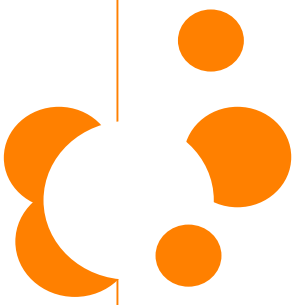


# Um olhar para 5 alternativas ao WAM

Declarativa:
Luís Carvalho
Carlos Elói
Mário Araújo
Miguel Calejo
CCG/ZGDV:
Paulo Doellinger

21/09/2011

versão 1.0
ID83

# TABLE OF CONTENTS

# 1 CakePHP

## 1.1 Instalação

A instalação do CakePHP é teoricamente simples. É necessário descarregar o zip e descompactar para o directório da aplicação. Ao utilizar no Windows, o mais demorado é a instalação do PHP no IIS, configuração do PHP na linha de comandos, configuração de URL Rewrites no IIS, variáveis de ambiente, etc.

Após a instalação, é necessário configurar alguns parametros da aplicação que estão distribuidos por 6 ficheiros do directório "config". Esta configuração é semelhante à realizada para o WAM no global.asa.

## 1.2 CRUD

Ao contrário do WAM, para obter uma interface, mesmo que básica, é sempre necessário escrever ou gerar algum código para criar models e controllers. Exemplo de código para a tabela Order da Northwind:

Controller

```
<?php
class Order extends AppModel {
        var $name = 'Order';
        var $useTable = 'Orders';
        var $primaryKey = 'OrderID';

}
```

Model

```
<?php
class OrdersController extends AppController {
        var $name = 'Orders';
        var $scaffold;

}
```

No exemplo anterior, é utilizado scaffolding, que analisa as tabelas da base de dados e cria listas e rows standard com controladores CRUD. Apesar do código do exemplo ser simples, está longe de ser o indispensável para uma aplicação. Através da análise da base de dados, o WAM oferece de raiz um conjunto de funcionalidades que são úteis para qualquer aplicação que no CakePHP requerem trabalho e muito linhas de código, como por exemplo: validação de dados; criação de lookups, links e finders.

No CakePHP, é frequente ser necessário personalizar lógica ou views, quando isso acontece, o scaffolding deixa de ser opção e requer a subsituição da instrução $scaffold por código, o que ainda provoca maior extensão e complexidade de código.

Para reduzir a escrita de código, o CakePHP fornece uma ferramenta para a linha de comandos que permite gerar o código minimo de um model, view e controller para criar os controladores CRUD. É necessário executar o wizard uma vez para cada model, view e controller de cada tabela. Se pretendermos que os campos dos rows sejam validados antes de serem enviados para a bd, é necessário especificar para cada coluna qual é o tipo de validação que pretendemos (alphanumeric, boolean, date, email, url, ...). Apesar da ferramenta na linha de comandos simplificar o processo, a inicialização de um base de dados com várias tabelas torna-se extramente demorada.

## 1.3 Relacionamentos entre tabelas

Ao contrário do WAM, que analisa automáticamente os relacionamentos existentes na base de dados, no CakePHP é necessário definir *associations* para criar ligações entre models. Os tipos possíveis de associations são:

- hasOne: A lista passa a conter um join com a tabela. Permite desenhar um row com campos de duas tabelas e com botões para acções das duas tabelas.
- belongsTo: Permite aceder a “Related user data”. Cria de forma automática links para FKs em listas e rows; ao inserir o row, troca o habitual texto livre por uma combo box; permite aceder a dados de colunas da FK; ...
- hasMany: Permite criar detail lists no row.

- hasAndBelongsToMany: Utilizado em relações N-N. Ao adicionar um row, permite seleccionar multiplas FKs (através de combo e ctrl ou checkboxs ou caixa de texto separando por virgula) e adiciona directamente na tabela de ligação. Permite criar detail lists no row.

