**Padrões de Projetos – Padrão MVC**

História do MVC

A arquitetura MVC (Model-View-Controller) foi criada nos anos 80 na Xerox Parc, por Trygve Reenskaug, que iniciou em 1979 o que viria a ser o nascimento do padrão de projeto MVC. A implementação original foi descrita no artigo “Applications Programming in Smalltalk-80: How to use Model-View-Controller”. [5]

Os componentes do MVC

Tradicionalmente usado para interfaces gráficas de usuário (GUIs), esta arquitetura tornou-se popular para projetar aplicações web e até mesmo para aplicações móveis, para desktop e para outros clientes.[6] Linguagens de programação populares como Java, C#, Object Pascal/Delphi, Ruby, PHP, JavaScript e outras possuem frameworks MVC populares que são atualmente usados no desenvolvimentos de aplicações web.

**Camada de modelo ou da lógica da aplicação (Model)**

Modelo é a ponte entre as camadas Visão (*View*) e Controle (*Controller*),[2] consiste na parte lógica da aplicação, que gerencia o comportamento dos dados através de regras de negócios, lógica e funções.[7] Esta fica apenas esperando a chamada das funções,[7] que permite o acesso para os dados serem coletados, gravados e, exibidos.[2]

É o coração da execução, responsável por tudo que a aplicação vai fazer a partir dos comandos da camada de controle em um ou mais elementos de dados, respondendo a perguntas sobre o sua condição e a instruções para mudá-las. O modelo sabe o que o aplicativo quer fazer e é a principal estrutura computacional da arquitetura, pois é ele quem modela o problema que está se tentando resolver. Modela os dados e o comportamento por trás do processo de negócios. Se preocupa apenas com o armazenamento, manipulação e geração de dados. É um encapsulamento de dados e de comportamento independente da apresentação.

**Camada de apresentação ou visualização (View)**

Visão pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama. É onde os dados solicitados do Modelo (*Model*) são exibidos.[2] É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores. A Visão também provoca interações com o usuário, que interage com o Controle (*Controller*). O exemplo básico disso é um botão gerado por uma Visão, no qual um usuário clica e aciona uma ação no Controle.[2]

Não se dedica em saber como o conhecimento foi retirado ou de onde ela foi obtida, apenas mostra a referência. Segundo Gamma et al (2006), ”A abordagem MVC separa a View e Model por meio de um protocolo inserção/notificação (subscribe/notify). Uma View deve garantir que sua expressão reflita o estado do Model. Sempre que os dados do Model mudam, o Model altera as Views que dependem dele. Em resposta, cada View tem a oportunidade de modificar-se”. Adiciona os elementos de exibição ao usuário: HTML, ASP, XML, Applets. É a camada de interface com o usuário. É utilizada para receber a entrada de dados e apresentar visualmente o resultado.

**Camada de controle ou controlador (Controller)**

Controle é o componente final da tríade, faz a mediação da entrada e saída, comandando a visão e o modelo para serem alterados de forma apropriada conforme o usuário solicitou através do mouse e teclado.[7] O foco do Controle é a ação do usuário, onde são manipulados os dados que o usuário insere ou atualiza, chamando em seguida o Modelo.[2]

O Controle (Controller) envia essas ações para o Modelo (Model) e para a janela de visualização (View) onde serão realizadas as operações necessárias.



Vamos a um exemplo…

Vamos utilizar o exemplo de uma página web, onde o usuário pode realizar o cadastro de clientes. Neste caso, provavelmente você teria uma classe chamada cliente.php que contém as informações do cliente que você deseja guardar (como nome, endereço, cidade, etc.). Essa classe seria o seu model.

Aqui, ainda poderíamos acoplar aspectos de manipulação de bancos de dados, concentrando nesta estrutura os métodos para inserir, alterar, excluir e listar os clientes a partir de uma tabela em um banco de dados.

A página HTML seria nossa view, que mostraria, por exemplo, a lista de usuários cadastrados ou mesmo o próprio formulário para cadastro de novos usuários.

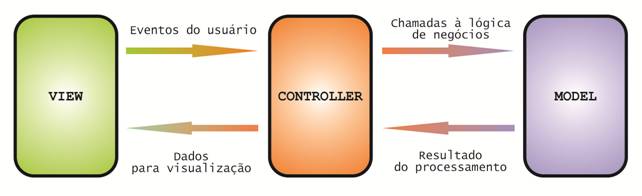
O controller faz o meio de campo entre o model e a view. Ele é necessário porque as estruturas presentes com view não deveriam acessar diretamente os models, já que isso poderia criar um acoplamento entre as estruturas de apresentação e definição de negócio: é necessária uma estrutura intermediária para fazer essa ligação.

