

# Partial View: Um padrão para reutilização de *views* na arquitetura MVC

RICARDO VIANA, Universidade Federal do Ceará  
THALISSON OLIVEIRA, Universidade Federal do Ceará  
FERNANDO TRINTA, Universidade Federal do Ceará  
ROSSANA ANDRADE, Universidade Federal do Ceará

---

Nesse trabalho, apresentamos o padrão *Partial View*, que pode ser utilizado por desenvolvedores de sistemas para Internet criados sob o padrão MVC (*Model-View-Controller*). Na construção de *views* (camada de visualização), muitas vezes, faz-se necessária a repetição de partes de funcionalidades para a montagem das páginas. Sendo assim, usando o padrão *Partial View*, pode-se evitar a duplicidade de código separando esse fragmento de funcionalidade em um único arquivo. Depois, pode-se incluir essa parte isolada em várias partes do sistema, centralizando a funcionalidade e facilitando modificações futuras. Esse arquivo pode conter código dinâmico ou não, e pode receber parâmetros para diferenciar algumas partes da *view*.

Categories and Subject Descriptors: H.5.4 [Information Interfaces and Presentation]: Hypertext/Hypermedia—*Architectures*; D.2.11 [Software Engineering]: Software Architectures—*Patterns*

General Terms: Web

Additional Key Words and Phrases: Programação, Web, Reutilização, MVC

## ACM Reference Format:

Viana, R. and Oliveira, T. and Trinta, F. and Andrade, R. 2014. Partial View: Um padrão para reutilização de *views* na arquitetura MVC. *jn* 2, 3, Article 1 (November 2014), 7 pages.

---

## 1. INTRODUÇÃO

Descrito primeiramente em 1979 [Reenskaug 1979] por Trygve Reenskaug (trabalhando em Smalltalk na Xerox), o padrão arquitetural MVC (Model-View-Controller) passou a ser bastante usado no desenvolvimento de sistemas a partir da década de 90 [Kotek ; Salihefendic ].

Trata-se de um padrão arquitetural orientado a objetos que separa a interface de usuário (*view* - visão) da lógica de negócios (*model* - modelo). O MVC, cuja estrutura de classes é mostrada na Figura 1[Salihefendic ], separa essas responsabilidades em três componentes, cada qual com suas responsabilidades [Popadiyn 2008]:

—A camada *model* é responsável pela lógica de negócios da aplicação e por prover estados que serão coletados pelo *controller*.

---

Este trabalho é apoiado pelo Grupo de Redes, Engenharia de Software e Sistemas (GREat) do Programa de Mestrado e Doutorado em Ciência da Computação (MDCC) da Universidade Federal do Ceará (UFC).

Endereços dos autores: Ricardo Viana. email: ricardoviana@great.ufc.br. Thalison Oliveira. email: thalissonoliveira@great.ufc.br. Fernando Trinta. email: fernandotrinta@great.ufc.br. Rossana Andrade: rossana@great.ufc.br.

Uma versão preliminar deste padrão foi apresentada na Miniconferência Latino-Americana de Linguagens de Padrões para Programação - MiniPlop 2013.

SugarLoafPLoP'14, 9 a 12 de Novembro, Ilha Bela, São Paulo, Brasil.

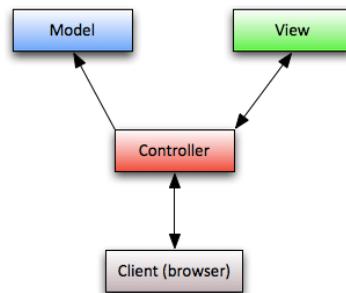


Fig. 1. Estrutura de classes do padrão MVC [Salihefendic ]

- A camada *view* é responsável somente pela apresentação dos elementos da interface de usuário. É ela que determina como os dados dos modelos serão mostrados. Na maioria das implementações, a *view* recebe um estado e os dados que ele necessita e os formata para visualização direta do usuário.
- A camada *controller* é responsável pela interação entre *view* e *model*. Ele recebe as entradas do usuário através de uma *view* e devolve os dados correspondentes coletados da camada *model* formatados através de outra *view*.

