, cada uma delas representada por um array de controles visuais:

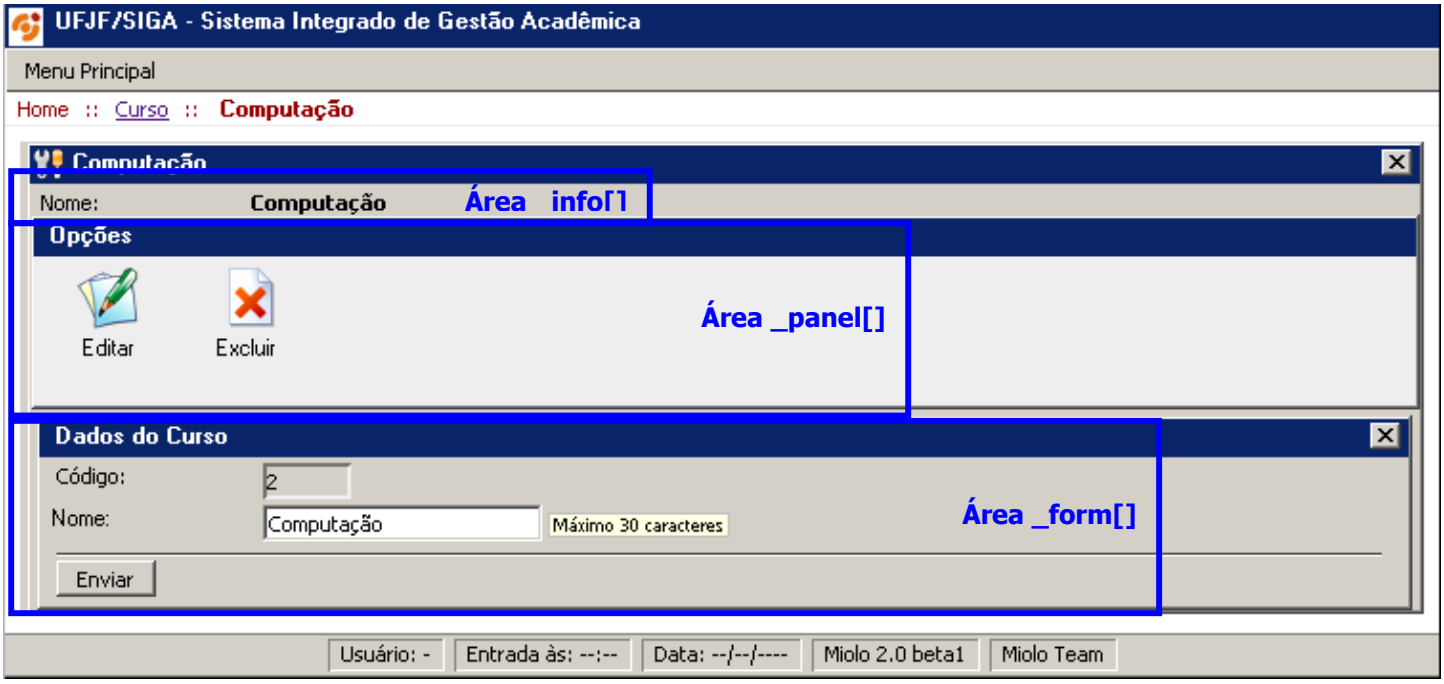
- _info[] : cada elemento exibe o valor de um atributo do objeto. Geralmente são usados controles MTextLabel.
- _panel[] : cada elemento é um MActionPanel que exibe os ícones que representam as ações possíveis do objeto. Geralmente cada ação está associada a um ou mais métodos do objeto.
- _form[] : cada elemento é um formulário ou um grid, associado a uma das ações expostas no MActionPanel.

A figura a seguir representa esquematicamente as partes do MCompoundForm.



Um exemplo de MCompoundForm é o formulário para acesso a um curso previamente selecionado (exemplo/forms/curso/frmCursoData.class):

```
<?php

// frmCurso estende (herda) de MCompoundForm
class frmCurso extends MCompoundForm
{
    protected $oid; // usado para representar o "object id" (identificador do objeto)
    protected $curso; // armazena o objeto curso recebido com parametro

    // metodo construtor: recebe o objeto que será acessado
    function __construct($objCurso)
    {
        $this->curso = $objCurso;
        parent::__construct();
        // barra de titulo do formulario
        $this->setTitle($this->curso->nome);
        // icone da barra de titulo
        $this->setIcon($this->manager->getUi()->getImage('exemplo','tools1.png'));
        // acao a ser chamada se o usuario fechar o formulario
        $this->setClose($this->manager->getActionURL('exemplo','main:curso:find'));
    }

// o metodo createFields e' executado automaticamente quando o formulario e' instanciado
    function createFields()
    {
        // variaveis globais
        global $module, $context;

        $curso = $this->curso; // apenas para diminuir a digitacao...
        MUtil::setIfNull($this->oid, $curso->idCurso); // copia o id do objeto

//form - define a area de formularios com base no parametro "form" passado na URL
        $ui = $this->manager->getUI();
        $form = MForm::getFormValue('form');
        switch ($form)
        {
            case 'dados' :
                 $this->_form[0] = $ui->getForm($module,'frmCursoData', $curso, 'curso');
                break;
            case 'excluir'     :
                $this->_form[0] = $ui->getForm($module,'frmCursoDel', $curso, 'curso');
                break;
        }
        // se for exibir um formulario, define a acao "fechar" para voltar ao McompoundForm
        if ($this->_form[0])
        {
            $this->_form[0]->setClose($this->manager->getActionURL('exemplo',   'main:curso',
$curso->idCurso));
        }

// panel - define a area de panel, com as ações possiveis para este objeto
        $action = $context->action; // obtem a variavel "action" passada na URL
        $this->_panel[0] = $panel = new MActionPanel('pnlCompound','Opções','',false);
        $panel->addAction('Editar',  $ui->getImage('exemplo','edit.png'),  'exemplo',$action,
$this->oid, array("form"=>"dados"));
        $panel->addAction('Excluir',$ui->getImage('exemplo','del.png'),'exemplo',    $action,
$this->oid, array("form"=>"excluir"));

// info - define a area de informacoes (atributos) do objeto
        $this->_info[0]    = new MTextLabel('txtNome',$curso->nome,'Nome');
    }
}
?>
```



