Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

ALDREY ROCHA DUARTE

METODOLOGIA RAILS: ANÁLISE DA ARQUITETURA MODEL VIEW CONTROLLER APLICADA

Belo Horizonte
2011

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Análise de Sistemas

METODOLOGIA RAILS: ANÁLISE DA ARQUITETURA MODEL VIEW CONTROLLER APLICADA

por

ALDREY ROCHA DUARTE

Monografia de Final de Curso

Prof. Ângelo de Moura Guimarães
Orientador

Belo Horizonte
2011

ALDREY ROCHA DUARTE

METODOLOGIA RAILS: ANÁLISE DA ARQUITETURA MODEL VIEW CONTROLLER APLICADA

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção de título de Especialização em Ciência da Computação.

Área de concentração: Análise de Sistemas.

Orientador: Prof. Ângelo de Moura Guimarães.

Belo Horizonte
2011

RESUMO

O objetivo do presente trabalho é propor a aplicação do *pattern* Modelo-Visão-Controlador (MVC) para sanar os problemas de coesão e acoplamento oriundos do desenvolvimento de sistemas *web* sem uma arquitetura bem definida. Após a identificação desses problemas, será estudada a organização de MVC e as razões pelas quais o mesmo pode contribuir para melhorar a eficiência do processo de desenvolvimento, da manutenção e do ganho através do reuso. Como forma de aplicar MVC efetivamente, será apresentado o *framework Ruby on Rails* (Rails), reconhecido não somente pela forma eficiente com que aplica a arquitetura em estudo, mas por dispor de outros recursos destinados à melhoria do processo de desenvolvimento de aplicações *para* uso na internet. Finalmente serão expostos problemas inerentes a um caso de uso prático baseado em um sistema pré-existente constituído sob uma arquitetura pouco consolidada. Nesse caso, o *framework* Rails possibilitará a reconstrução de módulos de um sistema, de forma a avaliar se os benefícios esperados de MVC e outros propostos pelo *framework* serão devidamente alcançados.

Palavras-chave: Arquitetura MVC. Ruby On Rails. *Framework*. Sistemas *web*.

ABSTRACT

The objective of this study is to propose the use of the Model-View-Controller (MVC) pattern to solve problems of cohesion and coupling derived from the development of web systems without a well-defined architecture. After identifying these problems, the organization of the MVC will be studied together with the reasons why it can help to improve the efficiency of the development process, maintenance and gain by reusing. In order to effectively implement MVC, this paper presents the Ruby on Rails (Rails) framework, which is recognized not only by efficiently applying MVC architecture, but by making use of other resources for improving the process of developing web application. Finally will be exposed the problems inherent to a practical use case based on an existing system established under a low consolidated architecture. In this case, the Rails framework will enable the reconstruction of some modules of this system in order to assess whether the expected benefits of MVC and others proposed by the framework will be sufficiently achieved.

Keywords: MVC architecture. Ruby On Rails. Framework. Web systems.

LISTA DE FIGURAS

FIG. 1 Relacionamento entre as camadas da arquitetura MVC 11

FIG. 2 Antiga organização dos módulos do sistema 21

FIG. 3 Estrutura proposta para o módulo de cadastro de usuário 26

FIG. 4 Comparativos entre implementações da ação de “Exibir lista de usuários” no módulo de Cadastro de Usuários 28

LISTA DE SIGLAS

AJAX	Asynchronous Javascript and XML ou *Javascript* e XML Assíncronos
DAO	*Data Access Object* ou Objeto de Acesso a Dados
DOM	*Document Object Model* ou Modelo de Objeto de Documentos
DRY	*Dont Repeat Yourself* ou Não se repita
HTML	*Hypertext Markup Language* ou Linguagem de Marcação de Hipertexto
HTTP	*Hypertext Transfer Protocol* ou Protocolo de Transferência de Hipertexto
MVC	Modelo-Visão-Controlador
ORM	Object-Relational Mapping ou Mapeamento Objeto-Relacional
PDF	Portable Document Format ou Formato de Documentos Portável
RJS	*Rails Javascript ou* Javascript Rails
REST	*Representational State Transfer* ou Transferência de Estado Representacional
XML	*Extensible Markup Language* ou Linguagem de Marcação Extensível

SUMÁRIO

1 INTRODUÇÃO 8
2 REVISÃO DE CONCEITOS 9
2.1 Metodologia Rails 9
2.2 *Pattern* Model-View-Controller 9
2.3 Ruby on Rails 12
2.2.1 A linguagem Ruby 14
2.2.2 Componentes análogos ao MVC em Rails 15
2.2.2.1 Active Record: O Modelo 15
2.2.2.2 Action Pack: A Visão e o Controle 16
2.2.2.2.1 Action Controller 16
2.2.2.2.2 Action View 17
2.2.2.3 Action Dispatch 17
2.2.3 Outros componentes 17
2.2.3.1 Action Mailer 18
2.2.3.2 Active Model 18
2.2.3.3 Active Resource 18
2.2.3.4 Active Support 19
3 DESENVOLVIMENTO 20
3.1 Apresentação do sistema 20
3.2 Problemas do sistema em análise 22
3.3 Criação de uma nova aplicação 23
3.4 Criação de um módulo MVC com Rails 24
4 CONCLUSÕES 30
REFERÊNCIAS 32

1 INTRODUÇÃO

A forma mais básica de se pensar em desenvolvimento de software é a construção de um bloco único que contenha todas as funções necessárias, tanto para interface com o usuário, quanto para manipulação das regras do negócio e o acesso a dados persistentes. Com o tempo observou-se que uma arquitetura desse tipo leva a sérios problemas de manutenção, pois a mesma tende a falhar por não observar dois aspectos importantes relativos aos sistemas: coesão e acoplamento. Uma alternativa é a proposta do Rails, conforme 37signals (2011), uma arquitetura específica, sendo a mesma baseada no *pattern* Model-View-Controller (MVC).

O objetivo dessa pesquisa é expor uma base teórica aplicável relativa à metodologia Rails e os benefícios de sua aplicação na construção de sistemas para a internet. Ainda será analisado o estilo de arquitetura MVC e as características de uma aplicação desenvolvida em camadas bem definidas, o que é um fator determinante na prevenção de falhas em projetos. Nesse contexto acredita-se que a apresentação de Rails como um *framework* que aplica na prática os conceitos de MVC será uma forma de encorajar a utilização, não somente dessa arquitetura, mas de outras que também possam se distinguir positivamente com relação ao desenvolvimento não-estruturado.

No decorrer do trabalho, também serão expostos recursos de Rails que tornam sua arquitetura mais funcional, pois, ainda que esses recursos não sejam derivados diretamente de MVC, foram úteis na constituição de uma tecnologia funcional para desenvolvimento de aplicações web.

2 REVISÃO DE CONCEITOS

Neste capítulo serão apresentados os conceitos obtidos através de pesquisas, com objetivo de expor conhecimentos sobre Rails e como essa tecnologia aplica os conceitos de MVC.

2.1 Metodologia Rails

PRESSMAN (2002) define acoplamento como uma medida qualitativa do grau em que as classes são conectadas entre si, sendo que, à medida que as classes tornam-se interdependentes, o acoplamento cresce. O autor afirma que se deve manter o acoplamento o mais baixo possível, pois, à medida que a comunicação e a colaboração entre as classes aumentam, a complexidade também cresce. Na mesma obra, coesão foi definida como a correlação entre os componentes internos de uma classe. Um módulo coeso tende a desempenhar um único papel bem definido, reduzindo a comunicação com outros módulos do programa.

