



Engenharia de Software III

IES-300

Tecnologia em Análise e Desenvolvimento de Sistemas

Prof. Me. Álvaro d'Arce
alvaro@darce.com.br

Tópicos

- **Arquitetura de Software**

- Modelo 3 Camadas

- Modelo MVC

Projeto de Software

Considerações

- Na fase de **projeto**:
 - Há a preocupação com a **arquitetura** da aplicação, dando-se valor à **tecnologia**
 - diferente da fase de análise, onde é esboçado o problema a ser resolvido;
 - Define-se a plataforma de desenvolvimento e **como os componentes** do sistema serão **organizados**
 - mas os requisitos continuam sendo importantes e influenciam na **arquitetura**;
 - A **arquitetura** define os **elementos** do software e como eles **interagem** entre si.

Arquitetura de Software

*“Arquitetura é um conjunto de **estruturação de princípios** que possibilita um sistema de ser **composto** por outro **conjunto mais simples**, onde cada um possui seu próprio contexto independente, não podendo ser incompatível com todo o sistema.”*

[Sun Microsystem, Inc.]

Arquitetura de Software

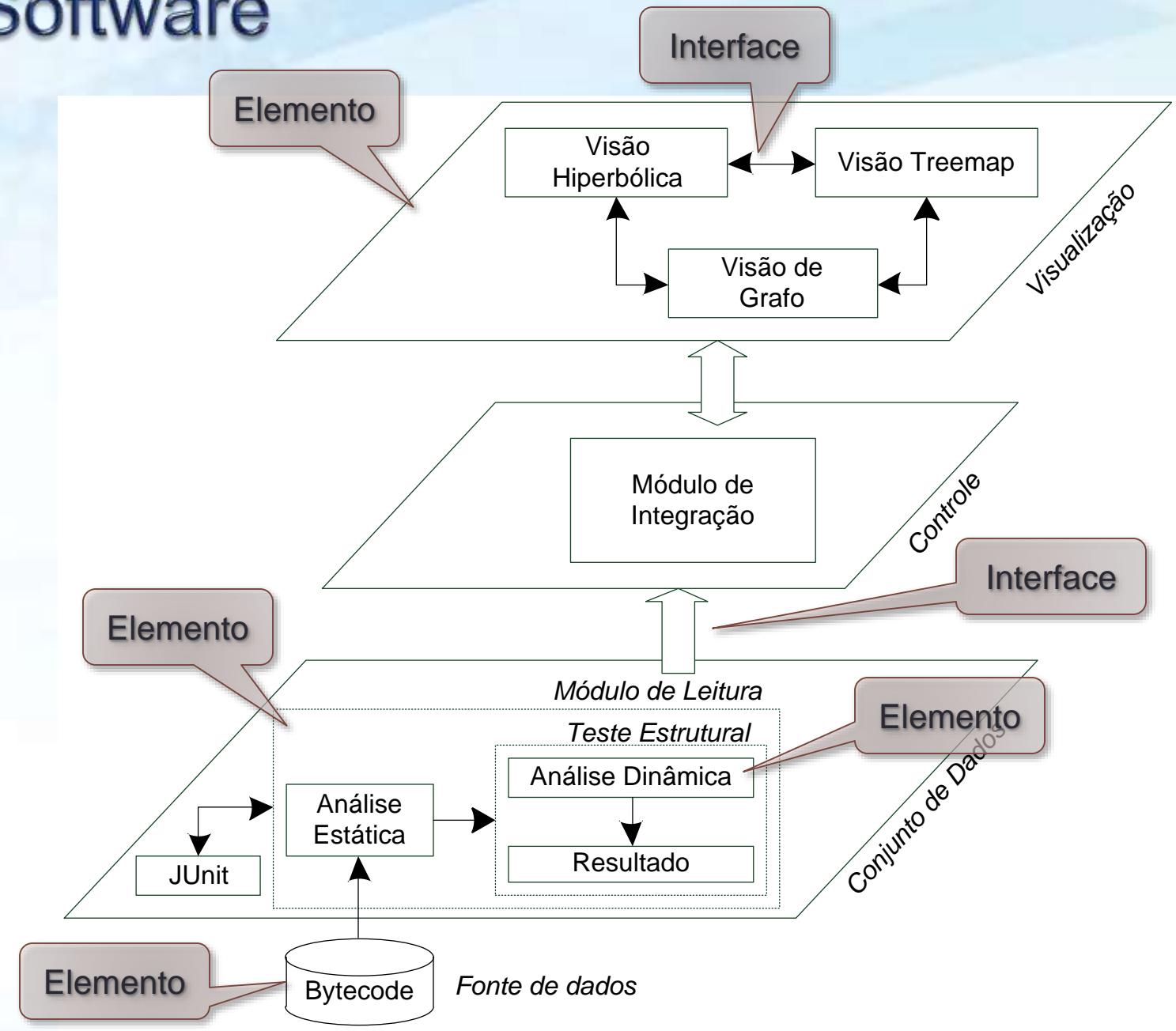
Introdução

- Conjunto de **decisões** significativas sobre:
 - a **organização** de um software
 - a seleção dos **elementos estruturais** e suas **interfaces** pelas quais o sistema é composto, juntamente com seu comportamento (especificado nas colaborações entre esses elementos)
 - a composição desses elementos estruturais e comportamentais
 - e o **estilo arquitetural** que dirige essa organização – esses elementos e suas interfaces, suas colaborações e sua composição

