Abordagem Teórico-Prática

Prof. Giuliano Prado M. Giglio, M.Sc

Desenvolvimento de Aplicações Distribuídas - WEB

Agenda

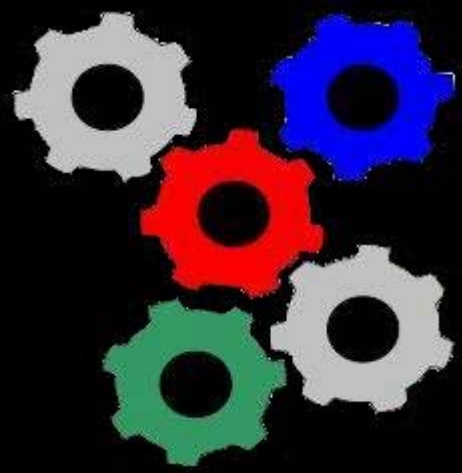
- Padrão MVC
  - Objetivo
  - Características
  - Problemas e se aplicar o MVC
  - Sistema Exemplo
  - Aplicação MVC para Web
  - Problemas que o MVC pode causar

Sistema

- **Um sistema é uma parte da realidade, ou seja, um conjunto de elementos interagindo entre si para realizar determinadas tarefas no mundo real**

Exemplos:

- **Sistema de venda**
- **Sistema de controle bancário**
- **Sistema de automação industrial**
- **Sistemas de administração de pessoal**



SISTEMA EXEMPLO

Sistema de venda simples

- Clientes compram mercadorias em uma loja;
- A mercadoria é vendida pelos funcionários da loja;
- Os funcionários da loja ganham comissão sobre suas vendas;
- O gerente da loja compra mercadorias de seus fornecedores;
- O gerente da loja controla o estoque das mercadorias;

Etapas do desenvolvimento OO

- Elicitação das operações desejadas pelos usuários
- Projeto da arquitetura do sistema
- Implementação das classes componentes da arquitetura
- Teste do sistema

Etapas do desenvolvimento OO

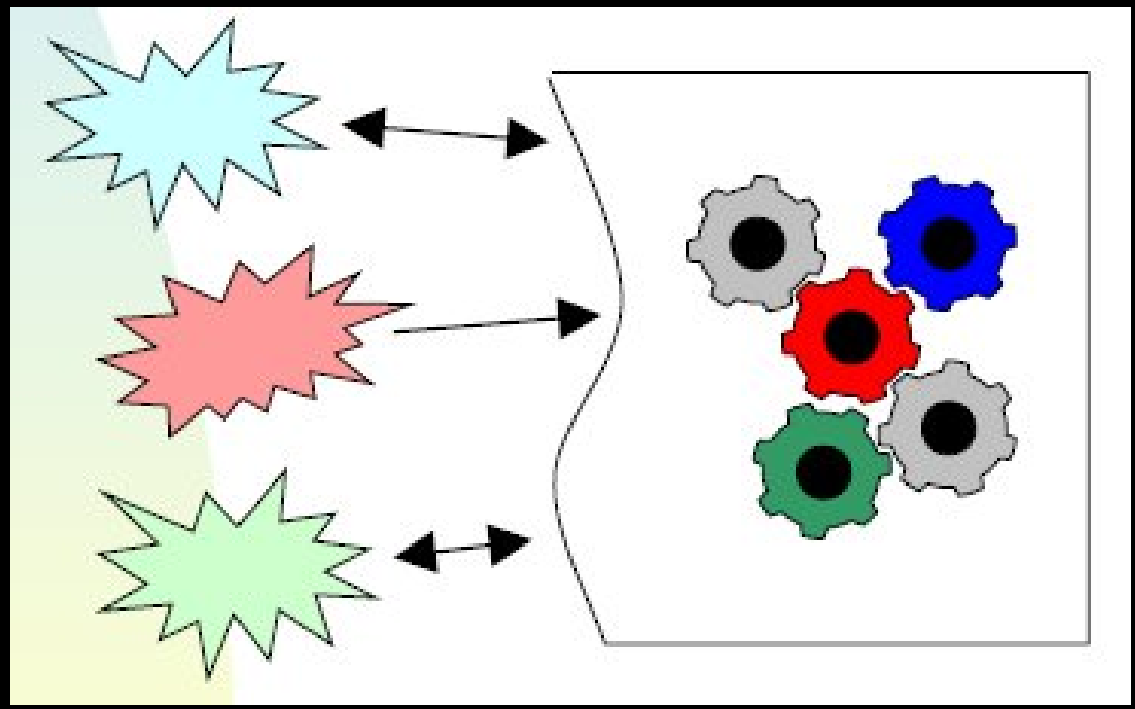
- Observamos três categorias de classes
  - Classes de domínio
  - Classes de interface
  - Classes de controle