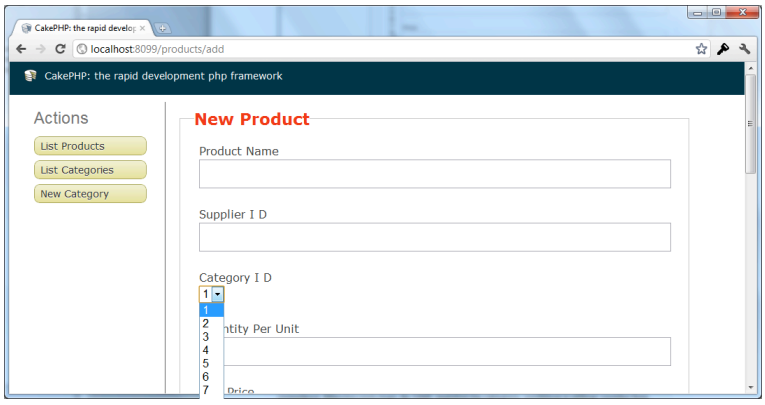
Comparando com o WAM, estas ligações proporcianam algumas funcionalidades que existem por defeito no WAM (ex: links em listas), outras que não existem (ex: inserir registos em tabelas de ligação na mesma janela), e outras que no WAM são conseguidas inserindo registos no WAMmodel.

## 1.4 Lookups

Para utilizar lookup column é necessário criar uma "association" no model. Por exemplo, para criar uma lookup em Northwind.Products.CategoryID é necessário acrescentar o seguinte código ao model de Products:

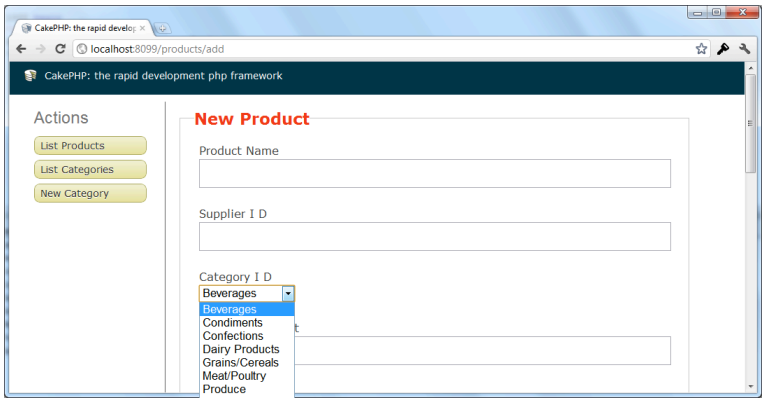
```
var $belongsTo = array(
        'Category' => array(
            'className'    => 'Category',
            'foreignKey'    => 'CategoryID'
        )
    );
```

Com esta "association", ao criar/editar um novo row, o campo CategoryID que por defeito é um *input text* é substituido por uma combo box com as chaves da category como se vê na imagem:

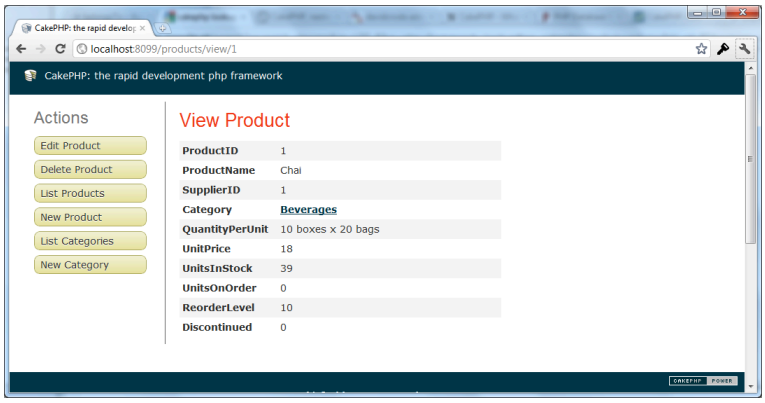



Em vez das chaves, podemos ver outra coluna, definindo a variável `$displayField` no model da Category. Por exemplo:

var $displayField = 'CategoryName';



Esta alteração provoca também que nos row e lista de produtos a Products.CategoryID seja substituida pela Categories.CategoryName:




No entanto, utilizando o scaffold, não parece ser possível utilizar várias colunas para o lookup tal como existe no WAM. Se não utilizar scaffold, criando uma *view* personalizada, poderá ser possível desenhar uma segunda combo box ou outro elemento com os valores de outra coluna, mas não parece ser uma feature nativa.