Voltando à proposta de desenvolvimento de software em um único bloco, percebe-se que devido aos módulos desempenharem diferentes funções (fraca coesão), será necessária intensa troca de informações entre si (forte acoplamento) acrescentando complexidade e levando à propagação de qualquer alteração e, consequentemente, retransmitindo eventuais erros. Diante desse contexto, será posposto o desenvolvimento do site em camadas que agrupem módulos ou classes correlatas, mais precisamente o *pattern* MVC aplicada à metodologia Rails, tendo em vista que o foco principal do presente trabalho são as aplicações para a web.

Como sugerido anteriormente, Rails é constituído de forma a aplicar um estilo de arquitetura específico, sendo o mesmo conhecido como Model-View-Controller (MVC), o qual cria grupos de objetos afins, como descrito a seguir.

2.2 *Pattern* Model-View-Controller

Antes de adentrar na definição de MVC, é importante notar que, algumas vezes o mesmo é referido pelo termo *architecture*, como no caso de SUN (2011), e mais comumente pelo termo *design pattern,* como por GAMMA (1998), que explica que MVC se utiliza de outros *design patterns* para propor a solução do problema. A tradução de *design pattern* encontrada em SOMMERVILLE (2003) é “padrão de projeto”, no entanto, GAMMA (1998) explica que os *patterns* descritos em sua obra não se referem a implementações específicas, mas descrições de como classes e objetos de organizam para resolver um problema comum em projeto, abstraindo o contexto onde e modelo de solução será efetivamente aplicado. Essa última definição de *pattern* condiz com a proposta de MVC, pois o mesmo propõe a solução de uma categoria de problemas, não de uma aplicação específica, sendo assim não pode ser utilizado de forma rigidamente padronizada, mas deve ser interpretado para aplicar-se a um contexto.

A tradução literal do termo *architecture* é arquitetura, porém, dicionário MERRIAM-Webster (2011) possui, dentre as definições do termo *architecture*, uma que se aplica mais intimamente ao contexto da computação, sendo “*the manner in which the components of a computer or computer system are organized and integrated*” cuja tradução é “a forma como os componentes de um sistema computacional são organizados e integrados”. No presente trabalho, os termos arquitetura e padrão serão desencorajados para evitar significados ambíguos, o primeiro por refere-se comumente a forma com que um objeto material se constitui através da organização de suas partes, o segundo por sugerir regras específicas a serem seguidas. Ao invés disso, MVC será referenciado como um *pattern* ou um estilo de arquitetura, pois propõe a solução de uma classe de problemas sem determinar como os componentes de *software* serão efetivamente organizados para cumprir o proposto, ficando a implementação e estruturação de uma arquitetura a cargo da tecnologia que aplicar o *pattern* MVC ou, no caso desse trabalho, o *framework* Rails.

De acordo com GAMMA (1998), MVC consiste em três tipos de objetos que compõem camadas bem definidas: Modelo (*Model*) é o objeto da aplicação, responsável por implementar a lógica da regras de negócio tal qual o armazenamento persistente. Visão (*View*) é camada que compõe a apresentação do sistema para o usuário. RUBY (2009) define o controlador (*Controller*) conduz a

aplicação recebendo eventos do mundo externo, interagindo com o modelo e exibe uma visão apropriada. GAMMA (1998) salienta que MVC desacopla visões e modelos estabelecendo um protocolo de subscrição/notificação entre eles. A visão deve assegurar que sua aparência reflita o estado do modelo, ao passo que, sempre que houver alterações nos dados do modelo, o mesmo notifica a visão que depende desses dados, de forma que essa visão possa se atualizar. Essa abordagem permite que sejam criadas diferentes visões para um mesmo modelo sem necessidade de alterá-lo, sendo esse um importante aspecto ao se tratar de aplicações web que, em sua maioria, são destinadas à utilização em múltiplas plataformas que podem demandar diferentes tratamentos de interface, enquanto preservam o mesmo modelo em um único servidor remoto ou em bases distribuídas.

A figura 1, abaixo, ajuda a esclarecer a configuração das três camadas, com as subscrições representadas por setas grossas e preenchidas e as notificações representadas por linhas tracejadas, se assemelhando à visão de GAMMA (1998), disponibilizada acima:

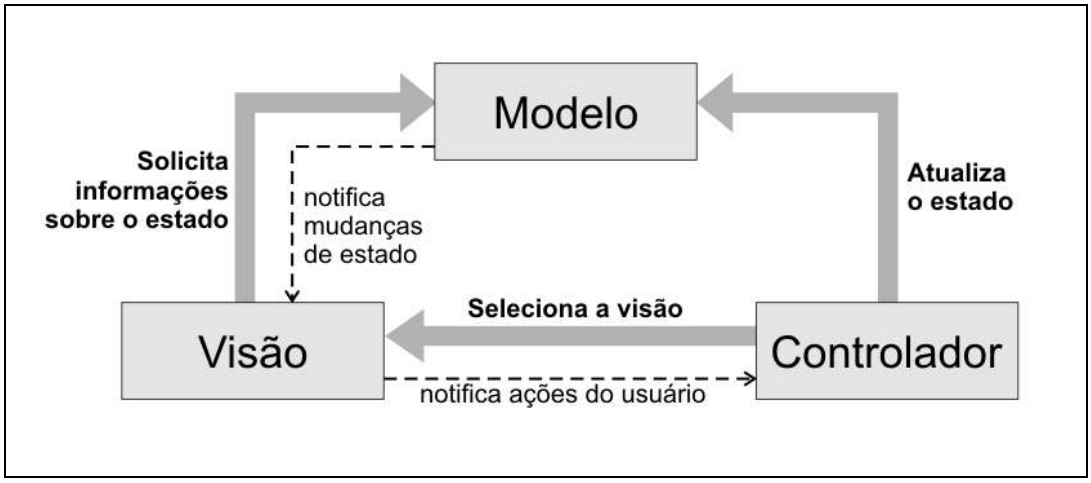


FIG. 1 Relacionamento entre as camadas da arquitetura MVC
Fonte: SUN, 2011

A figura apresentada é uma adaptação e tradução de outra imagem de autoria da SUN (2011), que divide as responsabilidades de cada camada da seguinte forma:

- Modelo: Representa dados empresariais e regras de negócio que controlam a atualização desses dados. Frequentemente o modelo funciona como a aproximação de um processo do mundo real;

- Visão: Acessa o modelo e determina a forma como o mesmo deve ser apresentado, sendo também a camada responsável por manter a consistência entre a apresentação podendo, para o tal, aplicar uma das estratégias seguintes: Cadastrar-se junto ao modelo para receber notificações de mudanças, ou; Solicitar ao modelo o seu estado atual quando for necessário representá-lo;
- Controlador (Controle): Traduz interações com a visão em ações que devem ser desempenhadas encima do modelo. O controlador tem dentre suas funções: Ativar processos de negócio; Mudar o estado do modelo; Escolher a visão apropriada de acordo com as ações dos usuários e com os resultados das mudanças no modelo.

É importante salientar que a visão não é necessariamente uma tela de usuário, mas uma interface de comunicação entre o sistema e seu cliente, que pode ser outro sistema. Por exemplo, um programa acessa um serviço remoto através de uma requisição HTTP (Protocolo de transferência de Hipertexto) onde são enviadas variáveis e recebe respostas nesse mesmo protocolo.