CLASSES DE DOMÍNIO

- Classes que fazem parte do universo do problema
- Representam conceitos do mundo real
- Aspectos da solução computacional são desconsiderados
- Exemplos:
  - Cliente Mercadoria Venda
  - Funcionário Fornecedor

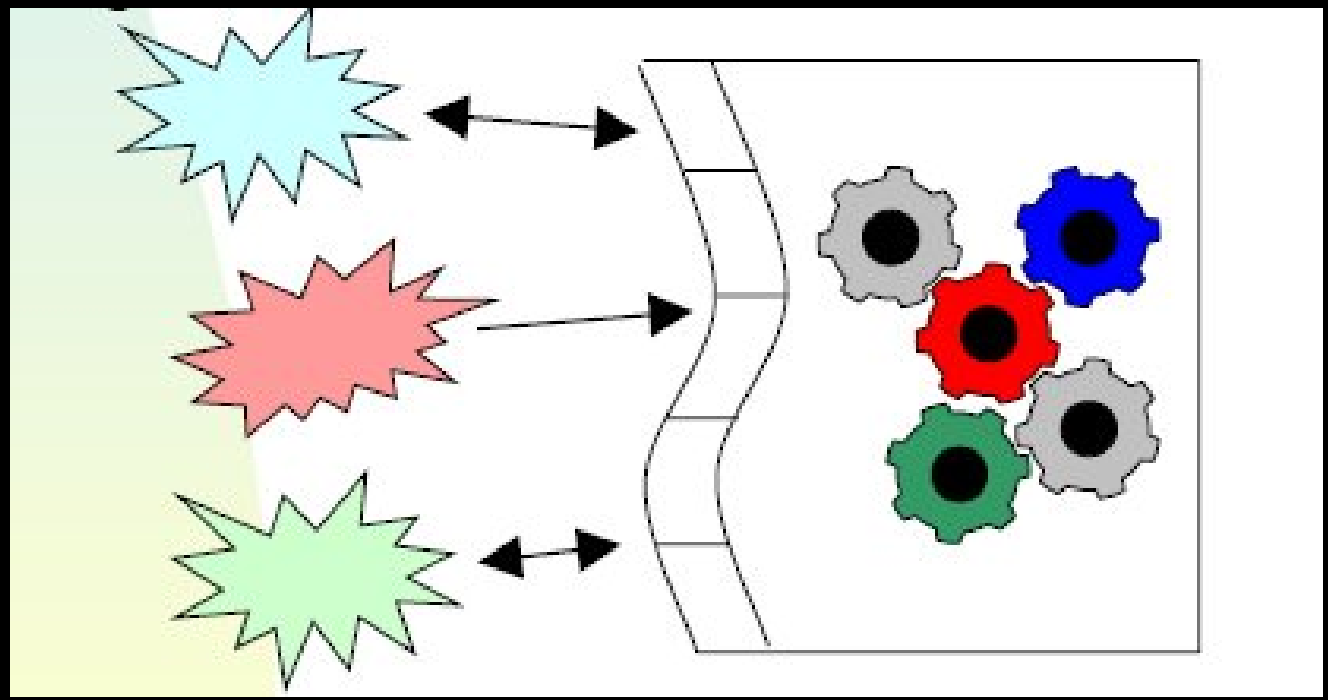
CLASSES DE INTERFACE

- Todos os sistemas de software possuem limites
  - Informações transitam pelos limites do sistema
  - Informações são inseridas ou consultadas por agentes externos



CLASSES DE INTERFACE

- Classes de interface estão no limite entre o sistema e os agentes externos que enviam e/ou recebem informações
- Elas realizam a comunicação entre os agentes e o sistema



CLASSES DE INTERFACE

- Os agentes externos podem ser:
  - Usuários (pessoas ou grupos de pessoas)
  - Periféricos ligados ao sistema
  - Outros sistemas ligados ao sistema
- As classes de interface não são responsáveis apenas pela interface com o **usuário**, mas por qualquer conexão do sistema com o **mundo exterior** !!!

CLASSES DE INTERFACE

- Exemplos:
  - Classe que representa a janela de vendas
  - Classe que representa a janela de cadastro de clientes
  - Classe que representa o relatório de nota fiscal de venda

CLASSES DE INTERFACE

- **Observação:**
  - Como o sistema não possui interface com outros sistemas ou periféricos, existem apenas interfaces com o usuário
  - Considere a interface do sistema de vendas com um sistema de folha de pagamento, em relação às comissões!