De notar que não parece exisitr outra forma de mostrar os dados além da combo box. Mesmo utilizando 1000 registos na category o CakePHP continua a utilizar uma combo.

## *1.5 Pormenores de interesse*

### 1.5.1 Routing

O CakePHP utiliza routing para mapear URL a acções. Ou seja, existe uma única página de entrada para todos os pedidos da aplicação (front controller), o sistema de routing determina qual é a função que deve ser executada com base no pedido efectuado e executa a acção. Esta funcionalidade torna o URL mais simples e também garante maior flexibilidade para alterar o nome de páginas sem quebrar ligações. Por exemplo:

- http://localhost/orders: lista de orders
- http://localhost/orders/view/10248: row do order com id=10248
- http://localhost/orders/delete/10248: elimina o row com id=10248

Para abrir o mesmo row no WAM o URL será mais complexo, http://localhost/WAMLibrary/Interface/standard.asp?witype=row&winame=dbo.Orders&key=10248, podendo ainda ser diferente se o row estiver personalizado.

### 1.5.2 Convenção de nomenclatura

O CakePHP supõe uma convenção de nomes para a base de dados, o que proporciona algum trabalho extra em base de dados que não seguem esse convenção como a Northwind. Mais grave, pelo menos para bases de dados já existentes, é a falta de suporte para chaves compostas.

### 1.5.3 Layout

O CakePHP segue a arquitectura MVC (Model-View-Controller) o que permite isolar melhor o layout. No que diz respeito a páginas com layout personalizado, parece garantir maior flexibilidade do que o WAM. Alterações na interface, como por exemplo, trocar a ordem entre duas colunas num row, são bastante mais simples no CakePHP.

## 1.6 Conclusão

Em resumo, o WAM permite gerar uma interface completa em menos tempo e com apenas alguns cliques. O CakePHP permite criar interfaces mais personalizadas do que o WAM, mas requer maior esforço de implementação e talvez de manutenção.

# 2 Symfony2

## 2.1 Instalação

A instalação do Symfony2 é identica ao CakePHP. É necessário descarregar o zip e descompactar para o directório da aplicação. O primeiro passo para a configuração do Symfony2, é abrir um URL da aplicação onde podemos ver a lista de problemas a corrigir: permissões, etc.

## 2.2 CRUD

É surpreendente não haver uma única referência sobre como criar rows e listas com controladores CRUD no "The Book" sobre o Symfony2. Ao pesquisar, encontramos frases de contribuidores da framework, escritas ainda este ano, tais como: "At it's current state Symfony2 is more of a foundation rather than a full fledged framework"[1].

Há dois meses atrás, foi lançado uma versão beta, de um generator que prentende corrigir essa lacuna. Ainda não existe muita documentação e vem com um aviso, "The code is quite new and probably comes with a lot of bugs. Let's enhance it in the coming weeks!"[2], mas percebe-se o generator já permite gerar código para construir rows e listas com CRUD controllers.

Contúdo, o código necessário para criar uma interface, através do generator ou não, é sempre muito extenso. Não contém nenhuma opção como o scaffold do CakePHP para simplificar o código necessário para construir a interface. Se utilizando o scaffold no CakePHP, quando comparado com o WAM, criar e manter uma interface simples parecia trabalhoso, no Symfony2 torna-se tedioso.

A excução do generator é feito através de um aplicação PHP na linha de comandos, e requer os seguintes passos:

1. Criar uma Bundle
2. Criar Entities
3. Executar o CRUD generator. O CRUD generator gera o código para criar o routing, controllers e templates necessários.

### 2.2.1 Bundle

No Symfony2, todo o código tem de estar associado a uma Bundle. Na prática, uma Bundle, é um directório que contém tudo relacionado com uma feature: classes PHP, configuração, CSS, javascript, etc. O objectivo é criar flexibilidade para incluir third-party bundles ou distribuir as nossas bundles. A criação de uma Bundle é habitualmente feita através de um script PHP, na linha de comandos, que gera os directórios e configurações iniciais.