Arquitetura de Software

Exemplo

- Arquitetura lógica elaborada para uma ferramenta de visualização de software:



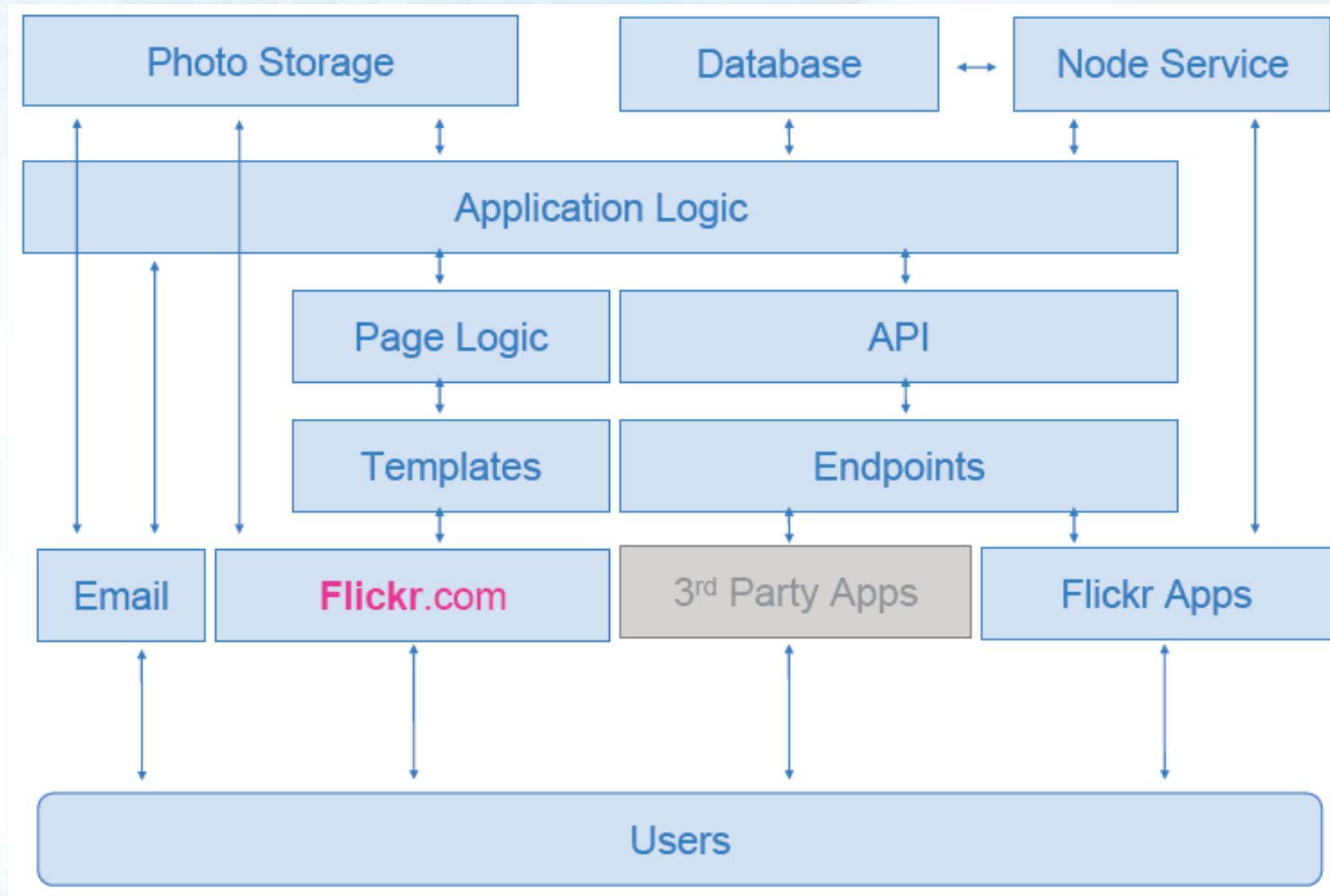
Arquitetura de Software

Arquitetura Lógica

- Representa a organização do software em **módulos lógicos** e a especificação de interfaces e dependências entre os módulos.
- Organiza o software em agrupamentos lógicos (ou **pacotes**)
 - Organização em larga escala das classes de software em pacotes, subsistemas e camadas
 - *Packages* no Java ou *NameSpaces* no C#
- É chamada de **arquitetura lógica** porque não há decisão sobre como esses elementos são implantados pelos diferentes processos do Sistema Operacional ou pelos computadores físicos em uma rede
 - tais decisões posteriores são parte de uma **Arquitetura de Implantação** ou **Arquitetura Física**