Neste trabalho, focaremos na camada de visualização (*view*), que renderiza o conteúdo de uma parte particular do modelo e encaminha para o controlador as ações do usuário. Acessa também os dados do modelo via controlador e define como esses dados devem ser apresentados [Macoratti 2012b].

Em se tratando de aplicações para a Internet, as *views* podem ser entendidas como código HTML (*HyperText Markup Language*) comum, com a vantagem de ser possível adicionar partes dinâmicas em alguma linguagem de programação como PHP, Ruby ou Java. Isso torna as *views* dinâmicas e parametrizáveis, o que as leva a ser potencialmente reutilizáveis.

## 2. CONTEXTO

Ao implementar um sistema para a Internet usando o padrão arquitetural MVC, um desenvolvedor (ou equipe de desenvolvimento) pode perceber que uma *view* (ou parte dela) é repetida em várias partes da aplicação. Além da duplicação desse código em várias partes da aplicação, pode-se enfrentar problemas com a manutenção do mesmo, caso seja necessário alterar a lógica do código repetido.

Na Figura 2 temos um exemplo de repetição de conteúdo em *views*, no qual um formulário *HTML* é idêntico nas funcionalidades de cadastro e edição de, por exemplo, professores, em um sistema hipotético.

## 3. PROBLEMA

Como um desenvolvedor de sistemas para a Internet, que está usando o padrão arquitetural MVC, pode evitar a duplicidade de código nas *views* (camada de visualização)?

## 4. SOLUÇÃO

Para evitar a duplicação de código nas *views*, pode-se separar uma funcionalidade parcial, necessária em diversas partes da implementação, num arquivo isolado e incluí-lo nos locais onde essa funcionalidade é requerida. Esse arquivo criado é chamado de *partial view* (ou somente *partial*) e pode receber parâmetros com configurações simples ou dados a serem mostrados na *view*.

Nos exemplos das Figuras 2.a e 2.b toda a parte do formulário para cadastro dos dados do professor é rigorosamente igual, tanto para cadastro quanto para edição. Dessa forma, usando o padrão *Partial View*, coloca-se essa parte igual em um arquivo distinto e faz-se a inclusão dele nas *views* originais.

Fig. 2. Exemplos de *views* para (a) cadastro e (b) edição de professores

Sendo assim, o padrão *Partial View* pode ser aplicado no desenvolvimento de *views* de duas maneiras: (a) após as *views* originais já existirem pode haver refatoração para que as *partial views* sejam criadas e o código reutilizado; ou, (b) antes da criação das *views*, ao projetar como serão as telas da aplicação, o *designer* pode detectar partes parecidas (ou iguais) e incluir a criação de *partial views*, aplicando o padrão em tempo de projeto.

## 5. CONSEQUÊNCIAS

Como principais consequências que podem ser atingidas com a utilização do padrão *Partial View* podemos citar:

- Se existirem erros nas *partial views*, eles são propagados para todas as *views* onde elas são incluídas.
- A utilização de *partial views* provoca a leitura não contínua do código, gerando a necessidade de acessar o mesmo ao longo de vários arquivos separados.
- Quando vários parâmetros são necessários de serem passados para as *partial views* fica mais difícil seu uso e, conseqüentemente, a implementação das *partial views* é mais complexa por ter de tratar esses parâmetros.

## 6. IMPLEMENTAÇÃO

São ilustradas nas Figuras 3 e 4 uma visão geral das abordagens para uso do padrão *Partial View*. Na primeira abordagem, o desenvolvedor pode utilizar o padrão para realizar a refatoração em um código já implementado.

Ao usar o padrão na refatoração de código, em um primeiro momento as *views* são criadas pelo desenvolvedor. Em seguida, esse código é avaliado e detecta-se que existem partes repetidas. Uma *Partial View* então é criada com a parte repetida do código e, nas *views* originais, faz-se a inclusão da *Partial View* (Figura 3).