CLASSES DE CONTROLE

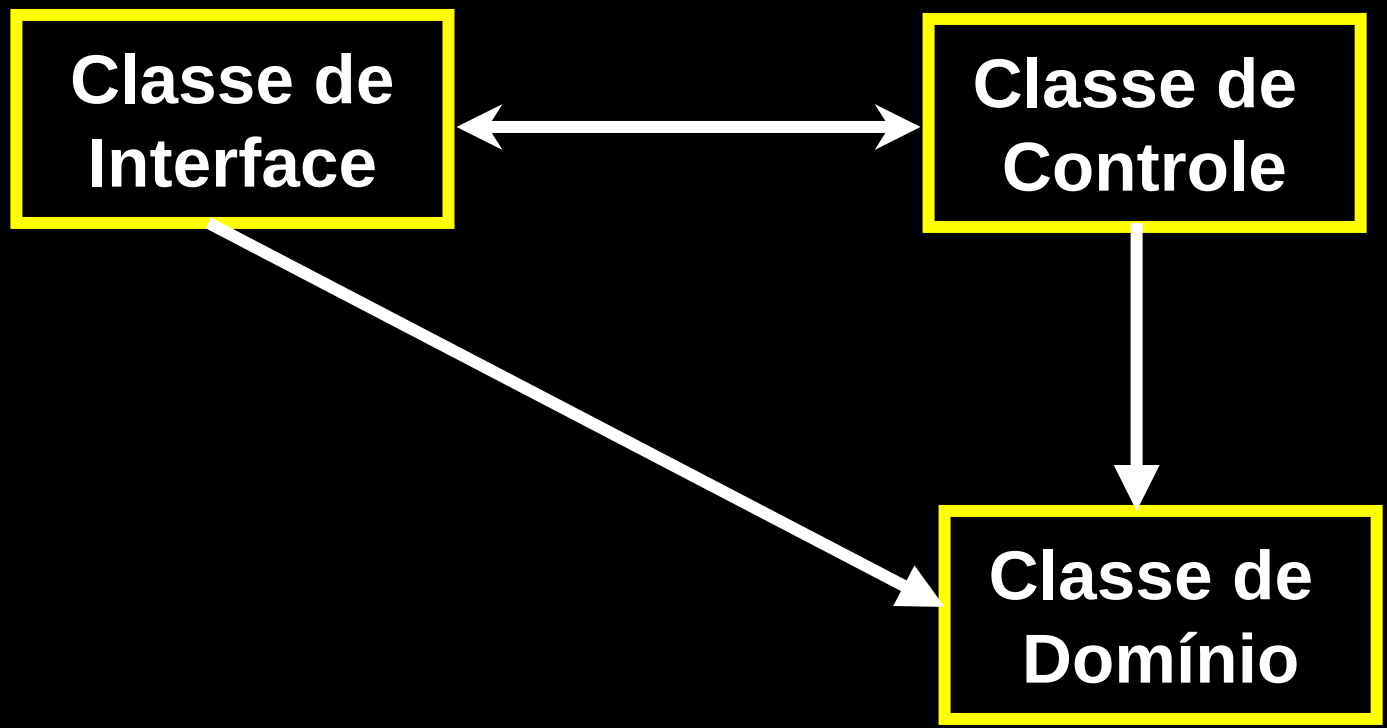
- As classes de controle administram a interação entre as classes de interface e as classes de domínio
- Elas conhecem os passos que devem ser realizados para o cumprimento de uma tarefa no sistema
- Classes de controle são utilizadas no tratamento de eventos na interface gráfica de sistemas Java
- Exemplos:
  - Gerenciador de vendas
  - Gerenciador de armazenamento

Desenvolvimento de Sistemas OO

- Determine as operações realizadas no sistema
  - Identifique os usuários que participam destas operações;
  - Identifique as classes necessárias para realizar as operações;
    - ✓ Que classes de domínio são necessárias?
    - ✓ Como será a interface com o mundo exterior?
  - Identifique as classes de controle necessárias para a operação;
    - ✓ Que operações envolvem um conjunto de classes?
    - ✓ Que operações são algoritmicamente complexas?
    - ✓ Que operações complementares são necessárias?

Padrão MVC

- Modelo Model – View - Controller
- Modelo clássico de desenvolvimento OO de aplicações