2.3 Ruby on Rails

Ruby on Rails (37signals, 2011) é um *framework* para aplicações web escrito pelo dinamarquês David Heinemeier Hansson, utilizando a linguagem de programação Ruby, conforme HELLSTEN e Laine (2006). Os autores ainda citam que o *framework* lançado ao público em 2004, surgiu da insatisfação de David com relação aos *frameworks* disponíveis na época em que trabalhava no projeto de uma ferramenta de colaboração online denominada Basecamp.

BAKHARIA (2007) descreve MVC como um encaixe perfeito para aplicações Web, e que Rails implementa por completo esse *pattern*. RUBY (2009) salienta que, apesar de desenvolvedores de outras linguagens como, por exemplo, Java estarem acostumados com *frameworks* baseados em MVC, Rails leva o estilo de arquitetura mais longe: quando se desenvolve em Rails, há um lugar certo para cada peça de código, e todas as partes da aplicação se interagem através de um padrão definido. É como se o desenvolvimento já começasse com o esqueleto da aplicação pronto.

Dentre outros, são aspectos importantes de Rails, segundo RUBY et al. (2009):

- Criação automática de stubs de teste;
- Implementação através a linguagem Ruby, totalmente orientada a objeto, humanamente legível e concisa;
- Redução de duplicação de código. *Dont Repeat Yourself* (DRY) ou, "Não repita a si mesmo" é um conceito chave dentro desta arquitetura, tornando-a quase obsessiva em restringir a duplicação;
- Geração automatizada de documentação.

Como citado acima, a implementação mais comum e amplamente aceita utiliza a linguagem Ruby, no entanto, existem autores que defendem Rails como uma arquitetura que não se restringe ao *framework* Ruby on Rails, que será discutido em seguida. DEVRIES e Naberezny (2009) definem Rails de uma forma menos dependente de linguagem, como uma filosofia geral de desenvolvimento focada em criar código manutenível. Os autores afirmam que, seguindo diretrizes simples, o desenvolvedor é capaz de manter um desenvolvimento mais uniforme e com menos riscos de danificar funcionalidades existentes. Essas diretrizes compreendem uma estrutura de diretórios para a aplicação como um todo, além de uma série de convenções para nomenclatura de arquivos, classes e tabelas de banco de dados, o que leva à apresentação de mais um conceito que, junto do DRY, são considerados por RUBY (2009) conceitos chave:

- *Convention over configuration* (Convenção acima de configuração): Significa que Rails possui convenções para praticamente todos os aspectos da orquestração da aplicação. Seguindo as convenções, é possível escrever programas com menos linhas de código e, é possível sobrescrever facilmente as convenções originais caso seja necessário. Pode-se concluir que, seguindo rígidas convenções, é possível reduzir a necessidade de configuração.

ORSINI (2007) anda cita um terceiro princípio que se relaciona intimamente com convenção acima de configuração:

- *Liberal use of code generation* (Uso liberal de geração de código): Rails é capaz de escrever código de forma automática. Ou autor cita como exemplo a definição de uma classe que representa uma tabela no banco de dados, onde Rails é capaz de escrever a classe e a maioria de seus métodos analisando a estrutura da referida tabela. Ainda permite ao desenvolvedor adicionador comportamentos especiais aos métodos criados, ou adicionar métodos novos.

Salvas as opiniões gerais, o próprio criador de Rails é co-autor da obra RUBY et al. (2009), onde o *framework* Ruby on Rails é comumente referenciado apenas como Rails, o que leva à conclusão de que esse *framework* em específico é a realização original do que se conhece por Rails, ainda que existam propostas de utilização das idéias para elaboração de ferramentas em outras linguagens. Sendo assim, o presente trabalho terá por base o *framework* Ruby on Rails e, intrinsecamente, poderá ser feita uma referência ao mesmo através do termo Rails, a menos que seja feita uma ressalva dentro do texto.

Nas seções seguintes serão apresentados importantes aspectos para entendimento de Rails, como a linguagem em que foi escrito, os módulos que o compõem e a sua estrutura de pastas.

2.2.1 A linguagem Ruby

Ruby é uma linguagem de programação orientada a objeto limpa e intuitiva. É uma linguagem interpretada, significando que o código fonte Ruby é compilado e interpretado em tempo de execução (De maneira similar ao JavaScript e ao PHP). Esse comportamento difere de linguagens compiladas como C e C++, onde o código é pré-compilado produzindo um arquivo executável diretamente na linguagem do processador. Linguagens interpretadas têm como desvantagem: A necessidade de compilar novamente o código a cada vez que o programa for acessado; A abertura do código-fonte para a máquina que necessitar executar o programa. As vantagens mais notáveis de linguagens interpretadas são: Portabilidade em múltiplas plataformas; Possibilidade de editar o código e executar o programa em tempo real. (TECHOTOPIA, 2011)

HELLSTEN e Laine (2007) acrescentam que a orientação a objeto em Ruby é completa, ou seja, toda variável faz referência a um objeto, não existindo tipos primitivos. Ruby ainda é altamente dinâmica, possibilitando alterar classes e inserir novos métodos em tempo de execução. Isso expande as possibilidade que comumente encontradas em outras linguagens.

Outro fator determinante da utilização de Rubyno *framework* Rails é exposto por HELLSTEN e Laine (2007): Ruby foi concebido com a intenção de aumentar a satisfação do programador, permitindo que ele se concentre mais na solução do problema que na sintaxe da linguagem. Essa característica funde perfeitamente com o fato de Rails ser altamente voltado a satisfação e produtividade.

2.2.2 Componentes análogos ao MVC em Rails

Nessa seção serão apresentados os módulos de Rails que correspondem ao Modelo, Visão e Controlador de MVC.

2.2.2.1 Active Record: O Modelo

Para compreender o significado desse componente, é necessário apresentar o conceito de *Object-Relational Mapping* (ORM), que pode ser traduzido como Mapeamento Objeto Relacional e, segundo TECTARGET (2011), trata-se de um mecanismo que possibilita endereçar, acessar e manipular objetos, sem que seja necessário preocupar-se com a forma como esses objetos se relacionam com suas fontes de dados. ORM permite aos programadores manter a uma visão consistente dos objetos no decorrer do projeto, mesmo que haja mudanças nas fontes, nos canais de comunicação e nas aplicações que utilizam os objetos de dados.

De acordo com BAKHARIA (2007), o componente Active Record segue os conceitos de ORM. A autora salienta que Active Record cria uma abstração orientada a objeto do banco de dados, onde tabelas são mapeadas em classes, enquanto os campos das tabelas são referenciados como propriedades dessas classes. Active Record também fornece métodos nas classes para permitir operações sobre os dados, como, por exemplo, salvar e localizar. Ao contrário de outras bibliotecas que

implementam ORM, Active Record não requer configuração complexa, além de ser capaz de propor mapeamentos com base em convenções de nomenclatura de tabelas e campos, o que ajuda a justificar a importância de um dos conceito chave, que é a convenção preferível à configuração. A mesma ainda afirma que o Active Record torna Rails o *framework* mais produtivo para sites orientados a bancos de dados.

Na prática, todas as classes criadas estendendo-se a classe nativa ActiveRecord::Base serão abstrações de uma entidade contida no banco de dados, assim, os objetos dessas classes correspondem a linhas das tabelas, como explica RUBY (2009). Como define ORSINI (2007), as classes Active Record são modelos de domínio, sendo constituídas de dados e uma série de regras que determinam sua interação com a aplicação.

2.2.2.2 Action Pack: A Visão e o Controle

BAKHARIA (2007) explica que, em Rails, as camadas visão e controle do MVC são tão ligadas que foram colocadas juntas em um componente denominado Action Pack. Mas, RUBY (2009) justifica que, ao contrário do que possa parecer, Rails fornece clara demarcação de código para controle e apresentação, sendo que, Action Pack é constituído por dois módulos: Action Controller, análogo ao controle do *pattern* MVC e; Action View, análogo à visão.