### 2.2.2 Entity Class

A Entity é uma class utilizada para abstrair a aplicação da base de dados. Na class devem ser especificadas as colunas da tabela e tipo de dados (type e lenght), validações, etc. É possível criar uma entity através da aplicação na linha de comandos, mas parece ser sempre necessário introduzir para cada coluna o nome e tipos de dados. Portanto, se após o desenvolvimento da aplicação for necessário introduzir alguma coluna ou alterar o tipo de dados, também vai ser necessário alterar a entity.

[1] http://spf13.com/post/symfony2
[2] http://symfony.com/blog/symfony2-getting-easier-interactive-generators

### 2.2.3 Routing

Criar uma nova página no Symfony2 envolve no minimo dois passos: criar o route e criar o controller. O route associa o URL ao controller. O controller é a função PHP que recebe o pedido e gera a resposta (HTML, XML, etc). Opcionalmente, podem ser criado um template para isolar o layout do controller.

O sistema de routing é idêntico ao CakePHP. Existe um front controller que recebe todos os pedidos e faz routing para função correspondente ao pedido. A configuração de routing no Symfony2 é definida no ficheiro de routing (em YAML, XML ou PHP) com instruções como:

```
contact:
    pattern:  /contact
    defaults: { _controller: AcmeDemoBundle:Main:contact }
```

Neste exemplo, ao abrir a página /contact é invocada a função *contact* do controller *Main* da bundle *AcmeDemo*.

## *2.3 Pormenores de interesse*

### 2.3.1 Flash messages

Tanto o CakePHP e o Symfony2 têm o conceito de flash messages. Através destas mensagens é possível enviar para o utilizador mensagens curtas do género: “Não tem permissão para executar esta operação”. Estas flash messages tanto podem aparecer num elemento da página, definido no template para o efeito, como podem aparecer no ecrã durante uns segundos seguido de um redirect para outra página. No WAM este tipo de mensagens normalmente são construidas com recurso a uma alert, o que não parece ser tão elegante.

### 2.3.2 Templates

O Symfony2, faz uma grande aposta nos templates e leve o conceito de template mais longe. É possível existirem vários templates para gerar vários tipos de formatos (HTML, XML, CSV, LaTeX ...) para o mesmo controller.

Os templates podem ser criados em PHP ou Twig. O Twig é um template engine realizado para expressar a apresentação, sem conter lógica. Permite que um web designer consiga criar/editar a apresentação sem preocupar-se com o que está por trás. No Symfony2, utilizam *template inheritance,* um template pode extender outro, em que no template filho, só é necessário criar os blocos que vão fazer override aos blocos do template pai.

## *2.4 Conclusão*

Em resumo, o Symfony2 oferece muita flexibilidade, talvez ainda mais do CakePHP. Permite a um webdesigner desenvolver e reutilizar templates de forma simples e sem preocupar-se com as outras camadas da aplicação. É possível desenvolver o mesmo controller para criar um ficheiro HTML, XML, json, etc, onde a única diferença será o template. No entanto, é sempre necessário escrever muito código. Sente-se a ausência de automatização e funcionalidades pelo menos no que diz respeito a aplicações com controladores CRUD.

# 3 Ruby on Rails

A Framework Ruby é baseada na arquitectura de sofware MVC (Model-view-controller[3]). O desenvolvimento é feito na camada intermédia, definindo-se classes que representam modelos de dados e respectiva lógica. A interface é desenhada pelo "middle-user" / programador ou serve-se do Rake que gera os ficheiros das views para cada modelo, com interface mínima (listas e campos).

- Target Users: Programadores (geeks)
- Flexível no desenho do layout. Rígido / mais trabalhoso na definição da BD.

## 3.1 Instalação

Assumindo a instalação prévia do Ruby e RubyGems, a configuração poderá ser feita através da linha de comandos. Por exemplo, para criar uma aplicação designada de blog:

```
$ rails new blog   # criar nova app Rails, dentro da pasta blog. É criada estrutura de diréctorios
$ cd blog
$ bundle install   #instalar dependências da aplicação (gems) definidas no gemfile
```

A configuração dos parametros da base de dados é realizada através da edição do ficheiro "config/database.yml". Por exemplo:

```
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000
```

O Rails permite ainda a utilização de bases de dados postregresSQL ou MySql após a instalação das respectivas gems.