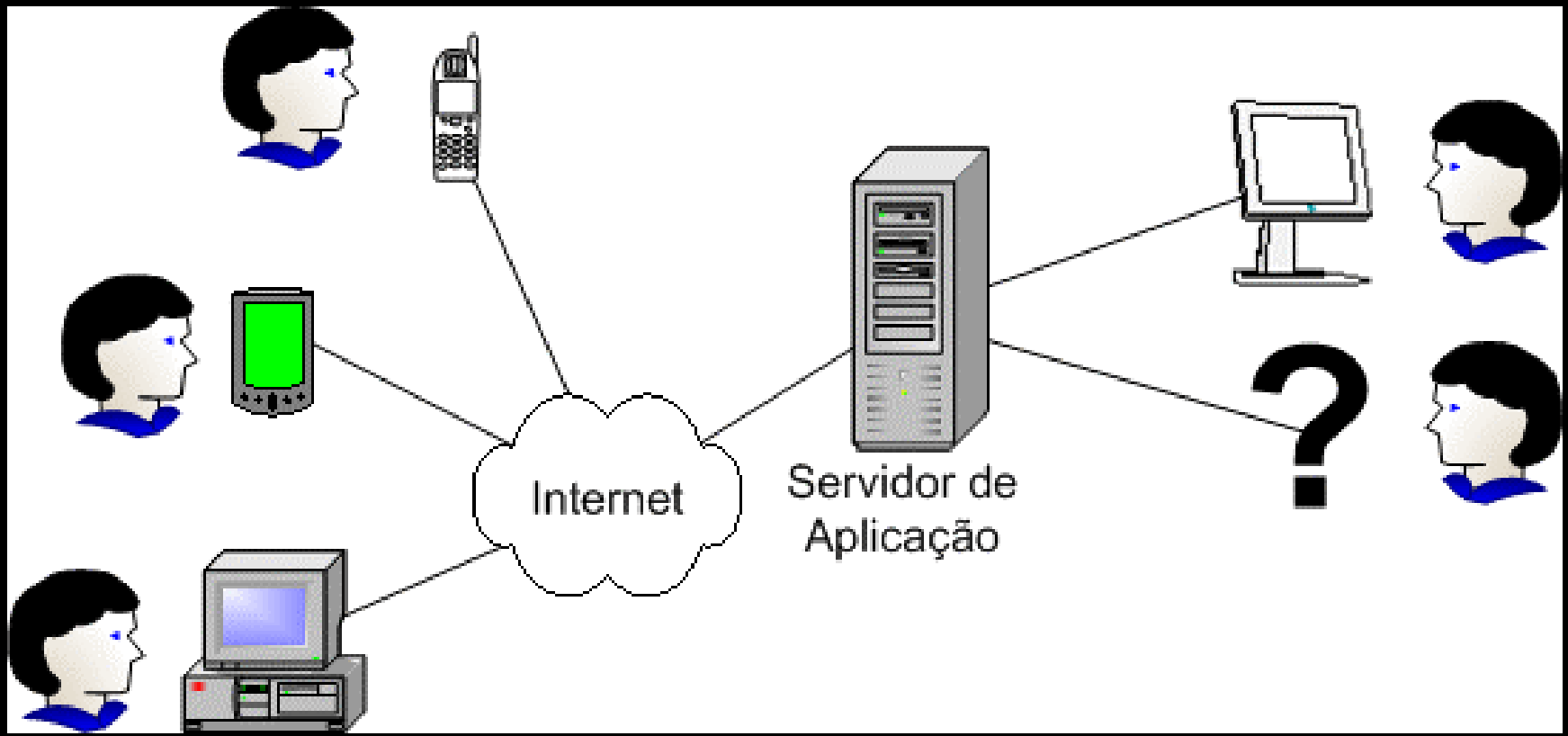


Objetivo

- Separar dados ou lógica de negócios (Model) da interface do usuário (View) e do fluxo da aplicação (Control)
- A idéia é permitir que uma mesma lógica de negócios possa ser acessada e visualizada através de várias interfaces.
- Na arquitetura MVC, a lógica de negócios (chamaremos de Modelo) não sabe de quantas nem quais interfaces com o usuário estão exibindo seu estado.

Objetivo

- **Com as diversas possibilidades de interfaces que conhecemos hoje, a MVC é uma ferramenta indispensável para desenvolvermos sistemas**



Características de MVC

- Separação entre dados (Model), apresentação (View) e Controlador (Controller) que gerencia as relações entre o modelo e a apresentação
- Separa a lógica da apresentação
- Maior reusabilidade
- Responsabilidades mais definidas
- Reduz o esforço na camada de apresentação

Características de MVC

- Metodologia ou design pattern que relaciona a interface do usuário e os seus dados
- Um dos primeiros patterns reconhecidos
- Utilizado inicialmente em Smalltalk-80
- Utilizado pelo GoF como exemplo de patterns

MVC – Principais Características

- A classe de controle conhece as classes de interface e domínio
- A classe de interface conhece as classe de controle e domínio

MVC – Principais Vantagens

- A classe de domínio é independente da interface do sistema
- A interface pode ser alterada sem afetar a classe de domínio
- A classe de domínio pode ser reutilizada em outras aplicações
- A seqüência de operações é encapsulada na classe de controle
- As dependências entre as classes do sistema são reduzidas