2.2.2.2.1 Action Controller

Após o roteamento determinar qual controle deverá responder a uma requisição, o mesmo deverá produzir os serviços requisitados, como explica 37signals (2011). Segundo ORSINI (2007), em uma aplicação Rails, um controle é uma especialização da classe base ActionController. Como explanado anteriormente, essas classes definem as regras de negócio da aplicação. Na verdade, em um aplicativo web deve existir um controlador que gerencia a lista de serviços e permite a localização dos mesmos e, outro controlador que dedicado à administração do site. Controles geralmente correspondem ao modelo em que operam, mas essa não é uma regra.

Os serviços disponíveis por um controle Rails correspondem aos métodos públicos de sua classe. Quando desempenha uma ação, o método pode acessar ou comandar a atualização de informações do modelo, ou fazer chamado a outros métodos. Ao concluir a atividade, o modelo normalmente atualiza a visão com o mesmo nome da atividade, no entanto, ele pode comandar a criação de outra visão. (ORSINI, 2007)

2.2.2.2.2Action View

Segundo BAKHARIA (2007), Action View é o componente responsável por produzir a visão associada a uma ação do controlador. Um controlador pode ter múltiplas ações (métodos) as quais estão automaticamente designadas a produzir uma visão com o mesmo nome. Uma visão também é conhecida como *template* ou arquivo de *template,* que pode ser entendido como um modelo ou gabarito. A visão pode produzir não somente arquivos HTML, mas inúmeros outros tipos de arquivo utilizados para visualização direta do usuário, como PDF, por exemplo, ou para interface entre sistemas web, como XML e RJS.

RUBY (2009) explica que o *template* é um arquivo que deve ser interpretado e expandido para produzir o seu resultado final. *Templates* contêm uma mistura de texto fixo e código. O código é escrito utilizando a liguagem Ruby e executado em um ambiente que o permite acessar diversas informações disponíveis no controlador.

2.2.2.3 Action Dispatch

De acordo com 37sinals (2011), Action dispatch é um componente responsável por interpretar rotas utilizadas pelos outros módulos rails. A existência do Action Dispatch permite que seja utilizada uma sintaxe de endereçamento mais simples e abrangente. Esse componente não tem um representante direto em MVC, mas contribui para a comunicação entre os componentes citados, além de fazer parte do Action Pack.

2.2.3 Outros componentes

Os componentes seguintes também integram o *framework* Rails e fornecem funcionalidades essenciais ou de suporte à aplicação.

2.2.3.1 Action Mailer

Action Mailer é um componente simples do Rails que permite à aplicação enviar e receber e-mails, como define RUBY (2009). Para BAKHARIA (2007), é importante um sistema que facilite a operação de envio de mensagens de e-mails porque essa é uma operação tem diversas aplicações comuns como: Enviar e-mails de confirmação de cadastro; Notificações de erros ao administrador do site; Confirmação de compra de produtos em lojas virtuais; Newsletters.

A classe Mailer de Rails possui métodos construtores para diferentes mensagens que a aplicação necessite. O formato da mensagem é determinado por um Action View, de forma similar a *templates* RHTML. Cada construtor tem um Action View correspondente que determina o conteúdo da mensagem de e-mail. (ORSINI, 2007)

2.2.3.2 Active Model

Conforme exposto por 37sinals (2011), Action Model fornece uma interface definida entre os serviços do Action Pack e módulos de abstração de bancos de dados (ORM) como o Active Record. Dessa forma, Rails permite que a aplicação utilize outros *frameworks* de ORM ao invés de se restringir ao uso do Active Record nativo do Rails.

2.2.3.3 Active Resource

Um importante conceito utilizado pelo Active Resource é o de aplicações RESTful. Transferência de Estado Representacional (Representational State Transfer - REST) é um estilo arquitetural para sistemas distribuídos de hipermídia elaborado por Roy Thomas FIELDING em sua dissertação de mestrado, FIELDING (2000), e definido pelo mesmo como um estilo hibrido derivado de diversos estilos de arquitetura

baseados em rede, combinados com novas regras que definem uma interface de conexão uniforme.

RICHARDSON (2007) define RESTful como sendo uma aplicação orientada a recursos disponibilizados remotamente (*web services*) e acessíveis através de requisições em um padrão bem definido, REST, onde cada requisição HTTP contendo informações que identificam a ação (método), tal qual os parâmetros necessário para que o mesmo execute essa ação. Se uma requisição não remete a um método, não poderá ser considerada RESTful. RUBY (2009) salienta que Rails promove a comunicação de *web services* RESTful.

Active Resource provê um *framework* para gerenciar a conexão entre objetos da aplicação e *web services* RESTful. (37signals, 2011)

2.2.3.4 Active Support

Active Support é uma série de bibliotecas compartilhadas por todos os componentes Rails. Muito do que está contido nesse módulo está destinado a uso interno do Rails. No entanto, Active Support também estende algumas das classes internas de Ruby de maneira útil e interessante. (RUBY, 2009)

3 DESENVOLVIMENTO

Neste capítulo, Rails será apresentado com uma proposta para reconstrução de alguns módulos de um sistema pré-existente, que inicialmente havia sido produzido sem uma arquitetura bem definida e. A intenção não é recriar o sistema utilizando Rails, mas descrever como o framework poderia ser aplicado como uma alternativa viável para resolver impasses encontrados no projeto, tal qual favorecer a manutenção do sistema.

O tópico 3.1, é uma breve descrição do sistema em que o trabalho manterá foco, e o tópico 3.2 identifica os principais problemas encontrados nesse sistema e aos quais estão sujeitos os sistemas construídos sem uma arquitetura consolidada. Cada subtítulo seguinte descreve um recurso de Rails que foi identificado como notável na solução do problema de modularidade do sistema, tanto através de pesquisas quanto em testes práticos. Apesar do sistema não ser refeito por completo, serão desenvolvidos alguns módulos com intuito de consolidar o conhecimento apresentado sobre MVC e Rails em sua versão 3.1.1.

3.1 Apresentação do sistema

O sistema em questão é um gerenciador de ramais, destinado a controlar a atribuição de ramais lógicos aos equipamentos físicos que possibilitam sua existência. O recurso físico que corresponde a um ramal existe na forma de uma porta que pertence a uma placa eletrônica digital. Usuários do grupo “Gestor de Ramais” são responsáveis pela tarefa de cadastrar ramais, cadastrar recursos de máquina, determinar a situação de funcionamento desses equipamentos e vinculá-los aos respectivos ramais.

Outra atribuição importante do sistema é a pesquisa de ramais pelos colaboradores da instituição e, para tornar esse recurso possível, o sistema permite que sejam cadastrados os setores, subsetores e usuários, tal como atribuir os ramais aos setores e subsetores, possibilitando a localização de ramais através da busca, ou

pelo nome do setor/subsetor ou pelo nome do funcionário. Essa parte do cadastro é uma atribuição de usuários do grupo denominado “Secretários”.

Existe um último grupo de usuários chamado “Gestores de Usuários”, responsável pelo cadastro de usuários do sistema e seus respectivos grupos de acesso.

Para desenvolvimento desse sistema, foi elaborado um planejamento com entrevistas com o usuário e a análise formal, incluindo elaboração de diagramas que conduziram o projeto. No entanto, o problema central e tema de discussão desse trabalho é a aplicação de uma arquitetura informal derivada apenas da experiência dos desenvolvedores e que não corresponde a nenhuma arquitetura consolidada através de sucessiva utilização, como é o caso da MVC.

