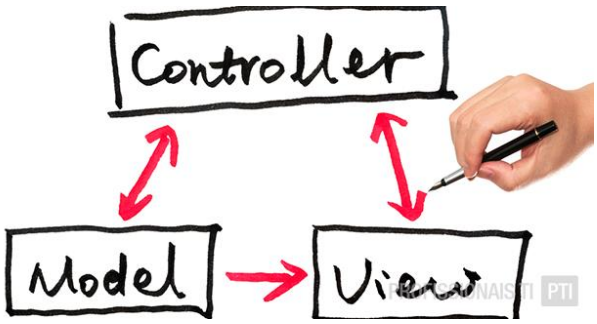
MVC

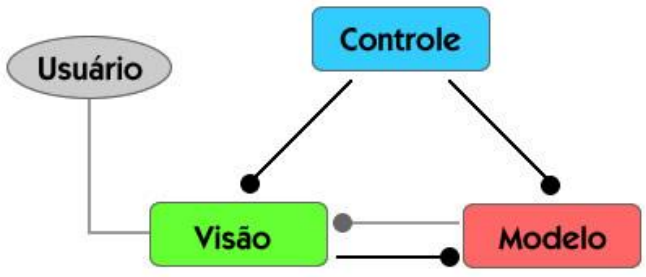
MVC é o acrônimo de Model-View-Controller (em português, Modelo-Visão-Controle) e que, como o próprio nome supõe, separa as camadas de uma aplicação em diferentes níveis. Ao contrário do que muitos desenvolvedores dizem, o MVC não é um Padrão de Projeto, mas sim um Padrão de Arquitetura de Software. Essa é uma das dúvidas que mencionei no enunciado do artigo.

Na época da universidade, ouvia muitos alunos comentarem que o "projeto do TCC foi desenvolvido utilizando o Design Pattern MVC", sem notarem que estavam errados. Apesar do MVC introduzir uma forma de desenvolvimento, ele não pode ser considerado um padrão de projeto, uma vez que este é especialmente relacionado com a arquitetura do software. Lembre-se de que existem 23 padrões de projeto (Design Patterns) e o MVC não está entre eles.



Pois bem, embora existam outros padrões de arquitetura de software – como Pipeline, Blackboard, Microkernel e Reflection – o MVC é um dos mais difundidos e utilizados pelos desenvolvedores principalmente pela funcionalidade e objetividade. O MVC define a divisão de uma aplicação em três componentes: Modelo, Visão e Controle. Cada um destes componentes tem uma função específica e estão conectados entre si. O objetivo é separar a arquitetura do software para facilitar a compreensão e a manutenção.

A camada de Visão é relacionada ao visual da aplicação, ou seja, as telas que serão exibidas para o usuário. Nessa camada apenas os recursos visuais devem ser implementados, como janelas, botões e mensagens. Já a camada de Controle atua como intermediária entre as regras de negócio (camada Modelo) e a Visão, realizando o processamento de dados informados pelo usuário e repassando-os para as outras camadas. E por fim, temos a camada de Modelo, que consiste na esseência das regras de negócio, envolvendo as classes do sistema e o acesso aos dados. Observe que cada camada tem uma tarefa distinta dentro do software.



Explicação do conceito do MVC

Certo, mas qual é a vantagem?

A princípio, pode-se dizer que o sistema fica mais organizado quando está estruturado com MVC. Um novo desenvolvedor que começar a trabalhar no projeto não terá grandes dificuldades em compreender a estrutura do código. Além disso, as exceções são mais fáceis de serem identificadas e tratadas. Ao invés de revirar o código atrás do erro, o MVC pode indicar em qual camada o erro ocorre. Outro ponto importante do MVC é a segurança da transição de dados. Através da camada de Controle, é possível evitar que qualquer dado inconsistente chegue à camada de Modelo para persistir no banco

de dados. Imagine a camada de Controle como uma espécie de Firewall: o usuário entra com os dados e a camada de Controle se responsabiliza por bloquear os dados que venham a causar inconsistências no banco de dados. Portanto, é correto afirmar que essa camada é muito importante.

Posso ter apenas três camadas no MVC?

Eis que essa é outra dúvida bastante questionada pelos desenvolvedores. Embora o MVC sugira a organização do sistema em três camadas, nada impede que você "estenda" essas camadas para expandir a dimensão do projeto. Por exemplo, vamos supor que um determinado projeto tenha um módulo web para consulta de dados. Aonde esse módulo se encaixaria dentro das três camadas?

Bem, este módulo poderia ser agrupado com os outros objetos da camada Visão, mas seria bem mais conveniente criar uma camada exclusiva (como "Web") estendida da camada de Visão e agrupar todos os itens relacionados a esse módulo dentro dela. O mesmo funcionaria para um módulo Mobile ou WebService. Neste caso, a nível de abstração, essas novas camadas estariam dentro da camada de Visão, mas são unidades estendidas.