No entanto, caso o desenvolvedor já conheça o padrão, pode planejar as *partial views* no início do projeto, evitando a repetição do código e diminuindo a necessidade de refatoração. Nesse caso, o desenvolvedor planeja as *Partial Views* antes mesmo de implementar as *views* para que, ao criá-las, apenas faça a inclusão das mesmas (Figura 4).

## 7. EXEMPLO

Utilizando-se o exemplo da Figura 2, pode-se verificar a forma de aplicar o padrão *Partial* da maneira a seguir:

Suponhamos que o fragmento de código fonte correspondente a implementação da tela de cadastro de professores mostrado na Figura 2.a seja o mostrado no Código 1. Nesse caso, usou-se o *framework Ruby on Rails*, no qual esse arquivo ficaria em uma pasta específica para *views*.

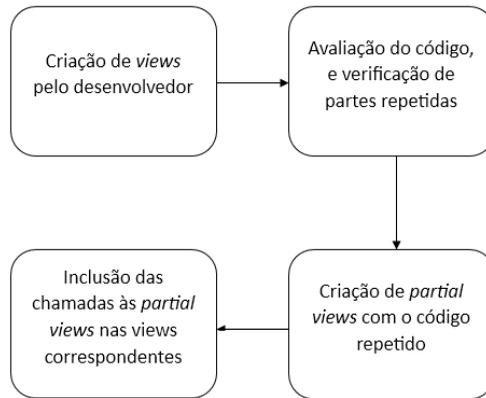


Fig. 3. Dinâmica do Padrão *Partial* no contexto de refatoração

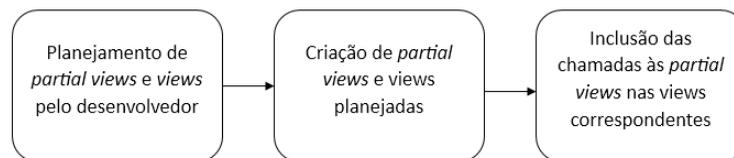


Fig. 4. Dinâmica do Padrão *Partial* no contexto de planejamento de views

---

**Código 1:** Formulário de Cadastro de Professor

---

```

<h1>Cadastrar Professor</h1>

<%= form_for @teacher do |f| %>
  <%= f.label :name, "Nome"%>
  <%= f.text_field :name %>

  <%= f.label :email, "E-mail"%>
  <%= f.text_field :email %>

  <%= f.label :program, "Programa"%>
  <%= f.text_field :program %>

  <%= f.label :lattes, "Currículo lattes"%>
  <%= f.text_field :lattes %>

  <%= f.submit "Criar"%>
<% end %>

<%= link_to "Ir para listagem", teachers_path %>
  
```

---

Suponhamos, ainda, que o fragmento de Código 2 é responsável pela tela de edição de professores apresentada na figura 2.b. Analogamente, esse arquivo está localizado, na mesma pasta de *views* que o código anterior.

---

**Código 2:** Formulário de Edição de Professor

---

```
<h1>Editar Professor</h1>
```

```
<%= form_for @teacher do |f| %>
  <%= f.label :name, "Nome"%>
  <%= f.text_field :name %>

  <%= f.label :email, "E-mail"%>
  <%= f.text_field :email %>

  <%= f.label :program, "Programa"%>
  <%= f.text_field :program %>

  <%= f.label :lattes, "Currículo lattes"%>
  <%= f.text_field :lattes %>

  <%= f.submit "Editar"%>
<% end %>

<%= link_to "Visualizar", @teacher %>
```

---

Pode-se perceber que as duas *views* tem muitas semelhanças e, segundo o padrão proposto, são fortes candidatos à refatoração. Para isso, criaremos um novo arquivo, em *Ruby on Rails* na mesma pasta das *views* originais, com um nome qualquer, sendo que deve ser iniciado pelo caractere *underscore* (`_`). Nesse arquivo, são colocadas as partes comuns encontradas nos arquivos originais, como mostrado no Código 3.