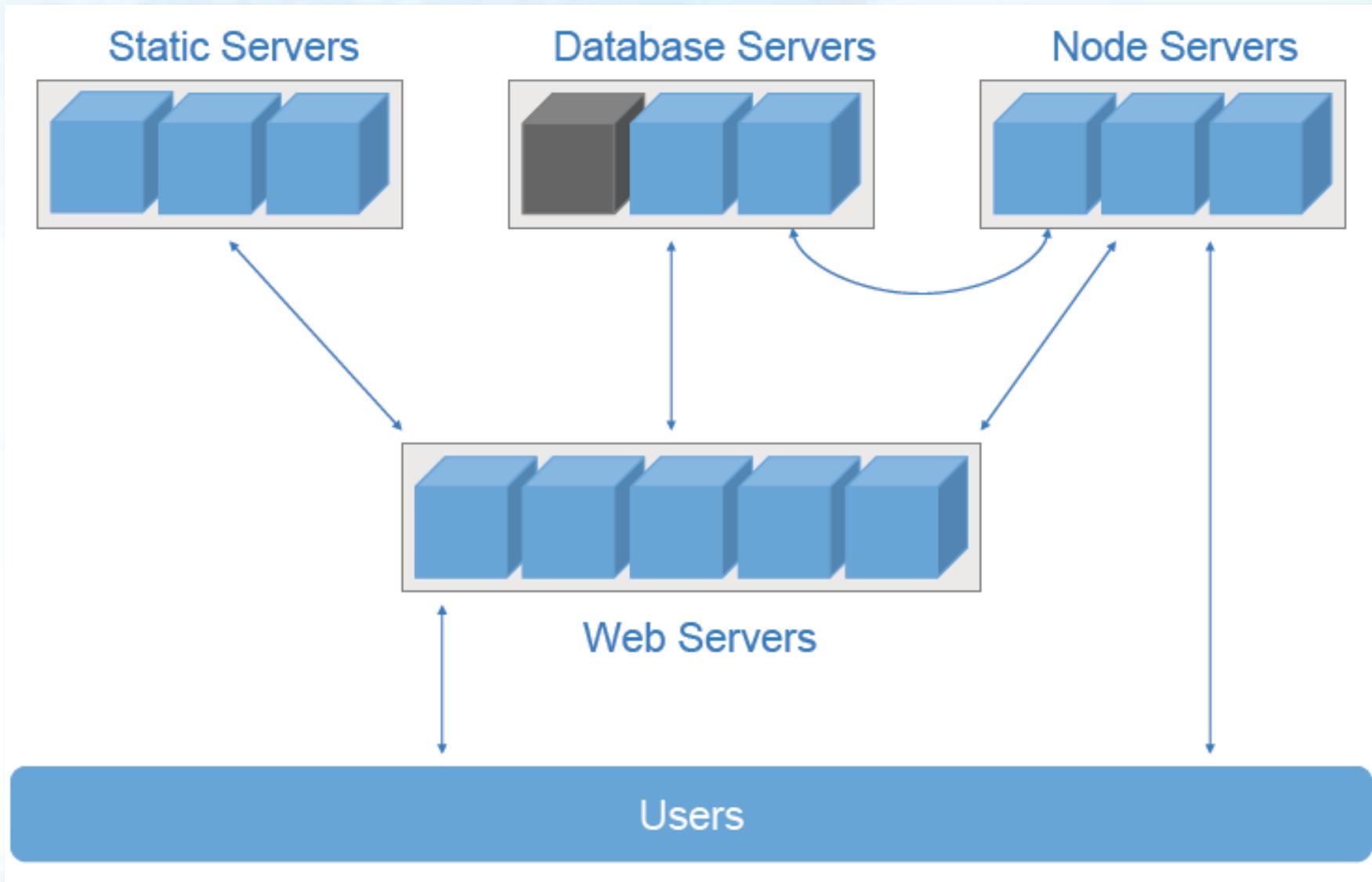
Arquitetura de Software

Exemplo: Arquitetura Lógica do Flickr



Arquitetura de Software

Exemplo: Arquitetura de Implantação do Flickr



“O **desenvolvedor** está concentrado no que ocorre quando **um** usuário pressiona um botão, enquanto um **arquiteto** está concentrado no que ocorre quando **dez mil** usuários pressionam um botão.”

