Frameworks para Desenvolvimento Rápido de Aplicações Web: um Estudo de Caso com CakePHP e Django

Adriano Pereira[1], **Vinícius Vielmo Cogo**[1], **Andrea Schwertner Charão**[1]

[1] Programa de Educação Tutorial (PET)
Curso de Ciência da Computação - Universidade Federal de Santa Maria (UFSM)
Campus UFSM – 97105-900 – Santa Maria – RS – Brasil

{apereira, vielmo, andrea}@inf.ufsm.br

Abstract. *In recent years, several frameworks for rapid Web development have emerged. Facing the diversity of such frameworks, it is important for developers to know use cases of such tools, before taking decisions about which of them to choose. This work is an analysis of the rapid Web development frameworks CakePHP and Django, based on the case study of a stock management system. To do so, we built two versions of this system, one using Django and another using CakePHP, allowing to compare different characteristics of both frameworks.*

Resumo. *Nos últimos anos surgiram vários* frameworks *que aceleram o desenvolvimento de aplicações para a Web. Face à diversidade existente, torna-se relevante conhecer casos de utilização de tais ferramentas antes de optar-se por uma ou outra. Este trabalho faz uma análise dos* frameworks *CakePHP e Django para desenvolvimento rápido de aplicações Web, utilizando como base o caso de um sistema gerenciador de estoque de materiais. Para isso, foram construídas duas versões do sistema, uma com Django e outra com CakePHP, permitindo comparar diferentes características de ambos os* frameworks*.*

1. Introdução

No atual mercado de desenvolvimento para a Web, muitas aplicações precisam ser desenvolvidas em um espaço de tempo reduzido. Para alcançar velocidade e produtividade no desenvolvimento, pode-se utilizar *frameworks* que facilitam a reutilização e a geração de código. No entanto, a diversidade de linguagens de programação e metodologias de desenvolvimento leva à existência de uma grande quantidade de *frameworks*, o que pode gerar incertezas quanto à escolha do *framework* correto para uma determinada tarefa.

Com estas questões em foco, o presente trabalho objetiva analisar funcionalidades presentes em dois *frameworks* recentes, CakePHP e Django, que são distribuídos como Software Livre e mantidos por uma comunidade ativa de desenvolvedores. Essa análise é feita por meio de um estudo de caso: um sistema gerenciador de estoque de materiais. Estima-se que este caso reúna características comuns a sistemas de informação semelhantes e, por isso, espera-se que o trabalho seja de utilidade àqueles desenvolvedores que procuram ferramentas de desenvolvimento rápido dentre as tantas existentes.

O presente artigo encontra-se divido em 5 seções. A seção 2 aborda alguns conceitos iniciais a respeito de *frameworks* de desenvolvimento rápido para a Web. A seção 3 descreve a aplicação escolhida para o estudo de caso, servindo como base para a seção 4, que discute a relação entre a aplicação, suas necessidades e os *frameworks* utilizados. A seção 5, por fim, apresenta algumas considerações sobre o trabalho e suas contribuições.

2. *Frameworks* para Desenvolvimento Rápido de Aplicações Web

Um *framework* consiste em um conjunto de classes, interfaces e padrões dedicados a solucionar um grupo de problemas através de uma arquitetura de programação flexível e extensível [Govoni 1999]. Pode-se ainda definir um *framework* como sendo uma pequena aplicação completa com uma estrutura estática e outra dinâmica, desenvolvidas para resolver um conjunto restrito de problemas [Fayad 2000]. Esta última definição remete ao Princípio de *Hollywood* (*Holywood Principle*), o qual é definido através da seguinte regra: "Não nos chame, nós chamaremos você."("*Don't call us, we'll call you.*"), que trata de uma forma resumida dos conceitos de Inversão de Controle (*Inversion of Control*) [Martin 1996]. Essa é uma das principais características dos *frameworks*, a qual os diferenciam de simples bibliotecas [Fayad and Schmidt 1997].

Existem algumas funcionalidades consideradas desejáveis em *frameworks* caracterizados como voltados ao desenvolvimento rápido de aplicações. Dentre elas, encontram-se a implementação automática de funções CRUD (*Create, Retrieve, Update and Delete*), ORM (*Object-Relational Mapping*) e MVC (*Model-View-Controller*). A metodologia deste trabalho baseia-se na análise destas funcionalidades, aplicadas ao caso em questão.