O Rails tem a vantagem de não necessitar de um Webserver dedicado assim como um SGBD em ambiente de desenvolvimento. O Rails tem o seu próprio "WebServer virtual" (um bocado há semelhança do .NET + Visual Studio), o WEBrick, assim como não necessita de uma instância de MySQL ou Postregre, bastando a instalação da GEM SQLlite, uma base de dados auto contida na própria app Rails.

O WAM necessita forçosamente de correr num IIS ( >= 5) e da existência de MS SQL Server (versão Express no mínimo) ou Informix.

## 3.2 CRUD

No WAM, uma vez que definimos o modelo de dados primeiro, ao instalar o WAM, "ganha-se" uma interface CRUD de forma gratuita sem qualquer linha de código. O WAM gera automaticamente listas referentes às tabelas, com filtros para refinar pesquisas, campos de ordenação e ainda a possibilidade de acrescentar / remover colunas à lista "on-the-fly". Também é gerado em run-time o formulário de inserção ou edição dos registos.

No Rails, pode optar por criar as classes de cada Model (representação de uma entidade de dados) ou utilizar a ferramenta scaffolding. Exemplo:

```
$ rails generate scaffold Post name:string title:string content:text
```

Esta linha de código gera vários ficheiros relativos a template, model class, dados "dummy" para serem inseridos posteriormente, controlador e migração.

[3] http://en.wikipedia.org/wiki/Model-view-controller

Em seguida correr a migração para o novo Modelo ser reproduzido na Base de Dados.

```
$ rake db:migrate
```

É sempre necessário editar as classes quando são acrescentados novos modelos, caso existam relações.

Comparando os dois, sem esforço algum (para além do desenho relacional da base de dados) para o programador, o WAM oferece as operações básicas CRUD para cada entidade de dados, respeitando as relações entre tabelas, restrições. O Rails não gera qualquer tipo de interface, o utilizador terá de melhorar o aspecto através do recurso a Stylesheets, ao contrário do WAM que, por defeito, tem um aspecto próprio base que pode ser melhorado / customizado (até certo ponto).

## *3.3 Customização*

### 3.3.1 Labels

No o WAM, o WAMAdmin é a chave. Quando estamos no modo de edição podemos alterar "just in time" labels, acrescendar lookupfields (detalhe em chaves estrangeiras), agrupar campos ou adicionar listas de detalhes para dados relacionados.

No Rails, toda a customização é manual, acedendo às propriedades / métodos do ActiveRecord.

### 3.3.2 Lookups

O WAM permite criar uma WAMDetailList através do WAMAdmin com poucos cliques. No Rails, é necessário obter o array de detalhes e iterar sobre o mesmo no template. Se tivermos um objecto Post, que representa o registo, o array de comentários é obtido da seguinte forma:

```
@post.comments
```

Itera-se da seguinte forma:

```
<h2>Comments</h2>
<% @post.comments.each do |comment| %>
  <p>
    <b>Commenter:</b>
    <%= comment.commenter %>
  </p>
  <p>
    <b>Comment:</b>
    <%= comment.body %>
  </p>
<% end %>
```

### 3.3.3 Cascading

Cenário: supondo uma base de dados com uma tabela "post" e outra "comment", em que um post pode ter vários comment, aquando a eliminação de um POST pretende-se que todos os comentários associados sejam também eliminados. Numa "aplicação" normal seriam necessárias algumas linhas de código antes do "Delete" ou implementação de um trigger a disparar imediatamente antes do registo ser eliminado. Tanto no WAM como no Rails este processo é facilitado.

No WAM, o WAMModel possui uma tabela "WAM_DELETE_RULES" para implementação de regras de Delete. Definimos a FK onde queremos adicionar a regra de DELETE, neste caso a entre "comment" e "post".

No Rails, na model class "Post", quando se define a relação com "Comment", podemos criar uma regra de dependência:

```
has_many :comments, :dependent => :destroy
```

### 3.3.4 Procedimentos

No WAM, para criar um procedimento é necessário criar um registo em WAM_PROCEDURE_CALL. Para criar uma custom ASP, utilizam-se objectos WAM para acesso / manipulação de dados de forma a criar a página desejada

No Rails, a criação de uma nova action é feita no Controller e numa view associada. É mais flexível que o WAM uma vez que este processo está contido na própria arquitectura MVC.