---

**Código 3:** Partial View `_form.html.erb`

---

```
<%= form_for @teacher do |f| %>

  <%= f.label :name, "Nome"%>
  <%= f.text_field :name %>

  <%= f.label :email, "E-mail"%>
  <%= f.text_field :email %>

  <%= f.label :program, "Programa"%>
  <%= f.text_field :program %>

  <%= f.label :lattes, "Currículo lattes"%>
  <%= f.text_field :lattes %>

  <%= f.submit botao %>
<% end %>
```

---

Dessa forma, as *views* originais são simplificadas, contendo apenas as partes diferentes e a implementação do formulário passa a ser centralizada. Pode-se retirar a parte que foi movida para a *partial view* e colocar uma chamada como mostrado no código 4 para incluir essa *partial view* nos arquivos originais.

---

**Código 4:** Inclusão de Partial em Ruby on Rails

---

```
<%=
  render :partial => 'nomeDoArquivoSemUnderscore'
%>
```

---

Observa-se, ainda, que o formulário possui uma pequena diferença: no botão a ser clicado para submeter o formulário tem-se o texto “Cadastrar” na *view* de cadastro, enquanto tem-se o texto “Editar” na *view* de edição.

Assim, é necessário passar esse texto como parâmetro para a *partial view*, conforme mostrado no código 5, na mesma chamada de inclusão da *partial view*.

Um parâmetro pode ser usado como uma variável para mostrar algum conteúdo dinâmico. Ao criar esse parâmetro, o *framework* disponibiliza uma variável de mesmo nome na *partial view* para ser usada a qualquer momento. Nesse caso (Código 5), a variável 'botao' pode ser usada para tornar uma *view* diferente da outra.

Essa mesma abordagem pode ser usada para modificações maiores, como habilitar/desabilitar campos no formulário ou esconder/mostrar informações do usuário.

---

**Código 5:** Inclusão de Partial em Ruby on Rails passando parâmetro

---

```
<%=  
  render :partial => 'nomeDoArquivoSemUnderscore', :locals => { :botao => 'Cadastrar' }  
%>
```

---

A *view new.html.erb* fica como o código 6 e a *view edit.html.erb* fica como o código 7.

---

**Código 6:** *View* de cadastro usando a *partial view* criada

---

```
<h1>Cadastrar Professor</h1>  
  
<%= render :partial => 'form', :locals => { :botao => 'Cadastrar' } %>  
  
<%= link_to "Ir para listagem", teachers_path %>
```

---

---

**Código 7:** *View* de edição usando a *partial view* criada

---

```
<h1>Editar Professor</h1>  
  
<%= render :partial => 'form', :locals => { :botao => 'Editar' } %>  
  
<%= link_to "Visualizar", @teacher %>
```

---

## 8. PADRÕES RELACIONADOS

—A utilização do padrão MVC [Reenskaug 1979] é mandatória, devido ao isolamento da lógica da aplicação da interface apresentada para o usuário final, permitindo o desenvolvimento independente.

—Relaciona-se com o padrão *Singleton* [sin ] por ser uma aplicação similar a esse padrão, porém aplicada em outra área, na construção de *views*, no qual tem-se apenas um lugar onde um pedaço de *view* será instanciado.

## 9. USOS CONHECIDOS

O padrão *Partial View* já pode ser encontrado em uso em diversos *frameworks* de desenvolvimento para Internet baseados no padrão MVC. Pode-se citar *ASP.NET*, *Ruby on Rails* e *PHP Zend Framework* como exemplos que suportam a implementação de *partial views*.

Em *ASP.NET*, ao criar uma *view*, o programador pode selecionar na *IDE* Visual Studio [vis ] (Ambiente Integrado de Desenvolvimento padrão desenvolvido pela Microsoft para esse *framework*) a opção "Create as a partial view". O arquivo *partial view* é gerado e pode ser invocado por outras *views* com o comando mostrado no Código 8 [Macoratti 2012a].