A figura 2, abaixo, descreve a organização com a qual cada módulo do sistema foi desenvolvido. É importante observar que que os itens destacados em azul correspodem a páginas de código específicas de cada módulo, enquanto os itens em verde são comuns a todos os módulos do sistema.

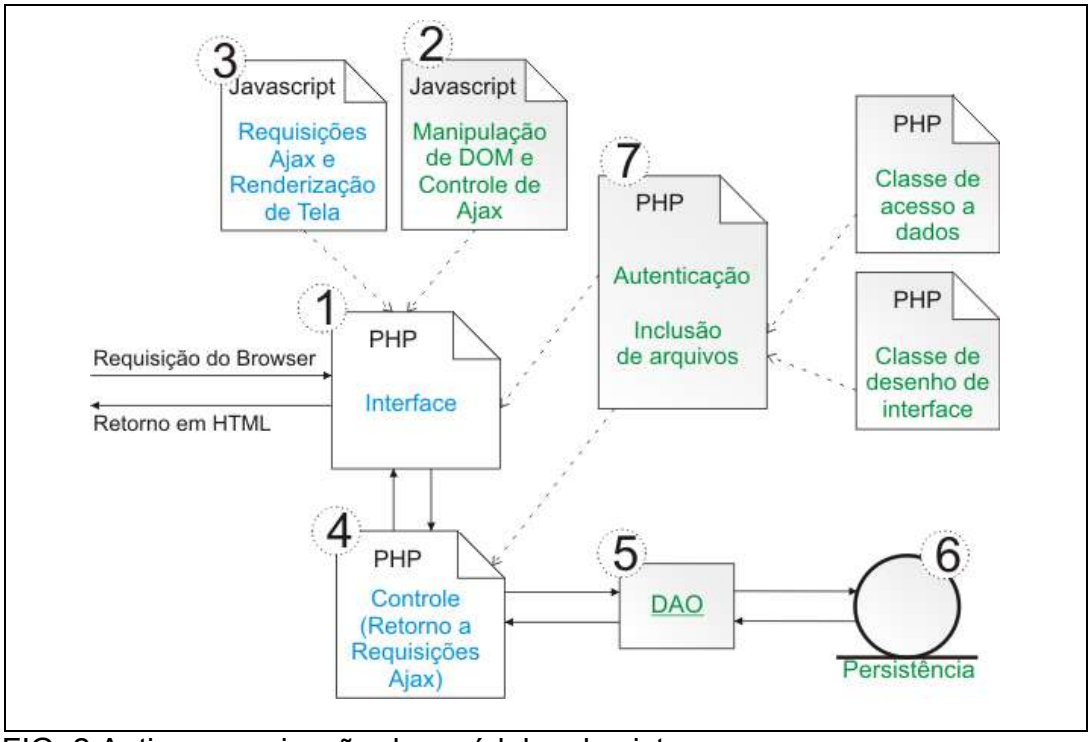


FIG. 2 Antiga organização dos módulos do sistema

FONTE: PRÓPRIO AUTOR.

Através da figura 2 é possível identificar que o sistema foi escrito na linguagem PHP, sendo apoiado por páginas de programação Javascript de forma a acrescentar o dinamismo da metodologia AJAX. Cada módulo é composto por:

1. Arquivo PHP que recebe as requisições do navegador (browser) do usuário. Esse arquivo te a função de agregar os recursos necessários ao desenho da tela do usuário e produzir uma instância da classe de desenho de interface, comandado o desenho da tela com o conteúdo central descrito em si;
2. Arquivo javascript contendo as funções comuns utilizadas para manipulação de DOM e produção de requisições AJAX;
3. Arquivo javascript específico da página. Utiliza-se das funções do javascript comum para compor as transformações de layout específicas da tela em questão. Responsável por enviar as requisições do usuário ao arquivo descrito no item 4 e tratar os retornos fornecidos;
4. Arquivo PHP que contém todas as funções que tratam de requisições dos usuários, comandam transformações nos dados armazenados no banco de dados e fornecem informações à página de interface (item 1) que, por sua vez, depende dos comando de seu javascript específico (item 3) para proceder com o feedback para o usuário ;
5. Implementação em PHP de um *Data Access Object* básico que permite a abstração parcial do acesso ao banco de dados, se responsabilizando pela conexão e disponibiliza comandos padronizados para as operações de bancos de dados;
6. Banco de dados MySQL;
7. Arquivo que contém as funções básicas para autenticação de usuários. Também é responsável por comandar a importação de arquivos comuns a todos os módulos do sistema.

Para a composição das interfaces do usuário ainda foram utilizadas imagens e folhas de estilo CSS, sendo que o gerenciamento desses itens cabia à página descrita no item 7, caso fossem de uso comum, ou do item 1, caso fossem específicos de um módulo.

3.2 Problemas do sistema em análise

Não favorece o reuso: Nota-se que essa organização é um padrão específico do projeto, o que o torna difícil de interpretar para novos colaboradores que venham a trabalhar no mesmo projeto. Também é importante considerar que essa arquitetura tomou tempo e trabalho para ser projetada.

Prejudica a integração e escalabilidade: O sistema não se constitui em uma interface comum para comunicação com outros sistemas nem para inclusão de módulos disponíveis no mercado.

Propaga atualizações: Cada módulo é composto por diversos arquivos, no entanto esses arquivos não constituem módulos suficientemente coesos, o que leva as alterações no controle PHP, por exemplo, a demandarem alterações no javascript específico da página ou até mesmo na página.

Dificulta a comunicação entre módulos: Sempre que for necessário que um módulo do sistema forneça informações a outro será necessário confiar em sessões ou na disponibilização de variáveis globais no arquivo PHP comum a todas. Em ambos os casos é necessário dedicar especial atenção à segurança da informações compartilhadas e à sobrecarga da memória e do *namespace* com informações que não são utilizadas pela maioria dos módulos.

3.3 Criação de uma nova aplicação

A instalação do servidor que interpreta a linguagem Ruby, tal qual a instalação do framework Rails não serão discutidos, visto eu não apresentam distinção notável com relação à instalação de outros produtos semelhantes destinados ao desenvolvimento de aplicações web. Detalhes desse processo podem ser encontrados no tutorial de HARTL (2011).

Com o servidor e os módulos de Rails instalados, a criação da estrutura de uma nova aplicação se mostrou um processo bastante simples, onde, apenas um comando de uma linha de código gerou a estrutura de pastas, alguns arquivos de configuração, outros contendo referências de código. Essa estrutura já define o local correto para cada peça de código, o que poupa o trabalho de definir uma estrutura

própria, além de oferecer uma forma de trabalho padrão entre projetos, resulta de pesquisas e experiências da equipe desenvolvedora do framework Rails. Outro benefício notável é a eliminação de um processo manual que poderia conduzir a erros mais facilmente.

Além disso, durante o processo de desenvolvimento foi necessário incluir novos módulos na aplicação. Um exemplo é o módulo de banco de dados sqlite3 que precisou ser atualizado para eliminar conflito de versões. O processo de inclusão de módulos se mostrou igualmente simples, sendo que, a cada atualização no sistema, o programa de instalação de encarrega de mover módulos obsoletos para um destino próprio. Um arquivo denominado GemFile define os módulos presentes na aplicação, de forma que, se for necessário executá-la em um servidor diferente, basta executar o comando de instalação de módulos novamente.