[Paul R. Allen, Joseph J. Bambara – SCEA]

Arquitetura Lógica

Layers

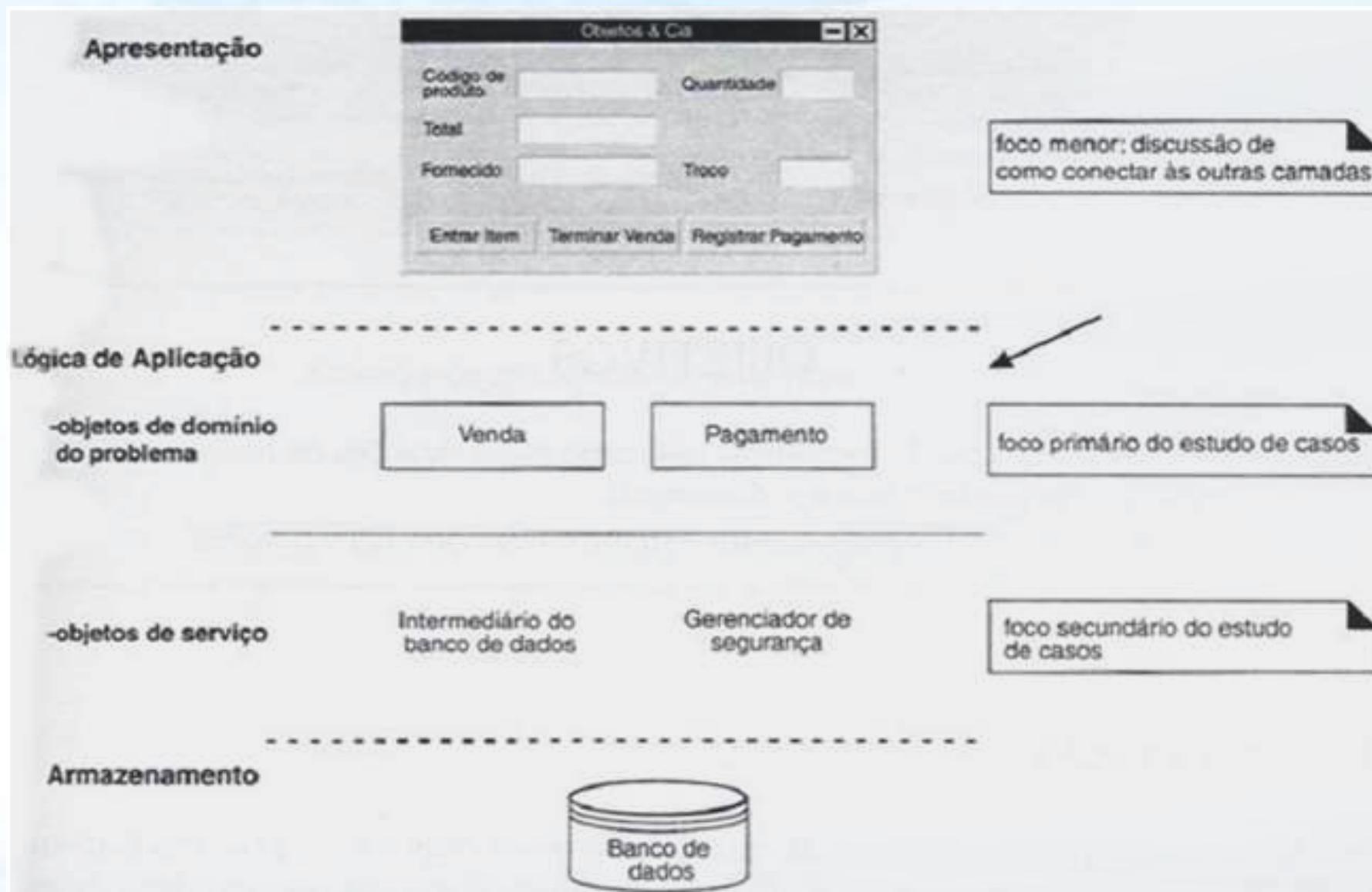
- É muito comum projetar a **arquitetura lógica** do sistema em camadas (layers):
 - Agrupamento de **classes**, **pacotes** ou **subsistemas** que tem responsabilidade **coesa** sobre um tópico importante do sistema
- Usar camadas ajuda a tratar **problemas**, tais como:
 - A lógica da aplicação está **entrelaçada** com a interface do usuário, de modo que **não pode ser reusada** com uma interface diferente
 - Modificações do código fonte se **espalham** por todo o sistema
 - muitas partes do sistema acabam ficando **altamente acopladas**
- Sendo assim, dividir um sistema em camadas coopera para:
 - Aumentar modularidade
 - Diminuir dependências
 - Facilitar possível troca de camadas

Layers – Exemplo: Sistema TPV [1/2]

- O TPV é um sistema de informação típico e pode ser visualizado em **várias camadas**:
 - **Apresentação** (interface): interface gráfica, janelas, ...
 - **Lógica da aplicação** (ou do negócio) – **objetos do domínio do problema**: representam os conceitos do domínio do problema que atendem aos requisitos do sistema.
 - Ex: objeto Venda.
 - **Lógica da aplicação – objetos de serviço**: objetos/funções que não fazem parte do domínio do problema mas oferecem serviços de infra-estrutura.
 - Ex: interface com o banco de dados (SGBD).
 - **Armazenamento**: um mecanismo de armazenamento permanente, como uma base dados relacional, orientada a objetos, objeto-relacional, baseada em grafos, ou um diretório de arquivos.