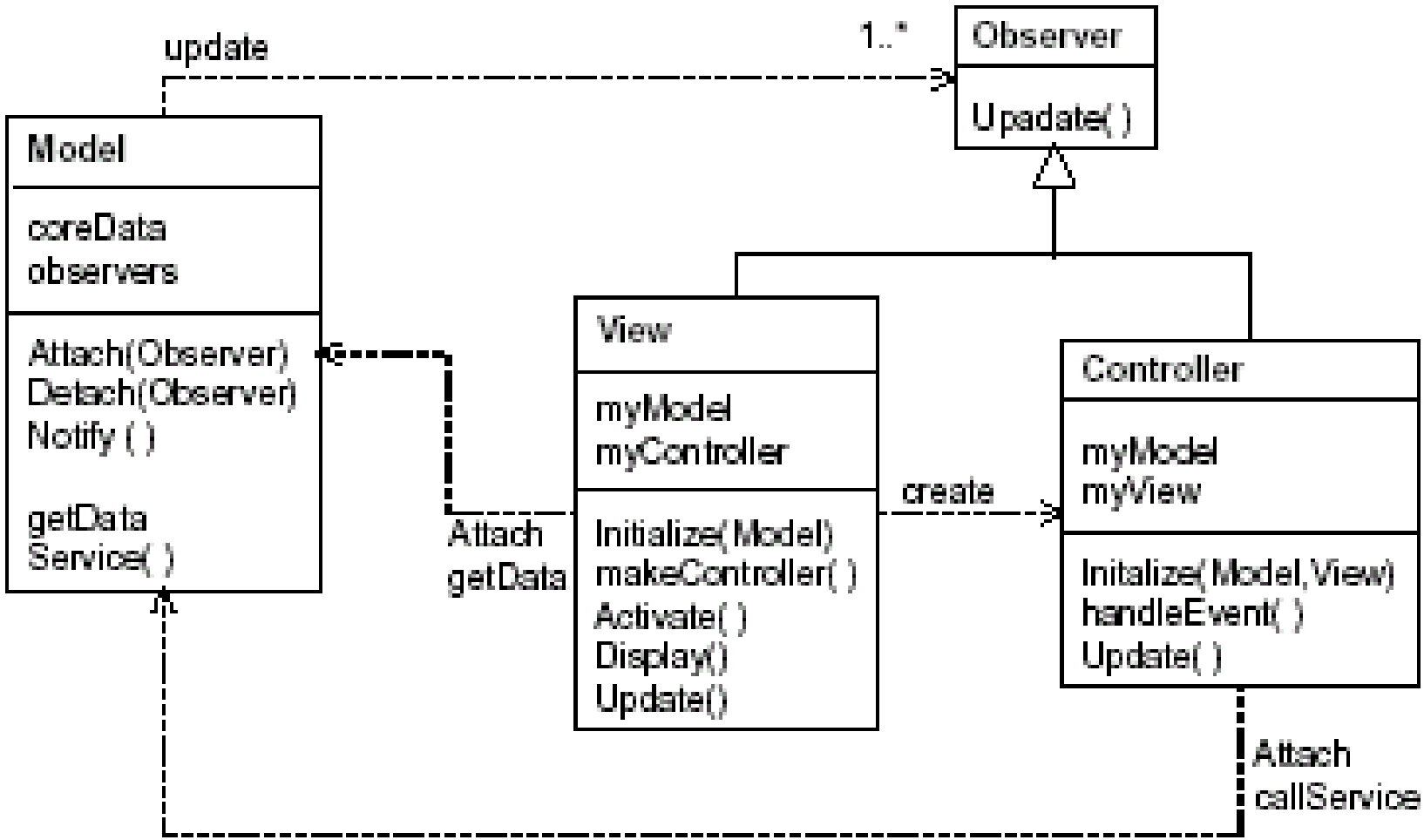
MVC

- O “conhecimento” entre duas classes é representado através de associações ou dependências
  - As associações são implementadas através de objetos atributos
  - Se a classe A conhece a classe B, a primeira terá um atributo da classe B
  - Através de seu atributo, a classe A pode chamar métodos para manipular a classe B

Variantes do modelo MVC

- Em uma primeira variante, a classe de interface não conhece a classe de domínio. Neste caso, a classe de controle é responsável por todo o trânsito de informações entre as duas classes.
- Em uma segunda variante (Document-View), as operações da classe de controle são embutidas na classe de interface.

Modelo MVC - ESTRUTURA



Problema a se aplicar o MVC

- Geralmente quando é desenvolvida uma aplicação para suportar apenas um tipo de cliente, se torna favorável o entrelace dos dados com a interface específica para a apresentação e controle dos dados.
- Esse procedimento é inadequado quando aplicado para sistemas que precisam suportar diversos tipos de usuários que exigem interfaces diversas.
- Se o código estiver dependente da interface e houver necessidade de mudar qualquer um dos dois $\Rightarrow$ ocorrerá uma duplicação dos esforços de implementação, bem como testes e manutenção.