3.4 Criação de um módulo MVC com Rails

Para testar o funcionamento de Rails, foi recriado o módulo para cadastro de usuários no sistema de telefonia. Sendo assim, verificou-se que existem, pelo menos, três maneiras de criar um novo módulo dividido nas camadas MVC:

- Utilizar console Rails para executar o comando de criação do controlador e de suas respectivas visões e utilizar o comando de criação de modelos para cada entidade que deva ser armazenada de forma persistente. Essa é a forma ideal para criar um novo módulo, pois cria um esqueleto do módulo dentro das convenções de diretório e nomenclatura do Rails eliminando erros que existiriam caso o processo fosse feito manualmente;
- Utilizar o comando *scaffold*: Esse comando cria um modelo de acordo com os parâmetros especificados e cria automaticamente um controlador para esse modelo contendo as operações de inclusão, edição, visualização e remoção. Também são criadas as visões que respondem a cada uma das operações básicas descritas. O processo citado é conhecido também como *scaffolding* e se mostrou útil por: Produzir um modelo que serve de referência para a compreensão do funcionamento de Rails; Substituir módulos que ainda não foram desenvolvidos durante os testes nas fases iniciais do projeto. É importante considerar que os módulos desenvolvidos dessa forma não

incluem itens como validação e segurança, logo carecem de alterações caso haja pretensão de utilizá-los para razões fora das citadas;

- Criar os arquivos manualmente: Dessa forma também será obtida total flexibilidade, mas exigirá mais trabalho do desenvolvedor, assim como serão maiores as chances de erros no processo.

Os dois primeiros métodos foram testados com sucesso. Nesse caso, especificamente, notou-se que o processo de *scaffolding* poderia ser útil, pois a inclusão de um usuário é semelhante a um *scaffold* básico.

O desenvolvimento se mostrou bastante eficiente devido a Rails possuir uma estrutura muito bem definida para divisão em camadas, somando-se aos comandos capazes de automatizar a criação dos módulos, constituem uma forma de desenvolvimento eficiente, menos vulnerável a erros, coesa e facilmente integrável.

Outro fator importante é que Rails permitiu abstração total entre as entidades do modelo e a forma como são armazenados fisicamente, bastando que fosse executado um simples comando de migração para que as alterações no modelo refletissem no banco de dados. Esse processo se mostrou útil para propagação de alterações entre as camadas do sistema sem intervenção do desenvolvedor.

Figura 3, abaixo, descreve como o módulo de cadastro de usuários foi constituído dentro da estrutura de pastas da aplicação Rails. É possível notar que o código da aplicação é convenientemente disposto no diretório app, enquanto os outros diretórios são utilizados pelas configurações do Rails de forma transparente.