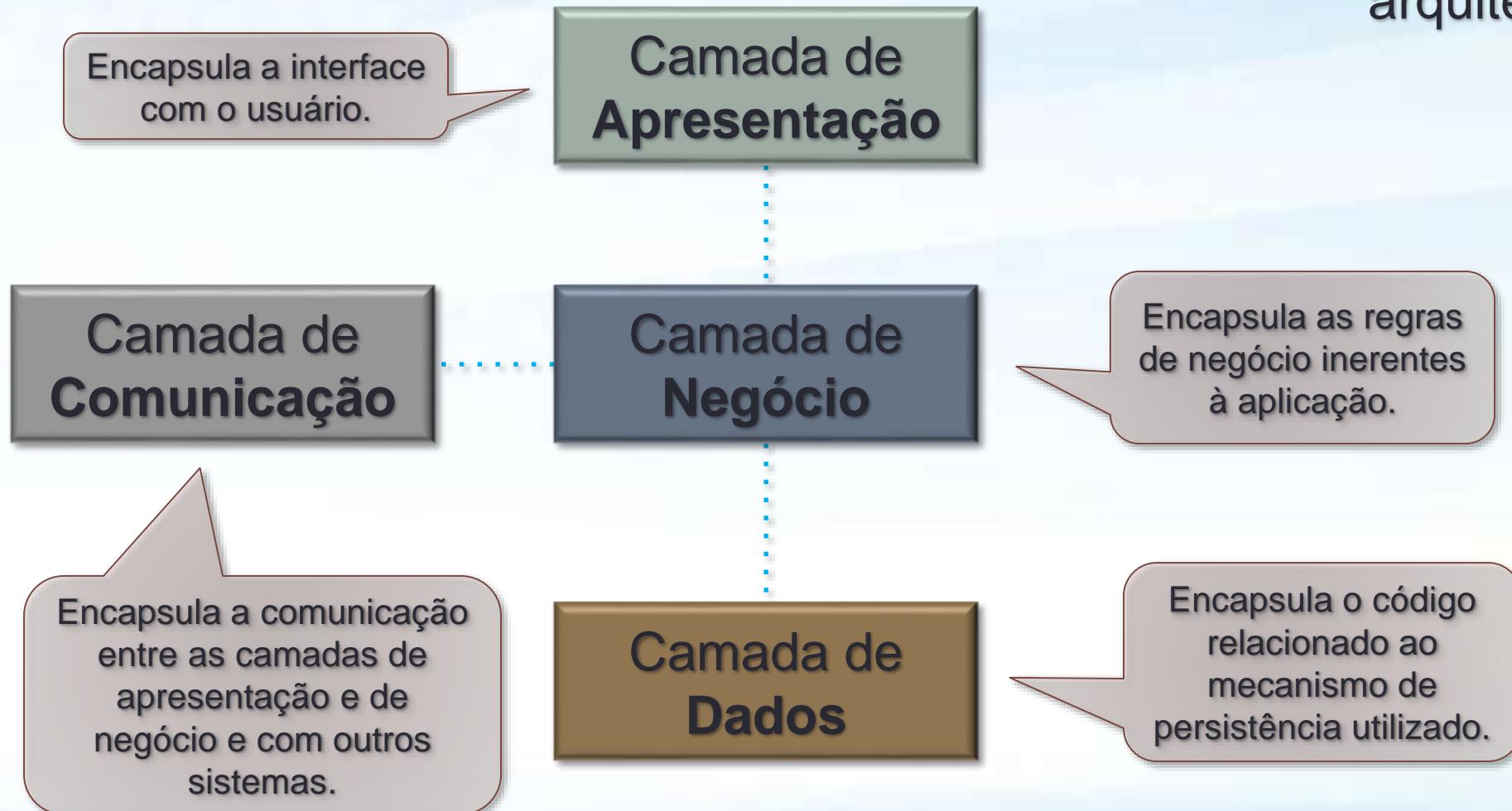
Arquitetura Lógica

Layers – Exemplo: Sistema TPV [2/2]



Arquitetura Lógica

Layers – Exemplo [1/2]



- Exemplo de uma arquitetura bem comum

Arquitetura Lógica

Layers – Exemplo [2/2]