E aqui entra o controller, que age como uma ponte entre os dois. Você pode ter uma classe dentro do seu projeto [PHP](https://www.treinaweb.com.br/blog/o-que-e-php/) para fazer o papel de um controller, realizando a ligação entre models e views.

Interação dos componentes

Além de dividir a aplicação em três tipos de componentes, o desenho MVC define as interações entre eles.

* O **Controlador** (**controller**) envia comandos para o modelo para atualizar o seu estado (por exemplo, editando um documento). O controlador também pode enviar comandos para a visão associada para alterar a apresentação da visão do modelo (por exemplo, percorrendo um documento).
* Um **modelo** (**model**) armazena dados e notifica suas visões e controladores associados quando há uma mudança em seu estado. Estas notificações permitem que as visões produzam saídas atualizadas e que os controladores alterem o conjunto de comandos disponíveis. Uma implementação *passiva* do MVC monta estas notificações, devido a aplicação não necessitar delas ou a plataforma de software não suportá-las.
* A **visão** (**view**) gera uma representação (Visão) dos dados presentes no modelo solicitado, fazendo a exibição dos dados, sendo ela por meio de um html ou xml.



Uso em aplicações web

Mesmo tendo sido desenvolvida inicialmente para computação pessoal, o MVC foi drasticamente adaptado como uma arquitetura para as aplicações web em todas as maiores linguagens de programação. Muitos frameworks de aplicação comerciais e não comerciais foram desenvolvidos tendo como base esse modelo. Tais frameworks variam em suas interpretações, principalmente no modo que as responsabilidades MVC são separadas entre o cliente e servidor.

Os frameworks web MVC mais recentes levam uma abordagem de thin client que quase colocou o modelo, a visão e a lógica do controlador inteiros no servidor. Nesta abordagem, o cliente envia requisições de hiperlink ou entrada de formulário ao controlador e então recebe uma página web completa e atualizada (ou outro documento) da visão. O modelo existe inteiramente no servidor. Como as tecnologias de cliente amadureceram, frameworks como JavaScriptMVC e Backbone foram criados o que permite que os componentes MVC executem parcialmente no cliente (ver também AJAX).

Um caso prático é uma aplicação web em que a visão é um documento HTML (ou derivado) gerado pela aplicação. O controlador recebe uma entrada GET ou POST após um estímulo do utilizador e decide como processá-la, invocando objetos do domínio para tratar a lógica de negócio, e por fim invocando uma visão para apresentar a saída

Vantagens e desvantagens

**Vantagens do modelo MVC:**

1. Como o modelo MVC gerencia múltiplos views usando o mesmo modelo é fácil manter, testar e atualizar sistemas compostos;
2. É muito simples adicionar novos clientes apenas incluindo seus views e controles;
3. Torna a aplicação escalável;
4. É possível ter desenvolvimento em paralelo para o modelo, visualizador e controle pois são independentes;
5. Facilita o reuso do código;
6. Melhor nível de sustentabilidade, pois facilita a manutenção da aplicação;
7. Fácil transformação da interface, sem que haja necessidade de modificar a camada de negócio;
8. Melhor desempenho e produtividade, graças a estrutura de pacotes modulares;
9. A arquitetura modular permite aos desenvolvedores e designers desenvolverem em paralelo;
10. Partes da aplicação podem ser alteradas sem a necessidade de alterar outras.

**Desvantagens do modelo MVC:**

1. Necessita de um tempo maior para explorar e modelar o sistema;
2. Requer mão-de-obra especializada;
3. À medida que o tamanho e a complexidade do projeto crescem, a quantidade de arquivos e pastas continuará aumentando também. Os interesses de UI (interface do usuário) (modelos, exibições, controladores) se localizam em várias pastas, que não são formadas em grupos por ordem alfabética.

Model: Utilizando DAO + Javabeans

É opcional, mas recomendado como uma boa prática a utilização de duas camadas no Model.

São elas: uma classe **DAO** *(Data Access Object – Objeto de Acesso a Dados)* , responsável central efetivamente pelos acessos e operações ao banco de dados, e uma classe conhecida como **Javabeans**, onde será definido um objeto (a “caixa”) que carregará os dados a serem transportados ao longo das camadas ou módulos.

Trabalhar neste padrão irá gerar um trabalho adicional, mas trará benefícios como encapsulamento, isolamento, organização, padronização e modularização do código.

Além disso todo o acesso ao banco de dados fica centralizado em um só lugar: no DAO.