Problema a se aplicar o MVC

- Interfaces com o usuário são sensíveis a mudanças:
  - O usuário está sempre querendo mudar funcionalidades e a interface das aplicações. Se o sistema não suporta estas mudanças, temos um grande problema!
- A aplicação pode ter que ser implementada em outra plataforma.

Problema a se aplicar o MVC

- A mesma aplicação possui diferentes requisitos dependendo do usuário:
  - um digitador prefere uma interface onde tudo pode ser feito através do teclado e visualizado como texto.
  - um gerente prefere uma interface através do mouse e de menus com visualização gráfica
- Neste contexto, se o código para a interface gráfica é muito acoplado ao código da aplicação, o desenvolvimento pode se tornar muito caro e difícil.

Quando usar o MVC?

- O MVC deve ser aplicado:
  - Quando os mesmos dados precisam ser acessados por diferentes interfaces ou atualizados por diferentes interações, pois separa os dados da apresentação e a apresentação da lógica do controle que usa essas funcionalidades.
  - Essa separação permite múltiplas visualizações compartilharem do mesmo modelo de dados.

Aplicação MVC para a Web

Desenvolvimento de aplicações Web

- Servlets, JavaBeans, Jsp -

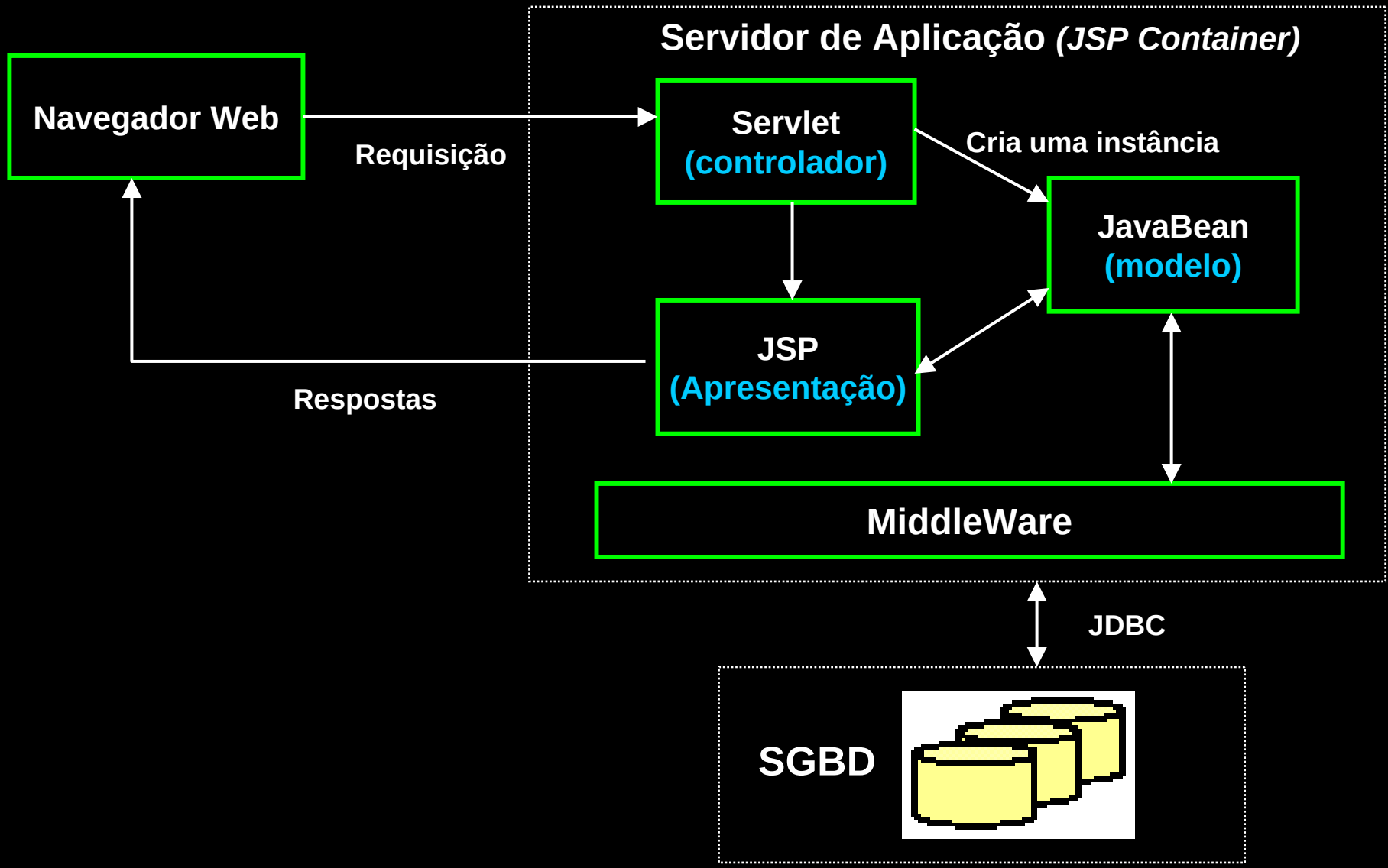
MVC – Integrando Servlets e JSP

- A arquitetura Model-View-Controller separa a geração do conteúdo de sua apresentação.
- Com isso, independente de como será apresentada esta informação, o que pode ser feito de diversas maneiras e mudar ao longo do tempo, o processamento (que engloba a lógica do negócio), será feito.
- Assim, se precisarmos mudar apresentação, o processamento será ou mesmo e vice-versa.

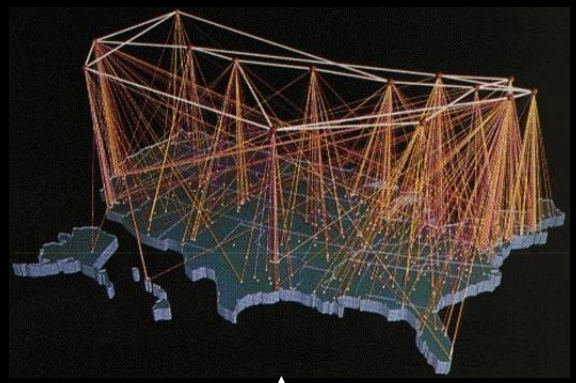
MVC – Integrando Servlets e JSP

- No caso do uso de Servlets e JSP mesclados, temos sempre a figura de um servlet controlador que despacha as solicitações HTTP para as páginas de apresentação jsp.
- Além disso, devemos ter classes auxiliares para:
  - Realizar conexões com o banco de dados e atender solicitações de acesso ao mesmo.