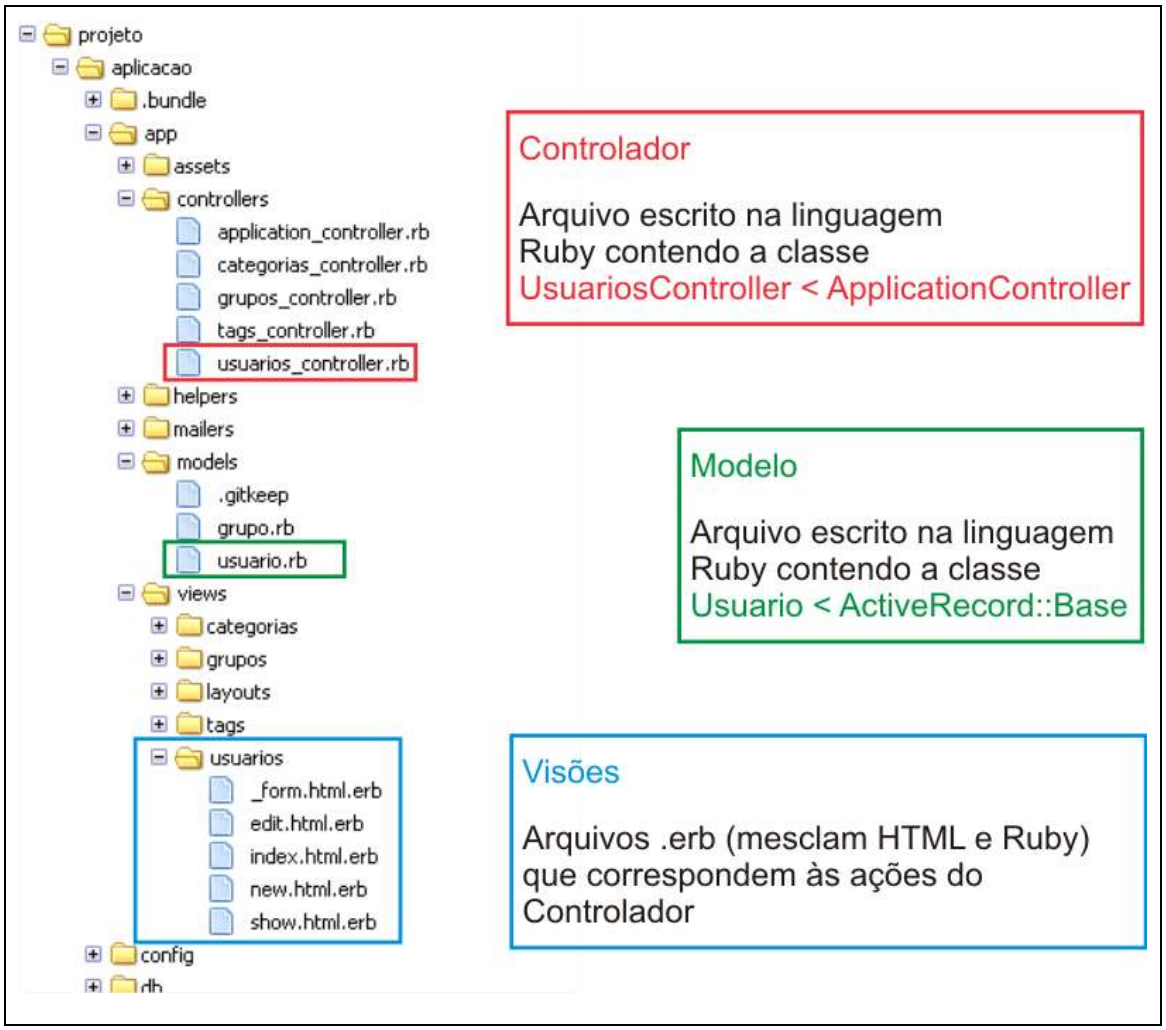


FIG. 3- Estrutura proposta para o módulo de cadastro de usuário

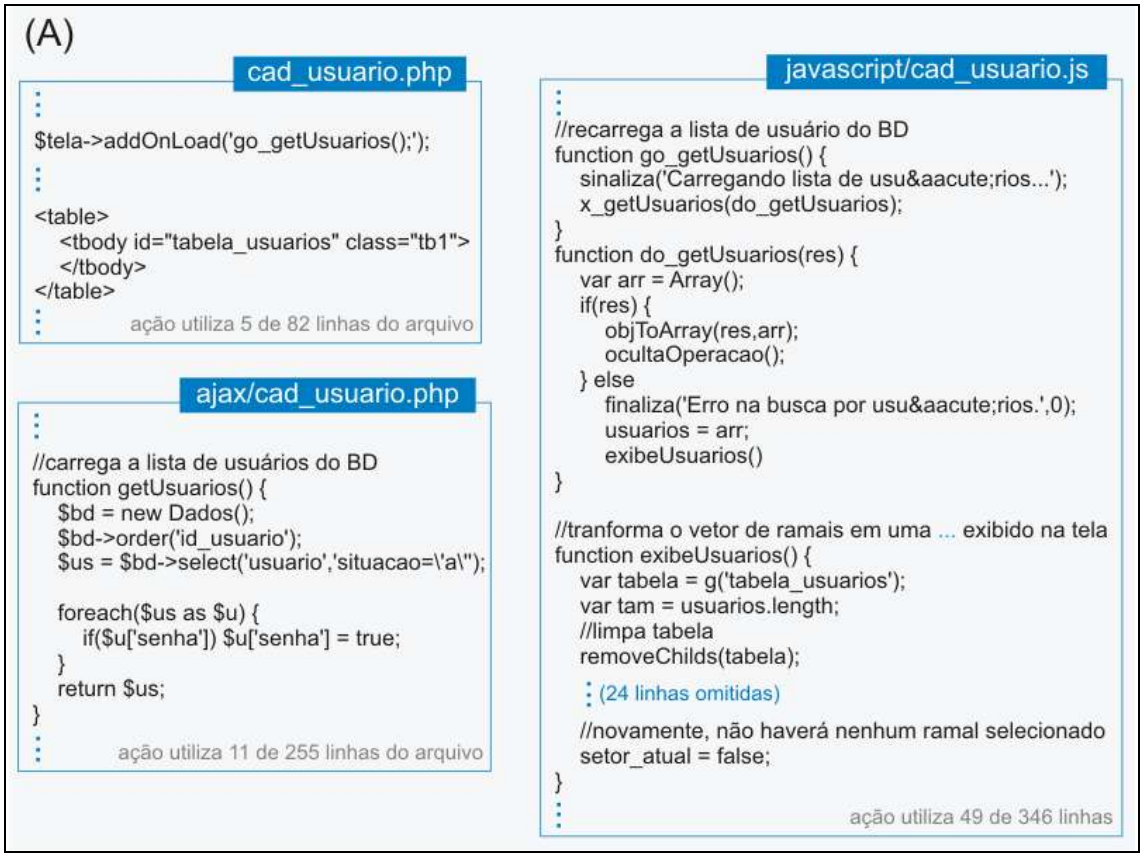
FONTE: PRÓPRIO AUTOR.

Na figura, acima, os arquivos que correspondem às camadas MVC no módulo de cadastro de usuários podem ser identificados com maior clareza. Os modelos e controladores são mantidos em arquivos próprios, enquanto as visões de um determinado controlador ficam armazenadas em um diretório de mesmo nome. Ainda é importante verificar que os controladores e modelos são especializações de classes Rails construídas especialmente para atender os requisitos de comunicação previstos na arquitetura MVC, o que confere maior confiabilidade que, por exemplo, as classes desenvolvidas no sistema original para desenho de tela e para abstração de bancos de dados.

Com relação à estrutura, vale salientar que Rails conduz ao desenvolvimento de aplicações que efetivamente aplicam a arquitetura MVC abstraindo os detalhes que

possibilitam esse feito, o que facilita em muito o processo, podendo ser utilizado inclusive como forma de educar a equipe de desenvolvimento, ao invés de depender passivamente dos esforços da mesma. No entanto, ainda é necessário que os desenvolvedores possuam conhecimento em MVC, ou a ferramenta pode ser subutilizada. Também é importante avaliar o custo do aprendizado de Rails, pois isso envolve a assimilação de muitos aspectos próprios do framework, muito embora o mesmo aplique em si diversos conceitos e tecnologias comuns a aplicações web.

Como forma de comparar a implementação original do sistema e sua contrapartida reformulada utilizando Rails, foi elaborada a figura 4, logo abaixo, que expõe os trechos de códigos envolvidos na ação de exibir a lista de usuários para o mesmo módulo da figura 3. Note que a figura 4 se divide em duas áreas: (A) representando a ação no sistema original PHP/Javascript e; (B) representando a mesma ação reconstruída utilizando Rails.



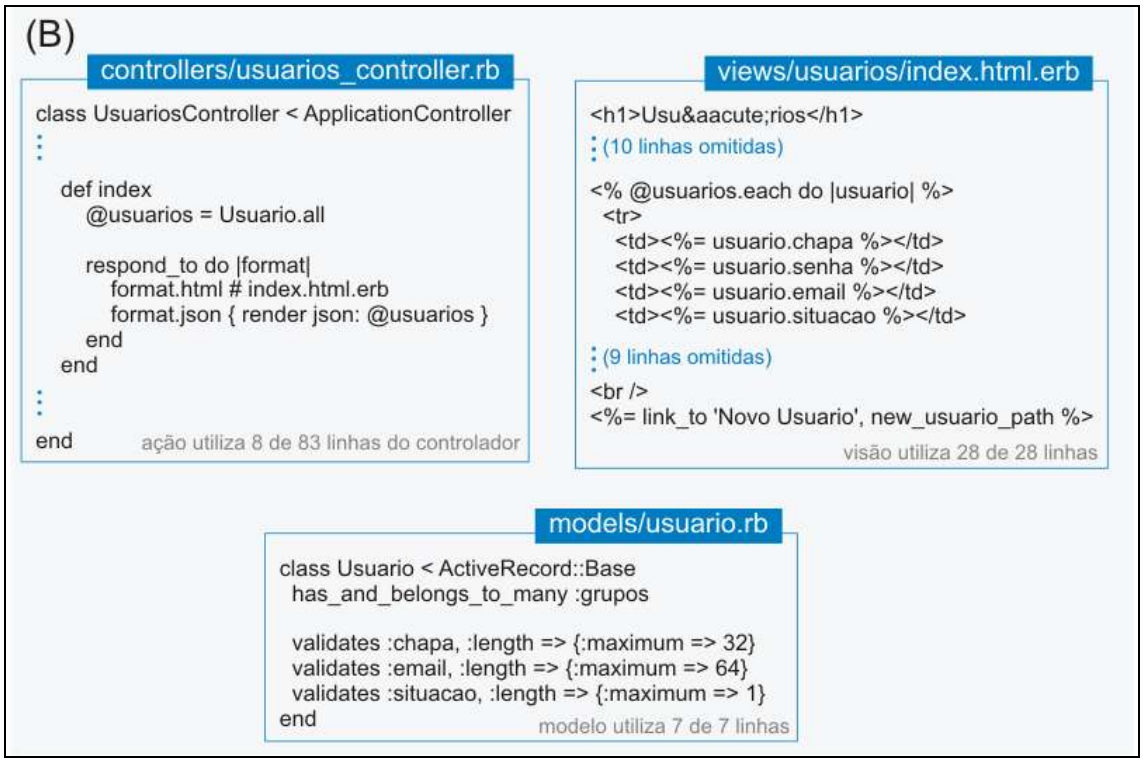


FIG. 4 Comparativos entre implementações da ação de “Exibir lista de usuários” no módulo de Cadastro de Usuários
FONTE: PRÓPRIO AUTOR.

