

## **Modelo MVC - Model View Controller**

**Introdução**

**Aplicações Monolíticas**

**Aplicações em 2 Camadas**

**Aplicações em 3 Camadas**

**Padrão MVC**

## Introdução

Os Padrões de Projeto são muito úteis para resolver problemas de modelagem de projetos se usados de forma adequada. Mas o que são estes benditos padrões de projetos ou Design Patterns ?

Padrões de projetos são soluções para problemas que alguém um dia teve e resolveu aplicando um modelo que foi documentado e que você pode adaptar integralmente ou de acordo com necessidade de sua solução. Vou abordar neste artigo o padrão de modelo MVC que tem por objetivo básico separar a lógica de negócio da apresentação.

O grande desafio das equipes de desenvolvimento de aplicações é cada vez mais produzir aplicativos seguros , eficientes , de fácil manutenção , reutilizáveis e em prazos cada vez menores.

O paradigma da orientação a objetos tem tido um grande avanço nestes últimos tempos (leia-se Java e agora .NET) , tanto é que , segundo o *Partner Group* , dentre as tecnologias que vão sobreviver no mercado nos próximos anos estão Java e .NET. Você então já percebeu por que a Microsoft lançou a tecnologia .NET e tornou orientada a objetos o bom e velho Visual Basic , trazendo consigo o C# para quem usa Java. Estima-se que a partir de agora será impensável produzir aplicações que não sejam orientadas a objetos desde sua concepção.

O sucesso para o desenvolvimento de aplicações com tecnologia orientada a objetos esta intimamente ligada á arquitetura que vamos usar para construir a aplicação. A tendência indica que esta arquitetura estará baseada na organização da aplicação em camadas e na observação dos padrões utilizados pelo mercado.

A organização em camadas é a chave para a independência entre os componentes e esta independência é que vai atingir os objetivos de eficiência , escalabilidade , reutilização e facilidade de manutenção.

Num primeiro instante produzir aplicativos multicamadas pode parecer mais complexo por isto vou procurar mostrar como evoluímos para chegar a esta solução. O termo camada pode significar uma separação física ou uma camada lógica , no nosso caso , a produção de software vamos considerar camada como uma referência a separação de responsabilidades.

## Aplicações monolíticas

Nos tempos antigos do reinado do grande porte e do computador pessoal independente um aplicativo era desenvolvido para ser usado em uma única

máquina . Geralmente este aplicativo continha todas a funcionalidades em um único módulo gerado por uma grande quantidade de linhas de código e de manutenção nada fácil. A entrada do usuário , verificação , lógica de negócio e acesso a banco de dados estava presente em um mesmo lugar. Podemos definir este tipo de aplicação como aplicação de uma camada ou monolítica , esquematizada a seguir :



### Aplicações em 2 camadas

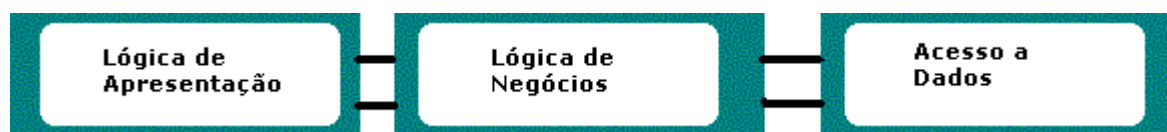
A necessidade de compartilhar a lógica de acesso a dados entre vários usuários simultâneos fez surgir as aplicações em duas camadas. Nesta estrutura a base de dados foi colocada em uma máquina específica, separada das máquinas que executavam as aplicações. Nesta abordagem temos aplicativos instalados em estações clientes contendo toda a lógica da aplicação (clientes ricos ou gordos). Um grande problema neste modelo é o gerenciamento de versões pois para cada alteração os aplicativos precisam ser atualizados em todas as máquinas clientes.



### Aplicações em 3 camadas

Com o advento da internet houve um movimento para separar a lógica de negócio da interface com o usuário. A idéia é que os usuários da WEB possam acessar as mesmas aplicações sem ter que instalar estas aplicações em suas máquinas locais. Como a lógica do aplicativo , inicialmente contida no cliente rico não reside mais na máquina do usuário este tipo de cliente passou a ser chamado de cliente pobre ou magro.(thin).

Neste modelo o aplicativo é movido para o Servidor e um navegador Web é usado como um cliente magro. O aplicativo é executado em servidores Web com os quais o navegador Web se comunica e gera o código HTML para ser exibido no cliente.



Neste modelo a lógica de apresentação esta separada em sua própria camada lógica e física. A separação em camadas lógicas torna os sistemas mais flexíveis permitindo que as partes possam ser alteradas de forma independente. As funcionalidades da camada de negócio podem ser divididas em classes e essas classes podem ser agrupadas em pacotes ou componentes reduzindo as dependências entre as classes e pacotes ; podem ser reutilizadas por diferentes partes do aplicativo e até por aplicativos diferentes. O modelo de 3 camadas tornou-se a arquitetura padrão para sistemas corporativos com base na Web.

A modelagem orientada a objetos ajuda a promover a modularidade pois os objetos encapsulam seus dados (propriedades , estados) e oferecem funcionalidades através de seus métodos. Projetando-se de forma adequada os objetos podem ter reduzidas as dependências entre si ficando assim fracamente acoplados e serão mais fáceis de manter e evoluir.