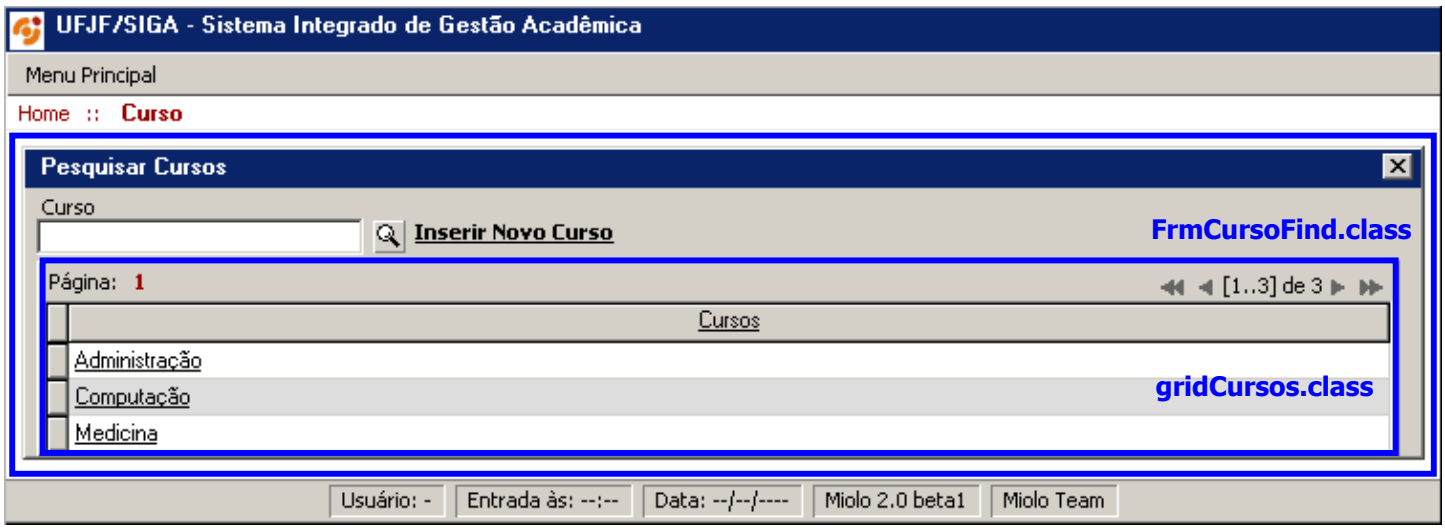
Passo 9 – Grid

O formulário frmCursoFind.inc permite localizar um curso específico com o qual o usuário vai trabalhar. Os cursos cadastrados são apresentados em um grid, de acordo com o argumento de pesquisa fornecido pelo usuário. Caso nenhum argumento seja fornecido são apresentados todos os cursos.

Os grids no Miolo permitem várias funcionalidades (paginação, colunas de ação, colunas de seleção, inclusão de controles nas linhas do grid, etc) Neste tutorial é apresentado um grid simples, construído com base nos resultados de uma query (uma consulta ao banco de dados).

```
<?php

// gridCursos herda de MDataGrid (um grid construido com base nos resultados de uma query)
class gridCursos extends MDataGrid
{
    function __construct()
    {
        global $MIOLO;

        // acao associada a cada link exibido no grid
        // #0# indica que o parametro deve ser substituido pelo valor da
        // coluna 0 de $query->result em cada linha
        $hrefCurso = $MIOLO->getActionURL( 'exemplo', 'curso:main', '#0#', array( 'form' =>
'dados' ) );

        // colunas do grid
        // MDataGridHyperlink renderiza a celula como um link com ação
        // definida por $hrefCurso
        // 'nome' indica que o conteudo sera obtido da coluna 'nome' de $query->result
        $columns = array(
            new MDataGridHyperlink('nome','Cursos', $hrefCurso, '100%', true, NULL, true )
        );

        // obtem o valor informado pelo usuario em frmCursoFind.class
        // onde este grid está inserido
        $filter = MForm::getFormValue('curso') . '%';

        // realiza a consulta ao banco de dados, atraves do metodo do objeto 'curso'
        // retorna um objeto da classe MQuery
        $objCurso = $MIOLO->getBusiness('exemplo','curso');
        $query = $objCurso->listByNome(strtoupper("'$filter'"));

        // acao que representa o proprio grid - usada como base na paginacao do grid
        $hrefGrid = $MIOLO->getActionURL($module,$self);

        // chamar o construtor da classe pai
        // 15 é o numero maximo de linhas exibidas por pagina
        parent::__construct($query, $columns, $hrefGrid, 15);

        // indica que os links do grid usarão o metodo GET (e não o metodo POST)
        $this->setLinkType('hyperlink');
```

```
    }

    function GenerateFooter()
    {
        // se não houver dados a serem exibidos, mostra a mensagem padrão
        if (!$this->data)
        {
            $footer[] = $this->GenerateEmptyMsg();
        }
        return $footer;
    }
}
?>
```



Passo 10 – Executar

No browser, deve ser acessada a URL

```
http://host.miolo/index.php/exemplo
```