3. Sistema Gerenciador de Estoque

O caso considerado neste trabalho consiste em um sistema para gerenciamento de estoque de materiais. Este caso foi escolhido através de reuniões com potenciais usuários finais. Outras justificativas para sua escolha foram a popularidade deste tipo de sistema e seu variado conjunto de requisitos que podem ser supridos com o uso de *frameworks*.

O banco de dados da aplicação foi modelado a partir dos casos de uso definidos nas reuniões, bem como a partir de funcionalidades comuns a sistemas gerenciadores de estoques. O modelo da aplicação pode ser definido em quatro divisões principais: Pessoas, Instituição, Produtos e Entrada/Saída. Essas divisões podem ser visualizadas na Figura 1.

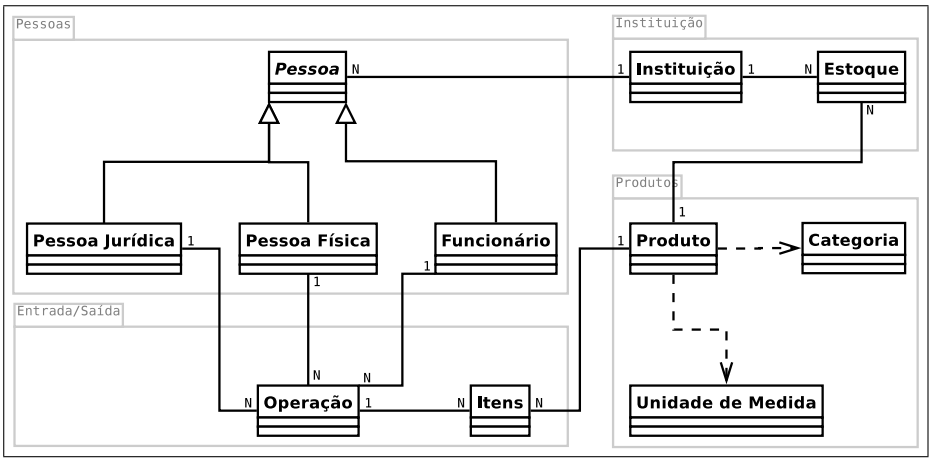

Figura 1. *Diagrama de Tabelas da Modelagem dos Dados para a Aplicação.*

A primeira seção, Pessoas, consiste das subdivisões Pessoa Física, Pessoa Jurídica e Funcionários e engloba todos aqueles que possuem relação com a instituição detentora

do estoque. A segunda seção, Instituição, está subdividida em Instituição propriamente dita e Estoques que esta possui. Na terceira seção, Produtos, encontram-se os próprios Produtos, Categorias de Produtos e Unidades de Medida. A quarta e última seção, Entrada/Saída, corresponde às operações de entrada e saída do estoque, representadas através de listagens de itens, de forma semelhante a uma nota fiscal.

4. Aplicando os *Frameworks* ao Caso do Gerenciador de Estoque

Através do modelo apresentado na seção anterior, pode-se inferir que as funcionalidades necessárias para a aplicação representam os principais requisitos desejáveis em um *framework*. Dentre estes requisitos, podemos destacar alguns, como por exemplo, a criação das operações CRUD, hierarquia entre tabelas, relacionamentos NxN (*ManyToMany*), divisão do projeto em camadas (trabalho em equipe) e mapeamento objeto-relacional. Tendo em vista esses requisitos, foram selecionados dois *frameworks*, os quais, além de possuir documentação completa e clara, já possuem casos de sucesso em uso. Os *frameworks* selecionados, CakePHP e Django, utilizam respectivamente as linguagens de programação PHP e Python. Decidiu-se também por iniciar o desenvolvimento de dois protótipos em paralelo para futuras análises e comparações dos mesmos.

A seguir encontram-se alguns dos principais requisitos da aplicação, analisados e implementados nos dois protótipos, permitindo a comparação entre as implementações com os dois *frameworks*.

4.1. MVC - Model-View-Controller

A abordagem MVC (*Model-View-Controller*) consiste em separar modelagem, apresentação e fluxo dos dados em camadas distintas. Nos dois *frameworks* analisados, o MVC encontra-se presente, porém com pontos de vista diferentes.