## **Padrão MVC**

O modelo de três camadas físicas ( 3-tier ) divide um aplicativo de modo que a lógica de negócio resida no meio das três camadas físicas. Isto é chamado de camada física intermediária ou camada física de negócios. A maior parte do código escrito reside na camada de apresentação e de negócio.

A arquitetura **MVC** - (Modelo Visualização Controle) fornece uma maneira de dividir a funcionalidade envolvida na manutenção e apresentação dos dados de uma aplicação. A arquitetura MVC não é nova e foi originalmente desenvolvida para mapear as tarefas tradicionais de entrada , processamento e saída para o modelo de interação com o usuário. Usando o padrão MVC fica fácil mapear esses conceitos no domínio de aplicações Web multicamadas.

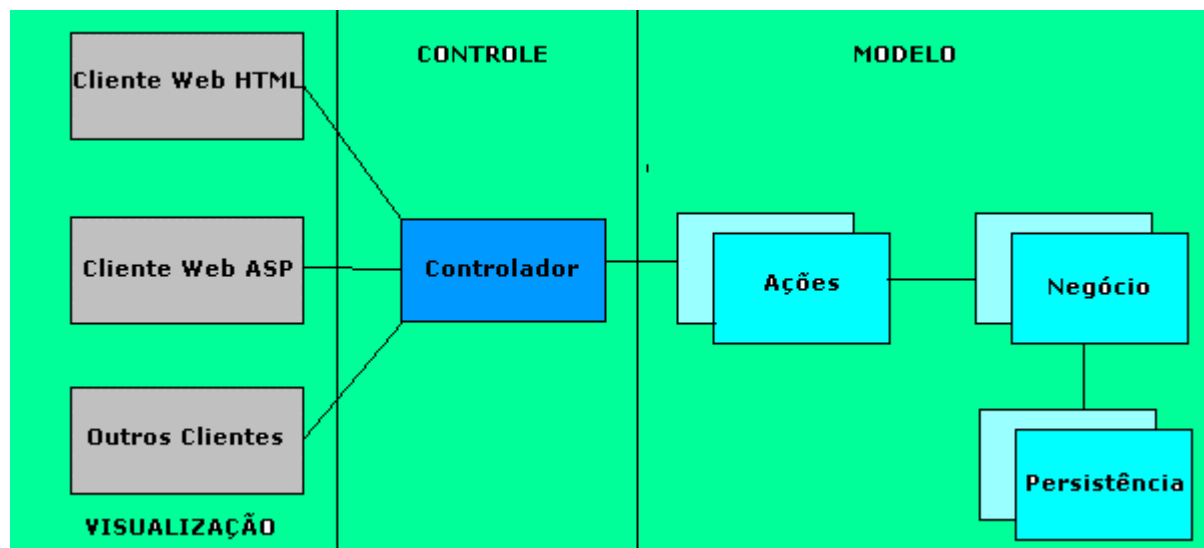
Na arquitetura MVC o modelo representa os dados da aplicação e as regras do negócio que governam o acesso e a modificação dos dados. O modelo mantém o estado persistente do negócio e fornece ao controlador a capacidade de acessar as funcionalidades da aplicação encapsuladas pelo próprio modelo.

Um componente de visualização 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.

Um controlador define o comportamento da aplicação , é ele que interpreta as ações do usuário e as mapeia para chamadas do modelo. Em um cliente de aplicações Web essas ações do usuário poderiam ser cliques de botões ou seleções de menus. As ações realizadas pelo modelo incluem ativar processos de negócio ou alterar o estado do modelo. Com base na ação do usuário e no resultado do processamento do modelo , o controlador seleciona uma

visualização a ser exibida como parte da resposta a solicitação do usuário. Há normalmente um controlador para cada conjunto de funcionalidades relacionadas.

A arquitetura de 3 camadas que esta representada abaixo é uma implementação do modelo MVC . O modelo MVC esta preocupado em separar a informação de sua apresentação.



**Camada de apresentação ou visualização** - Não esta preocupada em como a informação foi obtida ou onde ela foi obtida apenas exibe a informação.

- inclui os elementos de exibição no cliente : HTML , XML , ASP , Applets .
- É a camada de interface com o usuário.
- É usada para receber a entrada de dados e apresentar o resultado

**Camada de lógica da Aplicação** - É o coração da aplicação. Responsável por tudo que a aplicação vai fazer.

- modela os dados e o comportamento por atrás 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 Controle** - determina o fluxo da apresentação servindo como uma camada intermediária entre a camada de apresentação e a lógica.

- controla e mapeia as ações

**Vantagens** do modelo MVC :

1. Como o modelo MVC gerencia múltiplos visualizadores usando o mesmo modelo é fácil manter , testar e atualizar sistemas múltiplos
2. É muito simples incluir novos clientes apenas incluindo seus visualizadores e controles
3. Torna a aplicação escalável
4. É possível ter desenvolvimento em paralelo para o modelo , visualizador e controle pois são independentes.

**Desvantagens** do modelo MVC:

1. Requer uma quantidade maior de tempo para analisar e modelar o sistema
2. Requer pessoal especializado
3. Não é aconselhável para pequenas aplicações