  - Criar classes de domínio que refletem os principais objetos a serem tratados no domínio

MVC – Projeto de uma arquitetura para aplicações Web



Fluxo



WebServer

Request

Controlador

Forward

Request

JavaServer Page

Response

Modelo

Visão

Model, View e Controller

Model, View e Controller

Servlet Controller

Visão

Model
EJB, JDO, JDBC

MVC – Projeto de uma arquitetura para aplicações Web

- **Servlets** - Atuam como controladores, recebendo as requisições dos usuários e acionando os beans e páginas JSP.
- **JavaBeans** - Atuam como modelo da solução, independente da requisição e da forma de apresentação. Comunicam-se com a camada intermediária que encapsula a lógica do problema.

MVC – Projeto de uma arquitetura para aplicações Web

- **JSP** - Atuam na camada de apresentação utilizando os javabeans para obtenção dos dados a serem exibidos, isolando-se assim de como os dados são obtidos.
- **Middleware** - Incorporam a lógica de acesso aos dados. Permitem isolar os outros módulos de problemas como estratégias de acesso aos dados e desempenho.

Vantagens da arquitetura

1) **Facilidade de manutenção:** a distribuição lógica das funções entre os módulos do sistema isola o impacto das modificações.
2) **Escalabilidade:** Modificações necessária para acompanhar o aumento da demanda de serviços (database pooling, clustering, etc) ficam concentradas na camada intermediária.
3) **Independente de fabricante:** usa apenas padrões abertos.

Exemplo Proposto

- Queremos implementar um sistema de votação, fazer uma enquete.
- A enquete deve permitir o voto dos usuários.
- Os votos são contabilizados e exibidos de duas formas:
  - Tela com votos absolutos, que mostra os totais de votos para cada opção;
  - Tela com percentual de votos.

Solução sem MVC

- Uma solução simples seria a criação de uma tela de votação (classe TelaVotacao) que armazena o estado da enquete e incrementa os votos à medida que os botões são clicados.
- Ao incrementar uma opção, as telas que exibem os resultados seriam atualizadas

Solução funciona, mas...