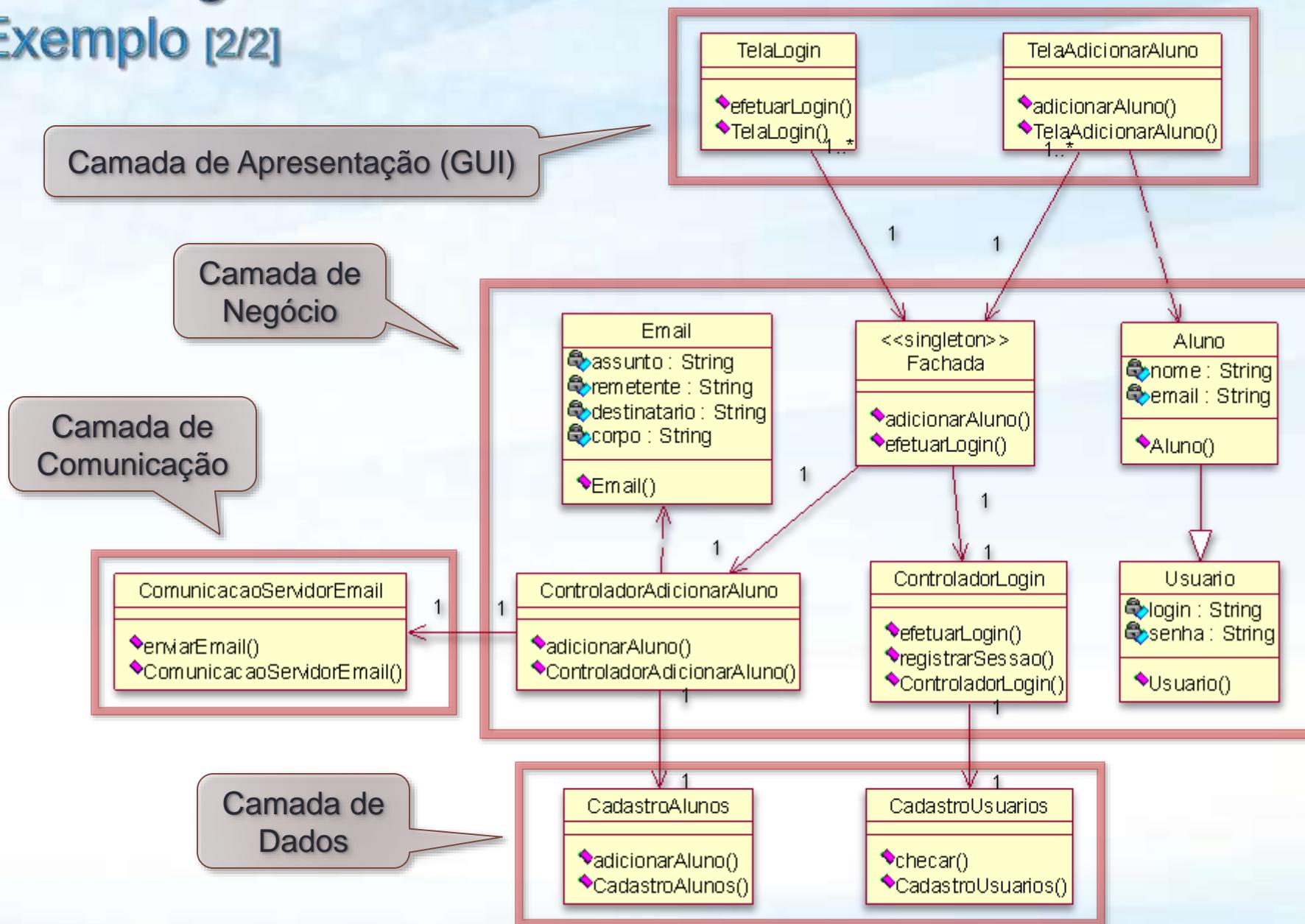


Diagrama de Pacotes

Introdução

- Um **Diagramas de Pacotes** é utilizado para modelar a **arquitetura lógica** de um sistema de software
 - Também é chamado de **Diagrama de Pacotes da Arquitetura Lógica**
- Descreve os **pacotes ou pedaços** do sistema (como o sistema é dividido em **agrupamentos lógicos**) e mostra as **dependências** entre os pacotes
 - Um pacote nada mais é do que um **agrupamento de classes**
- Possíveis critérios de agrupamento:
 - **Áreas funcionais** (lógica do sistema) e **Camadas**
 - Critérios escolhidos devem **minimizar a dependência** entre os pacotes

Diagrama de Pacotes

Notação e Exemplo [1/3]

- **Exemplo 1:** Divisão de um sistema de biblioteca em **áreas funcionais**.