Tanto para o CakePHP quanto para o Django, a camada *Models* é considerada a representação da modelagem dos dados, como pode ser observado na Figura 2, para o caso dos ”Produtos”do estoque. Porém quanto às definições de *Views* e *Controllers* não existe um consenso entre os dois *frameworks*. Para o CakePHP, a camada *Views* consiste em como os dados serão apresentados, enquanto para o Django consiste em quais dados serão apresentados, sendo que, neste último, a forma em que serão apresentados os dados é descrita em *templates*. Também no CakePHP, a camada *Controller* é aquela que controla o fluxo de dados entre o banco de dados e o resultado visual destes. No Django, por sua vez, esta camada é considerada como o *Framework* propriamente dito. No CakePHP, a distinção em *Models*, *Views* e *Controllers* necessita, para cada tabela do banco de dados, de três arquivos (um para cada componente da tríade), enquanto no Django o *Models* ocupa apenas um único arquivo para todas as tabelas do banco de dados.

4.2. ORM - Object-Relational Mapping

O Mapeamento Objeto-Relacional consiste em técnicas de conversão entre o modelo orientado a objetos usado na aplicação e o modelo relacional usado no banco de dados. Uma das formas de comunicação entre a aplicação e o banco de dados é através do uso dos conceitos de *Active Record Pattern* [Fowler 2002], ou seja, apenas invocando métodos dos objetos referentes a cada tabela da base de dados, sem a utilização direta de comandos SQL. Ambos os *frameworks* analisados neste trabalho implementam essa funcionalidade.

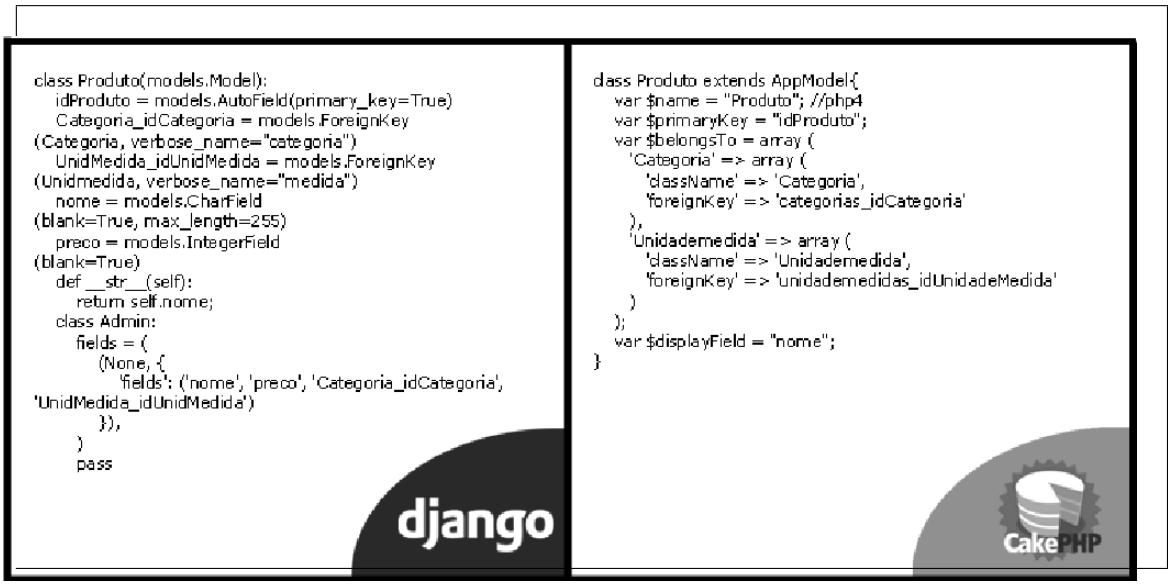

Figura 2. *Códigos de exemplo do modelo Produtos no Django e no CakePHP.*

O CakePHP realiza o mapeamento de cada tabela do banco de dados relacional em uma classe da aplicação que estende a classe *AppModel*, incluída no *framework*. O mesmo ocorre no Django, onde se deve estender a classe *Model*. Neste aspecto, os *frameworks* diferem quanto à organização destas classes em arquivos: o CakePHP utiliza um arquivo para cada classe/tabela, enquanto o Django incorpora todas as classes/tabelas no mesmo arquivo. Com o CakePHP, não há necessidade de definir nos arquivos todos os atributos existentes na tabela do banco de dados, cabendo ao desenvolvedor a preocupação apenas com detalhes envolvidos pela tabela, como relacionamentos (1x1, 1xN, NxN); já o Django requer que os campos sejam completamente especificados no arquivo, podendo ser gerados de forma automática a partir do esquema do banco de dados. É possível, no Django, realizar alterações na estrutura destas tabelas através da aplicação, o que não acontece no CakePHP.