# 4 ASP.Net MVC

## 4.1 Requisitos

O ASP.Net MVC usa o sistema operativo Windows, .NET 3.5 SP1 e Visual Studio/Webdeveloper 2008. Permite utilizar vários servidores de dados mediante a instalação de *plug-ins*.

## 4.2 CRUD

A inicialização de uma aplicação envolve os seguintes passos:

1. Criação de um "Model" com ligação a uma BD, selecção dos objectos SQL a serem usados na aplicação.
2. Criação do "Controller" para cada objecto SQL (tabela/view) e da respectiva "View"

Existe a possibilidade de instalação do MVCScaffolding que permite criar "Controllers" e "Views" automaticamente para os objectos SQL utilizados.

No WAM, após instalação, temos acesso à aplicação sem ter de fazer mais nada. Nesta framework, temos de criar os ficheiros do "Controller" e "View". Assim, julgamos que o WAM oferece uma solução melhor para inicializar a aplicação.

## 4.3 Customização

Uma vez que utiliza o modelo MVC, a customização é bastante simples, basta alterar as "Views". O WAM, permite a criação de asp's customizadas, mas não são tão fléxiveis como as "Views" uma vez que têm grande parte do "Controller" incluído. O modelo MVC tem a vantagem de a costumização ser independente dos controladores de dados.

### 4.3.1 Lookups

A forma que encontrámos para obter informação da tabela master, durante a criação de um registo novo na tabela de detalhe, consiste em construir a informação a mostrar. Por exemplo, fazer uma combo box com essa informação.

```
<%= Html.DropDownListFor(model => model.IdCategoria, ViewData["categoria"] as
IEnumerable<SelectListItem>)%>
```

Para consultar o registo da master na tabela de detalhe basta adicionar a seguinte linha:

```
<%= Html.TextBoxFor(model => model.dbCategoria.Nome) %>
```

Esta linha, feita na "View", permite, na visualização de um artigo, ver o nome da categoria que está associada à FK.

## 4.4 Alterações no modelo

No WAM, quando existem alterações no modelo de dados, basta fazer "Refresh" da aplicação para carregar as alterações. Dependendo das customizações feitas, poderá ser necessário rever a(s) asp(s) customizadas.

No ASP.Net MVC, sempre que são realizadas alterações no modelo é necessário reproduzi-las no "Controller" e na "View".

# 5 Django

## 5.1 Requisitos

Pode ser instalado em qualquer sistema operativo que permita a instalação do Python. Utiliza um servidor web feito à medida. Suporta os servidores de dados MySQL, PostgresSQL, MS SQL (ODBC), Oracle, Sqlite, Sybase SQL, IBM DB2 e Firebird.

## 5.2 CRUD

A inicialização de uma aplicação envolve os seguintes passos:

1. Criar uma aplicação usando os scripts do Django.
2. Alterar o ficheiro de configuração, actualizar parametros da BD.
3. Criar os "Controllers" e "Views" para os objectos SQL

Mais uma vez, ao comparar o WAM com uma framework que utiliza a arquitectura MVC, o WAM destaca-se não necessitar da criação de ficheiros do "Controller" e "View". Assim, mais uma vez, julgamos que o WAM oferece uma solução melhor para inicializar a aplicação.

## 5.3 Customização

Tal como as soluções anteriores, uma vez que utiliza o modelo MVC, a customização é bastante simples, basta alterar as "Views".

### 5.3.1 Lookups

Nas tabelas com FK's aparece uma combo com dados da tabela principal com a possibilidade de adicionar registos.

Para adicionar uma lista embebida é necessário adicionar a seguinte linha:

```
inlines = [ChoiceInline]
```

Onde "ChoiceInline" é uma "class", e tem o seguinte código:

```
model = Choice
```

```
extra = 3
```

O "model" é que a tabela de detalhes e o "extra", adiciona três linhas em branco, à lista embebida, para se poder adicionar registos.

Este funcionamento verifica-se no interface "admin", por exemplo na página do Django. No entanto, na interface pública, o exemplo "não tem nada".