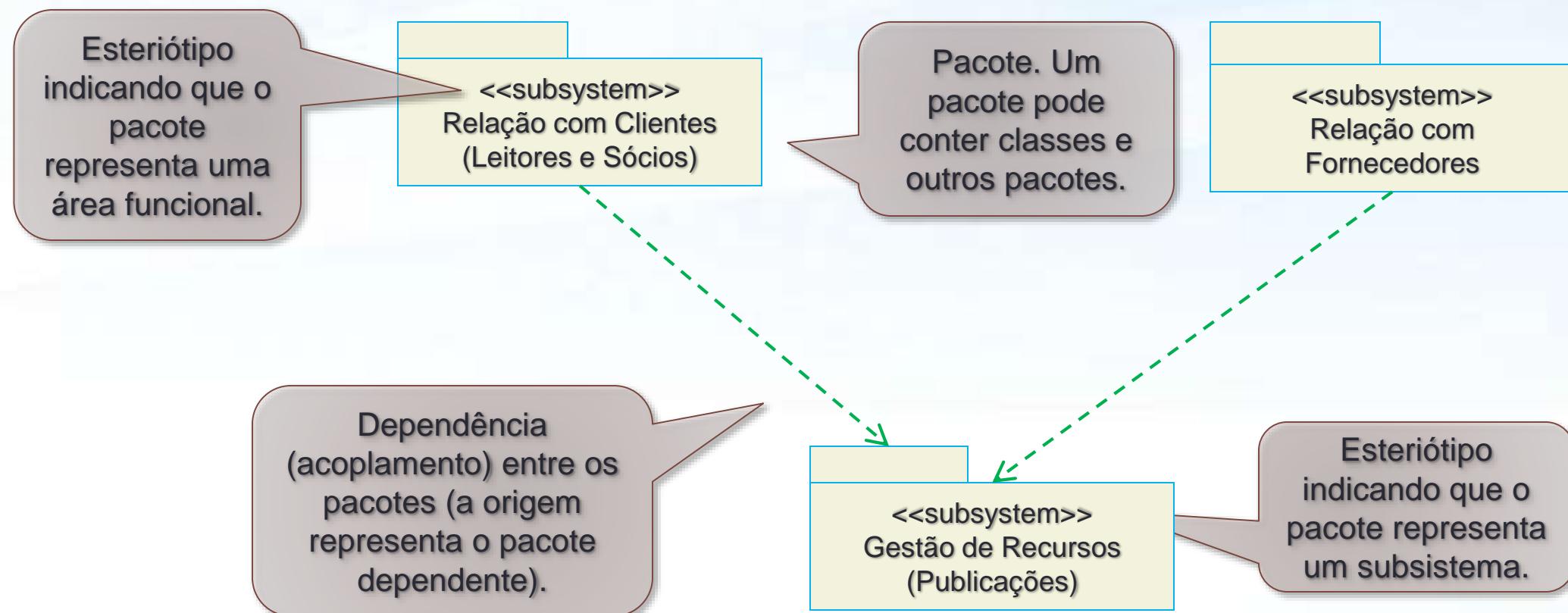


Diagrama de Pacotes

Notação e Exemplo [2/3]

- **Exemplo 2:** Divisão de um sistema de biblioteca em **camadas técnicas** no Modelo 3 Camadas.