---

**Código 8:** Inclusão de Partial View em ASP.NET

---

```
@Html.Partial(  
  "NomeDaPartialView", Model.nomeMetodoDoModelo  
)
```

---

No *framework Ruby on Rails*, as *partial views* são identificadas ao nomear o arquivo iniciando com o caractere *underscore* (por exemplo `_partial.html.erb`). Para usar essa *partial view* em outras *views*, invoca-se com o Código 9. Com isso, *framework* buscará pelo arquivo de *partial view* chamado `'_nomeDoArquivoSemUnderscore.html.erb'`, e renderizará o conteúdo do mesmo [Varella 2008].

---

**Código 9:** Inclusão de Partial View em Ruby on Rails

---

```
<%=  
  render :partial => 'nomeDoArquivoSemUnderscore'  
%>
```

---

Por último, no *PHP Zend Framework*, de maneira similar ao *ASP.NET*, o desenvolvedor cria um arquivo *partial*, e pode invocá-lo através de uma implementação como o código 10, indicando que o arquivo *partial view* nomeado 'adicionado.phtml' deve ser incluído nesse ponto do código [zen]. A extensão 'phtml' foi criada para identificar arquivos de *partial views* nesse *framework*.

---

**Código 10:** Inclusão de Partial View em PHP

---

```
<?php  
  echo $this->partial('adicionado.phtml', array('nomeParametro' => 'ValorParametro');  
?>
```

---

## 10. AGRADECIMENTOS

Os autores gostariam de agradecer a colaboração dos membros do *GREat* que ajudaram o desenvolvimento deste trabalho, em especial aos componentes da disciplina de Reutilização de *Software* em 2012.2 (Rute, que funcionou como *shepherd* do trabalho na disciplina) na qual o padrão foi desenvolvido, bem como a Ayla Rebouças pela valiosa contribuição no processo de *shepherdig* deste padrão quando submetido ao MiniPlop 2013.

### REFERENCES

- Gang of four patterns. Disponível por www em: <http://www.gofpatterns.com/>, acesso em 17/01/2013.
- Microsoft visual studio. Disponível por www em: <http://www.microsoft.com/visualstudio/ptb/products/visual-studio-overview>, acesso em 10/01/2013.
- Zend framework manual: Partial helper. Disponível por www em: <http://framework.zend.com/manual/1.12/en/zend.view.helpers.html#zend.view.helpers.initial.partial>, acesso em 09/01/2013.
- KOTEK, B. Mvc design pattern brings about better organization and code reuse. Disponível por www em: <http://www.techrepublic.com/article/mvc-design/1049862>, acesso em 04/02/2013.
- MACORATTI, J. C. 2012a. Asp .net mvc 3: apresentando partial views. Disponível por www em: <http://imasters.com.br/artigo/24124/dotnet/asp-net-mvc-3-apresentando-partial-views/>, acesso em 08/01/2013.
- MACORATTI, J. C. 2012b. Padrões de projeto : O modelo mvc - model view controller. Disponível por www em: [http://www.macoratti.net/vbn\\_mvc.htm](http://www.macoratti.net/vbn_mvc.htm), acesso em 10/01/2013.
- POPADIYN, P. 2008. Exploring the model – view – controller (mvc) pattern. Disponível por www em: <http://www.silverlightshow.net/items/Exploring-the-Model-View-Controller-MVC-pattern.aspx>, acesso em 10/01/2013.
- REENSKAUG, T. 1979. Thing-model-view-editor-an example from a planning system. *Xerox PARC technical note 12*.
- SALIHENDIC, A. Model view controller: History, theory and usage. Disponível por www em: <http://amix.dk/blog/post/19615>, acesso em 04/02/2013.
- VARELLA, A. 2008. Partials, reutilização de código no rails. Disponível por www em: <http://railsgirl.wordpress.com/2008/08/05/partials-reutilizacao-de-codigo-no-rails/>, acesso em 08/01/2013.