Na figura 4, verificam-se os trechos de código agrupados em caixas que representam os arquivos em que estão contidos, considerando apenas os arquivos específicos desse módulo, ou seja, omitido as funcionalidades básicas de cada sistema e focando no que é efetivamente produzido para se criar um novo módulo. Os trechos tracejados em azul são abreviações para possibilitar a visualização e o entendimento dos detalhes relevantes. Detalhando o código (A) é possível chegar às seguintes constatações:

- O arquivo cad_usuario.php contém a codificação PHP que descreve de forma inicial o layout e as scripts que serão utilizados, sendo assim, é necessário que o mesmo efetue chamadas para ações do arquivo cad_usuário.js, como acontece com a chamada de go_getUsuarios(), programada para ser executada no carregamento da página;
- O arquivo cad_usuário.js trabalha em conjunto com o arquivo cad_usuario.php para atualizar o layout dinamicamente em resposta a eventos da página ou a ações do usuário. Essa configuração revela o forte

acoplamento existente entres esses dois arquivos, já que trabalham juntos para produzir a visão do sistema;

- O arquivo Javascript também é responsável por enviar os dados usuários para processamento no arquivo AJAX/cad_usuario.php, que é capaz de acessar o banco de dados e fornecer o resultado em JSON para o Javascript. No exemplo, o acionamento da função javascript x_getUsuarios() na verdade utiliza um framework AJAX para acionar a função getUsuario() no AJAX/cad_usuario.php. Esse processo identifica esse último arquivo como uma analogia ao controlador e ao modelo de MVC, deixando clara a fraca coesão do módulo;

Detalhando o código de (B):

- A ação de exibir usuários corresponde à ação *index* (principal) do controlador, que é capaz de acessar as informações contidas no modelo e acionar a visão correspondente à ação;
- Cada arquivo isola com clareza sua função, o que representa forte coesão e os insere em uma camada isolada na arquitetura MVC;
- Existe uma clara convenção de nomenclatura original do Rails, sendo que o mesmo é capaz de inferir sobre detalhes que foram omitidos como, por exemplo, a chave primária da entidade Usuário, o que produz um código. Isso remete ao conceito de Rails: Convenção acima de configuração;
- O modelo Usuario contém informações que podem ser utilizadas para abstrair sua implementação no banco de dados, sendo utilizado uniformemente pelo controlador e pela visão, revelando o fraco acoplamento;

Além das diferenças exemplificadas acima, verifica-se que na concepção de (A) foi necessário criar manualmente, em cada arquivo, um trecho correspondente a cada ação, de forma que será necessário alterar todos os arquivos se houver uma alteração em uma ação. As alterações terão que ser propagadas para o banco de dados também manualmente caso modifiquem algo inerente ao modelo, representando novamente um problema com o forte acoplamento.

4 CONCLUSÃO

Primeiramente, acredita-se que a teoria apresentada sobre o *framework* Rails foi satisfatória no entendimento do mesmo. Nesse contexto, a análise da arquitetura MVC contribuiu para o entendimento desse *framework* e auxiliou na compreensão de como a complexidade dos sistemas pode conduzir a falhas e dificuldades de manutenção e como esses problemas podem ser solucionados com a aplicação de uma arquitetura consolidada como MVC. A apresentação das camadas com que se divide um sistema MVC foi essencial para o entendimento de Rails, pois o mesmo se constitui nas mesmas camadas, mas Rails também contribuiu para a desmistificação da arquitetura e ainda contribuiu para atingir o objetivo de expor os benefícios de Rails e MVC especialmente para sistemas web, já que se constituem como uma forma de tratar especificamente de sistemas remotos.

Um ponto negativo do trabalho é a pouca profundidade com que os aspectos de Rails foram tratados, já que o mesmo engloba inúmeras tecnologias que não se relacionam diretamente com MVC e fogem ao foco do trabalho, porém, até mesmo alguns recursos que contribuem diretamente para a arquitetura não foram suficientemente cobertos, como, por exemplo: O *Active Record*; A orientação a objeto fornecida por Ruby; O endereçamento *RESTfull*. Devido isso, certamente será necessário que os leitores desse trabalho que necessitem utilizar Rails tenham que se dedicar ao aprendizado de vários aspectos de forma a utilizar o *framework* em todo o seu potencial.

Na visão do autor como especialista em analise de sistemas, o trabalho foi relevante por vincular a teoria de MVC à sua implementação em Rails e, reforçando-se pela análise de aspectos práticos, reduziu ainda mais a distância entre teoria e funcionamento real. Como a migração de um paradigma de arquitetura envolve esforço inicial e a fuga da zona de conforto, o trabalho pôde contribuir por permitir ao analista enxergar os benefícios dessa mudança. Dessa forma, atingiu-se o objetivo de encorajar o uso, não apenas de Rails e MVC que se encaixam bem para softwares web, mas de outras arquiteturas e tecnologia que se apliquem

adequadamente aos diferentes tipos de sistema. Contudo, esse último objetivo não foi afetado pela falta de profundidade do trabalho.

O conhecimento adquirido poderá se desdobrar futuramente em uma nova pesquisa que apresente mais aspectos funcionais de Rails, focando em um número limitado de aspectos, como forma de atingir o aprofundamento necessário para aplicação prática. Ainda seria relevante o estudo de outras arquiteturas como quatro camadas, e como as mesmas podem ser aplicadas como alternativas ao desenvolvimento de sistemas *web*. Outra possível continuidade seria o estudo de outros *frameworks* destinados a propósitos semelhantes ao Rails, traçando um comparativo.

REFERÊNCIAS

37SIGNALS. *Ruby on Rails: Getting Started with Rails*. Disponível em: <http://guides.rubyonrails.org/getting_started.html>. Acesso em: 02 set 2011.

37SIGNALS. *Ruby on Rails Documentation*. Disponível em: <http://api.rubyonrails .org>. Acesso em: 03 set 2011.

BAKHARIA, Aneesha. *Design Ruby on Rails Power: The Comprehensive Guide*. Boston, MA: Thomson Course Technology, 2007.

DEVRIES, Derek; NABEREZNY, Mike. *Rails for PHP Developers*. Raleigh: The Pragmatic Bookshelf, 2009.

FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Disponível em: <http://www.ics.uci.edu/~fielding/pubs/ dissertation/top.htm >. Acesso em : Acesso em 26 set 2011.

GAMMA, Erich et al. *Design Paterns: Elements of Reusable Object-Oriented Software*. Reading: Addison-Wesley, 1998.

HARTL, Michael. *Ruby On Rails Tutorial*. Disponível em <http://ruby.railstutorial.org /ruby-on-rails-tutorial-book>. Acesso em 10 nov 2011.

HELLSTEN, Christhian; LAINE Jarkko. *Beginning Ruby on Rails E-Commerce: From Novice to Professional*. Apress, 2006.

MERRIAM-WEBSTER. *Architecture*. Disponível em: <http://www.merriam-webster.com/dictionary/architecture>. Acesso em: 06 dez 2011.

ORSINI, Rob. *Rails Cookbook*. Sebastopol, CA: O'Reilly Media, 2007.

PRESSMAN, Roger S. *Engenharia de Software*. Rio de Janeiro: MacGraw Hill, 2002.

RICHARDSON, Leonard. *RESTful Web Services*. Sebastopol, CA: O'Reilly Media, 2007.

RUBY, Sam; THOMAS, Dave; HANSSON, David. *Agile Web Development With Rails*. 3 ed. Raleigh: The Pragmatic Bookshelf, 2009.

SOMMERVILLE, Ian. *Engenharia de Software*. 6 ed. São Paulo: Addison Wesley, 2006

SUN MICROSYSTEMS. *Java BluePrints - J2EE Patterns - JAVA Sun*. Disponível em: <http://java.sun.com/blueprints/patterns/mvc-detailed.html>. Acesso em: 24 mai 2011.

TECHOTOPIA. *What is Ruby?*. Disponível em: <http://www.techopia.com/ index.php/What_is_Ruby%3F>. Acesso em: 03 out 2011.

TECTARGET. *object-relational mapping (ORM)*. Disponível em: <http://search windevelopment.techtarget.com/definition/object-relational-mapping>. Acesso em: 03 out 2011.