Quando desenvolvi o meu projeto na faculdade, também me deparei com a necessidade de uma camada adicional: a camada de acesso ao banco de dados.

Mas o acesso ao banco de dados não é uma função da camada de Modelo?

Sim, é. Inclusive muitos profissionais preferem denominar a classe de Modelo como Business Object Model, responsável pela modelagem e acesso ao banco de dados ao mesmo tempo. Mas essa é uma premissa particular de cada desenvolvedor. No meu caso, achei conveniente manter as classes do projeto (modelagem) e o acesso ao banco de dados em camadas diferentes. Sendo assim, criei uma camada estendida da camada de Modelo para o acesso ao banco de dados, conhecida como DAO (Data Access Object) ou DAL (Data Access Layer). É uma camada bem simples, apenas com os objetos de conexão com o banco de dados e as instruções SQL, mas que proporcionou uma melhor organização no meu código.

Mas lembre-se: um projeto com várias camadas torna-se tão confuso quanto um projeto de uma camada só. Separe a implementação quando notar que a mesma camada está realizando duas ou mais tarefas diferentes, e que dividindo-as tornará o projeto mais compreensível.

Model (ou modelo)

O model é a camada que representa os seus dados, provendo meios de acesso (leitura e escrita) à esses dados.

A regra é simples: tudo que diz respeito à escrita, validação e leitura dos dados está dentro da camada model, não necessariamente dentro do model em si, mas dentro da camada model.

Vemos aqui um exemplo de model de produtos (que vai prover o acesso à tabela products, no banco de dados) no arquivo Product.php:

```
 <?php
2-


/**
* Model de produtos
*/
class Product extends AppModel {
```

```
/**
* Regras de validação
*
* @var array
*/
public $validate = array(
'name' => array(
'notEmpty' => array('rule' => 'notEmpty')
),
'description' => array(
 'notEmpty' => array('rule' => 'notEmpty'),
 'minLength' => array('rule' => array('minLength', 100))
),
'price' => array(
'decimal' => array('rule' => array('decimal', 2))
)
);



/**
* Lista dos produtos mais recentes
*
* @param int $limit Quantidade de produtos
* @param array $params Parâmetros extras de busca
*
* @return array
*/
public function recent($limit = 5, $params = array()) {
$params = Hash::merge(array(
'conditions' => array('Product.published' => true),
'order' => array('Product.created' => 'DESC'),
```

```
'limit' => $limit

), $params);



return $this->find('all', $params);

}



}
```

Aqui temos um model bem simples, onde definimos as regras de validação e um método que vai auxiliar a encontrar os produtos mais recentes.

Não vou entrar nos detalhes do CakePHP, além do mais acho que o código é bem auto-explicativo.

Somente através desse model será possível cadastrar e buscar produtos, e quando um usuário for cadastrar ou editar uma notícia, aquelas regras de validação devem ser respeitadas, ou seja:

O título não pode ser vazio

A descrição não pode ser vazia nem ter menos de 100 caracteres

O preço precisa ser no formato decimal, com 2 casas decimais

Para saber mais sobre models no CakePHP, consulte a documentação: http://book.cakephp.org/2.0/en/models.html

Agora podemos partir para a camada que vai fazer uso desse model, pedindo uma lista de produtos..

Controller (ou controlador)

No controller você tem métodos públicos que são chamados de actions, cada action é responsável por uma “página” do seu site/sistema. É o controller quem decide:

Qual model usar;

Quais pedidos fazer pro model;

Qual combinação de views será usada para exibir os dados retornados pelo model.

Atente que não é o controller que busca os dados (responsabilidade do model) e nem é ele quem exibe os dados (responsabilidade da view)… ele está ali justamente pra controlar os dois e a aplicação como um todo.

Aqui temos duas actons:

Uma action (index) vai pedir (para o model) a lista de produtos mais recentes;

Outra action (view) vai pedir (para o model) os dados de um único produto.

Para saber mais sobre controllers no CakePHP, consulte a documentação: http://book.cakephp.org/2.0/en/controllers.html

Agora podemos exibir esses dados na camada responsável por isso…

View (ou visualização)

É na view que o seu sistema interage com o usuário. Tudo que ele ver (HTML / XML / RSS / CSV) deve ser gerado e exibido através dessa camada. A view, por sua vez, tem como responsabilidade:

Manipular os dados para - e apenas para - exibição;

Exibir os dados.

Mas a view não faz nenhum tipo de escrita/persistência no seu sistema… ela não salva dados no banco, na sessão e etc.. E ela também não busca esses dados, pois eles devem - obrigatoriamente - serem entregues pela camada controller.