Os relacionamentos são expostos, no CakePHP, através de associações do tipo *hasMany/belongsTo*, *hasOne* e *hasAndBelongsToMany*. Estas associações são implementadas por atributos, que contêm informações a respeito desses relacionamentos, como chave estrangeira, nome das classes envolvidas, entre outros. O Django implementa relacionamentos definindo que um campo é chave estrangeira e indicando seu tipo de dado, se é um auto incremento, entre outros. Dessa forma, pode-se agilizar o desenvolvimento da aplicação estudada, pois esses relacionamentos estão presentes entre várias tabelas; por exemplo, no caso em questão, operações e produtos possuem um relacionamento NxN através de itens, que pode ser realizado com a definição *hasAndBelongsToMany* no CakePHP e *FieldName = models.ManyToManyField(ClassName)* no Django.

Em formulários onde há relacionamentos, ambos os *frameworks* exibem, por padrão, a chave estrangeira de cada uma das tuplas da tabela relacionada, em campos de seleção. É possível alterar, para determinada tabela, o campo de exibição: no CakePHP, atribuindo-se à variável *displayField* o nome desse campo; no Django é necessário utilizar o método *mágico def __str__(self);* ou seja, deve-se definir na chamada *str()* do objeto qual campo deverá representá-lo na visualização.

No CakePHP, o conceito de herança do paradigma de orientação a objetos não possui ligação com o modelo de banco de dados relacional, sendo utilizado apenas em sua forma original do paradigma, ou seja, para herdar métodos e atributos da classe pai. No Django, esse conceito é responsável pela formulação de especializações do modelo relacional. Na aplicação estudada, o reaproveitamento em pessoa física e jurídica de dados comuns à pessoa pode ser feito através de herança de classes no Django, enquanto no CakePHP deve ser interpretado, por padrão, como um relacionamento 1x1.

4.3. CRUD - Create, Retrieve, Update and Delete

Em um sistema Web que faça uso de banco de dados, as principais operações consistem em adicionar, recuperar, editar e remover informações. A utilização dessas quatro operações para a maioria das tabelas faz com que a automatização seja vantajosa para o ganho de velocidade no desenvolvimento. Os *frameworks* estudados agregam formas para a geração de tais funcionalidades de forma automática.