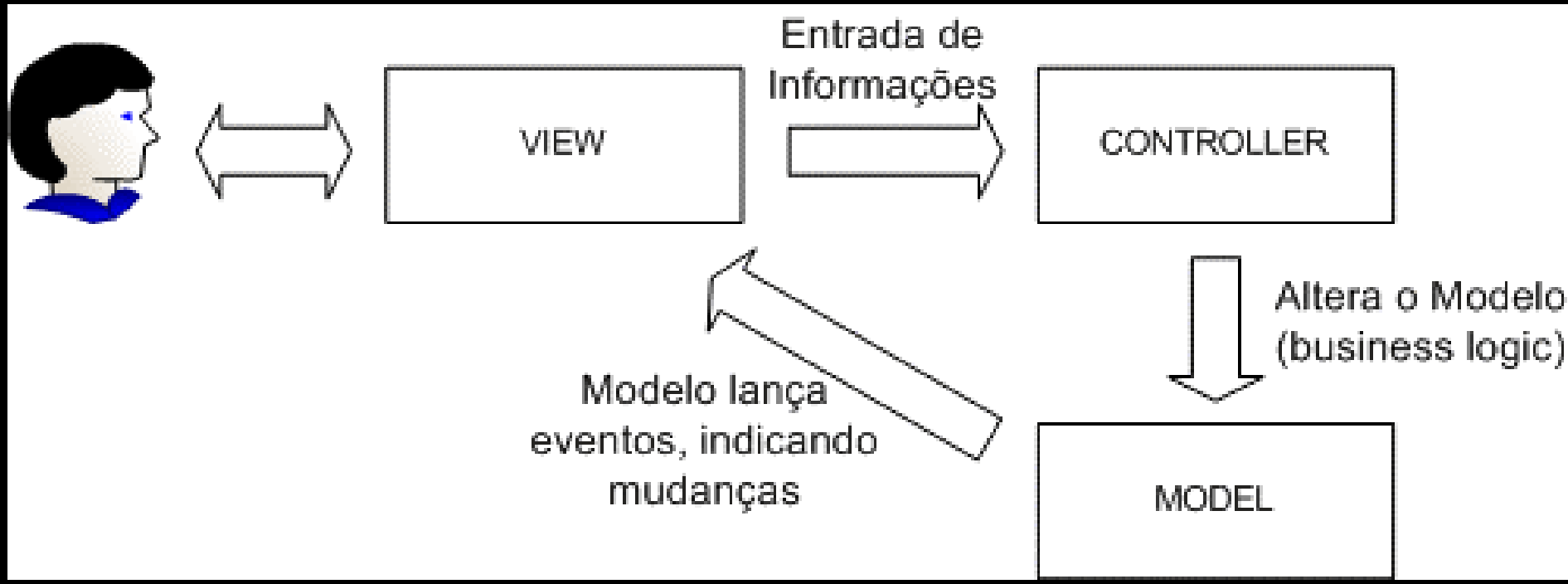
- E se, a partir de agora, o cliente começa a acessar a Internet e tem a idéia de colocar a enquete para os usuários da rede?
- E pior, o cliente comprou um celular com suporte a WAP e sacou que seria interessante ter a enquete também em WAP!
- Os problemas aumentarão à medida que nosso cliente comprar um celular com suporte a J2ME, um PALM, um relógio que acessa a Internet... Temos um problema.

Qual o problema da solução anterior?

- A classe TelaVotacao representa a interface de votação e ao mesmo tempo armazena o estado da enquete!
- Não conseguimos extrair o "business logic" da enquete pois ele está misturado com a interface!
- Premissa básica para uma boa solução:

"A LÓGICA DE NEGÓCIO NÃO DEVE CONHECER NADA SOBRE AS TELAS QUE EXIBEM SEU ESTADO!"

Solução com MVC



- **Modelo (MODEL): Lógica de negócio;**
- **Visão (VIEW): O usuário vê o estado do modelo e pode manipular a interface, para ativar a lógica de negócio;**
- **Controlador (CONTROLLER): Transforma eventos gerados pela interface em ações de negócio, alterando o modelo.**

Considerações Finais

- **MVC $\rightarrow$ 23 anos após sua criação ainda é um pattern aplicável**

- **Problemas que o MVC pode causar:**
  - Se tivermos muitas visões e o modelo for atualizado com muita freqüência, a performance do sistema pode ser abalada.

Considerações Finais

- **Problemas que o MVC pode causar:**
  - Se o padrão não for implementado com cuidado, podemos ter casos como o envio de atualizações para visões que estão minimizadas ou fora do campo de visão do usuário.
  - Ineficiência: uma visão pode ter que fazer inúmeras chamadas ao modelo, dependendo de sua interface.

Referências Bibliográficas

- Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns - Elements of Reusable Object-Oriented Software*. Reading-MA, Addison-Wesley, 1995.

- D. Alur, J. Crupi, D. Malks, *Core J2EE Patterns As melhores práticas e estratégias de design, Editora Campus, 2002.*

- Kerth, Norman L.; Cunningham, Ward. *Using Patterns to Improve Our Architectural Vision*, IEEE Software, pp. 53-59, janeiro, 1997.

Referências Bibliográficas

- http://www.burridge.net/jsp/jspinfo.html Web Development with JSP: JSP, Java Servlet, and Java Bean Information

- http://java.sun.com/ Página da Sun com informações, tutoriais e produtos Java.

- http://www.enode.com/x/markup/tutorial/mvc.html Tutorial MVC da empresa Markup Language contendo exemplificações.