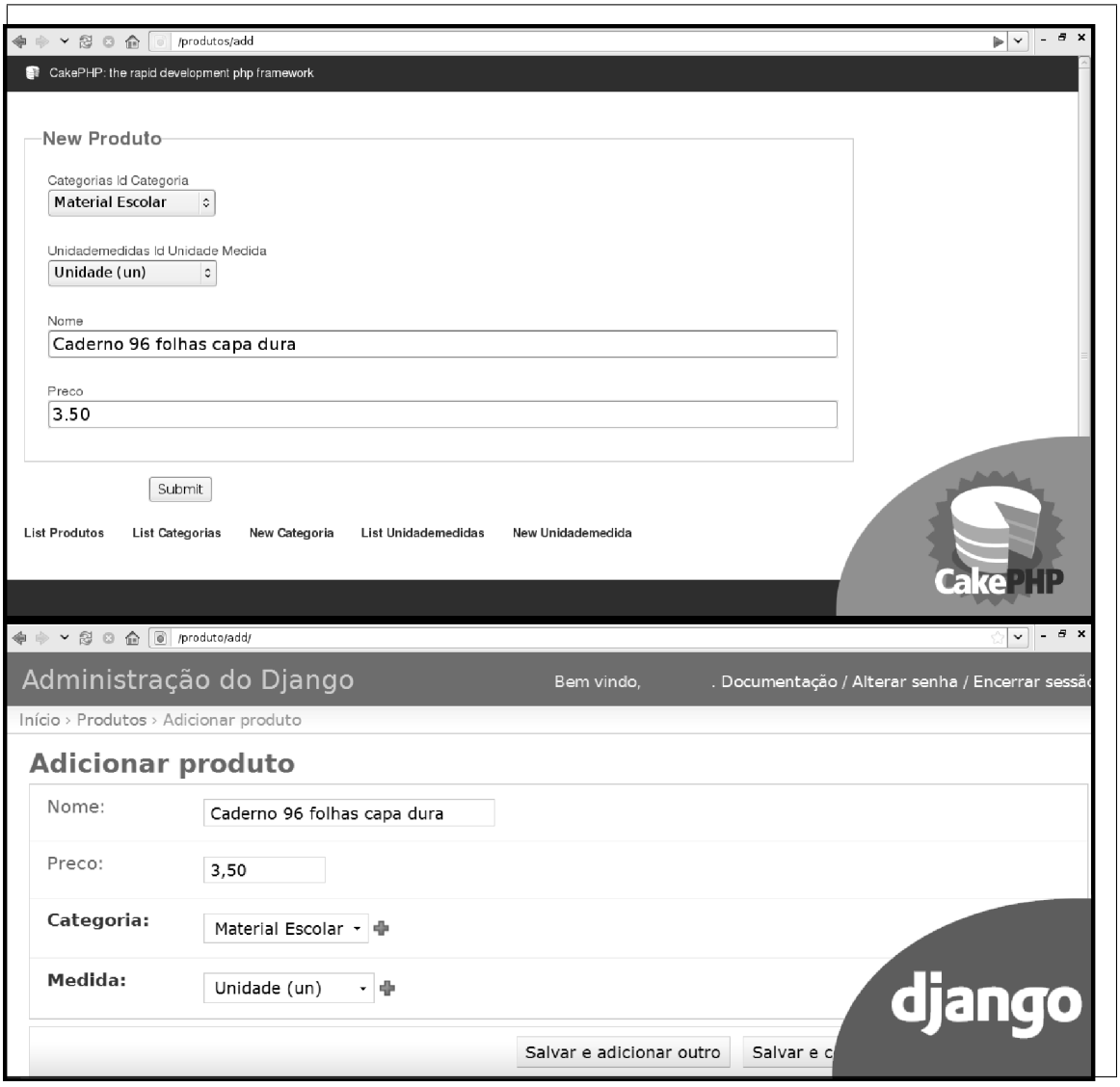

Figura 3. *Telas de inserção de Produto, no CakePHP e no Django.*

Com o CakePHP, utilizando-se o recurso conhecido como *Scaffold*, pode-se gerar formulários de manutenção das tabelas do banco de dados automaticamente. No Django, o CRUD propriamente dito deve ser gerado pelo programador, com uma pequena quantidade de linhas de código e *generic_views*. É possível permitir que o Django assuma a administração das operações do CRUD para os modelos da aplicação, inserindo-se, na classe referente ao *Model* de cada tabela, a especificação: *class Admin: pass*. A implementação do *Scaffold*, no CakePHP, é feita a partir da declaração da variável "*$scaffold*"dentro do *Controller* da tabela, o que torna disponíveis, automaticamente, os métodos de recuperação (exibição), inserção, edição e remoção. Relacionamentos simples entre tabelas, como relações 1x1, 1xN e NxN (sem atributos da relação), são gerados de forma automática em ambos os *frameworks*. Na Figura 3 são apresentados os formulários de inserção de dados, gerados automaticamente para a entidade "Produto"do sistema gerenciador de estoque através dos *frameworks* estudados.

5. Conclusão

Neste artigo foram analisados dois *frameworks* para desenvolvimento rápido de aplicações para a Web, CakePHP e Django. Ambos são distribuídos como Software Livre e foram aplicados a um mesmo caso de desenvolvimento de um sistema gerenciador de estoque de materiais. Observou-se funcionalidades comuns entre os *frameworks* e apresentou-se como cada um as interpreta e implementa, apoiando o desenvolvimento do caso em questão.

Pode-se perceber pontos de vista diferentes adotados pelos *frameworks* nos conceitos envolvidos a diversas funcionalidades. São diferenças que derivam desde a linguagem de programação até as metodologias escolhidas para as implementações. Mesmo com estas diferenças, ambos os *frameworks* mostraram-se efetivos quanto à aceleração no desenvolvimento de aplicações para a Web, pois sem eles seriam utilizadas muitas horas de programação para se chegar a um resultado próximo aos atuais. A escolha por um ou outro *framework* passa por diversos fatores que vão além dos recursos existentes nos servidores onde as aplicações estarão, pois pode depender de muitas variáveis, como até mesmo a capacidade dos recursos humanos envolvidos nos projetos.

Referências

CakePHP (2009). The CakePHP framework. Disponível em: http://cakephp.org/.

Django (2009). The Django framework. Disponível em: http://www.djangoproject.com/.

Fayad, M. E. (2000). Introduction to the computing surveys' electronic symposium on object-oriented application frameworks. *ACM Comput. Surv.*, 32(1):1–9.

Fayad, M. E. and Schmidt, D. C. (1997). Object-oriented application frameworks. *Commun. ACM*, 40(10):32–38.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

Govoni, D. (1999). *Java Application Frameworks*. John Wiley & Sons.

Martin, R. C. (1996). The dependency inversion principle. *The C++ Report*, 8. Disponível em: http://www.objectmentor.com/resources/articles/dip.pdf.