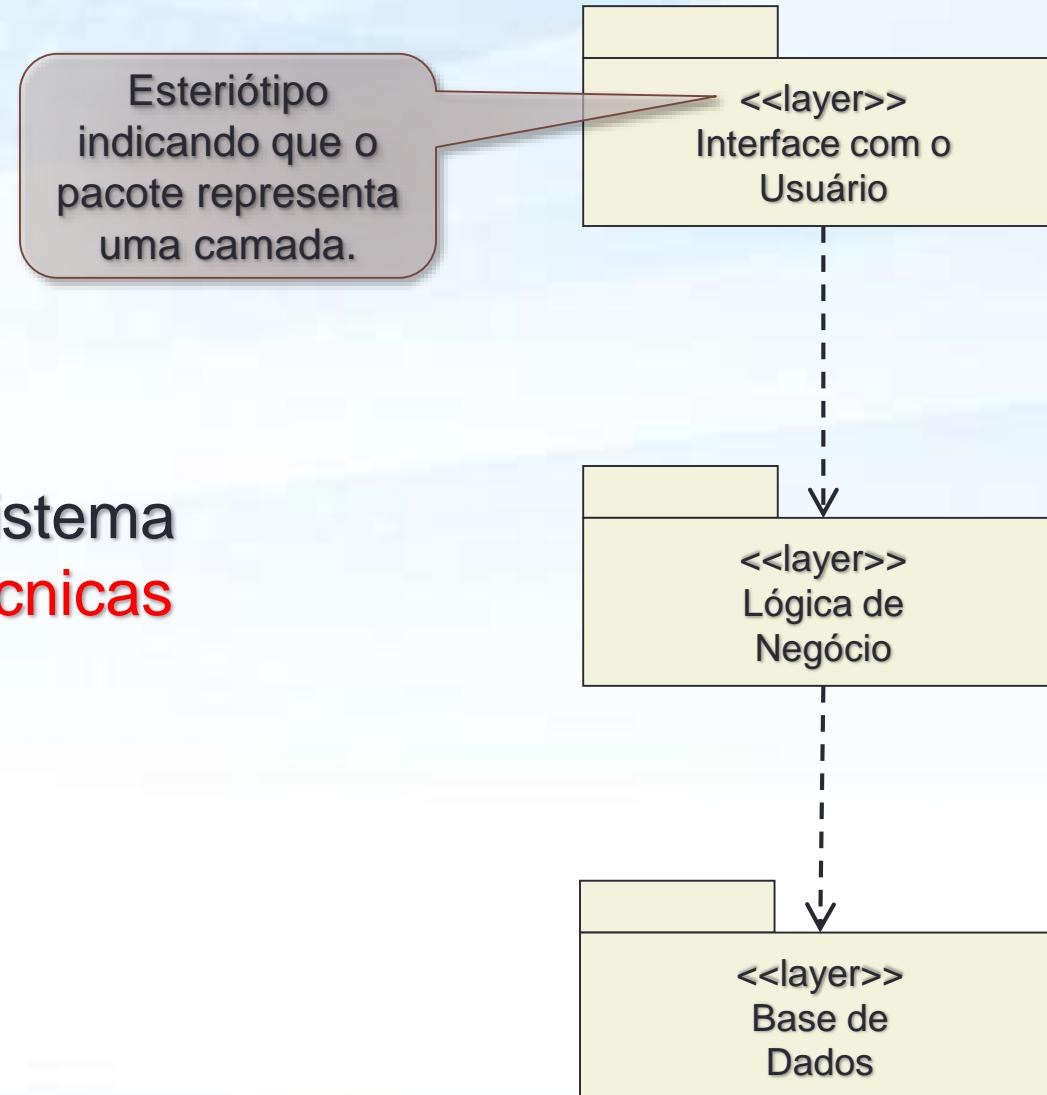


Diagrama de Pacotes

Notação e Exemplo [3/3]

- **Exemplo 3:** Divisão de um sistema em camadas em uma arquitetura comumente utilizada.

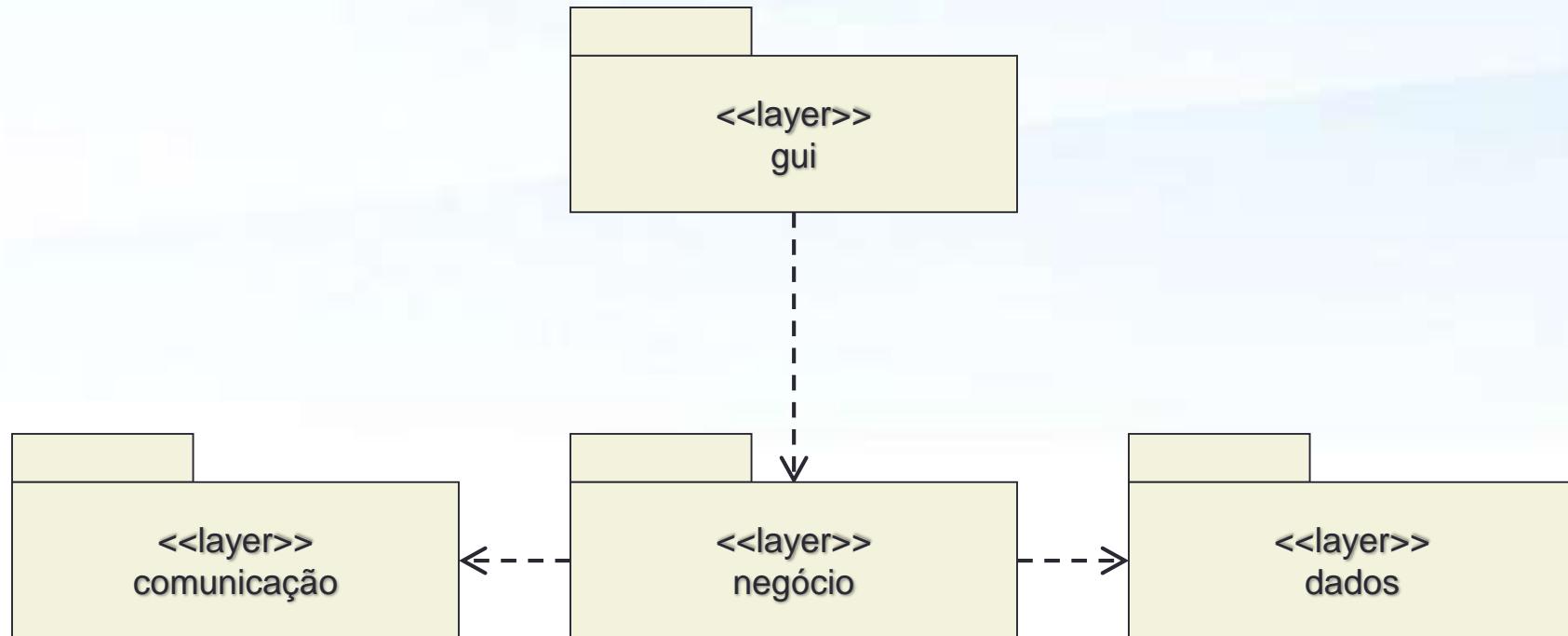
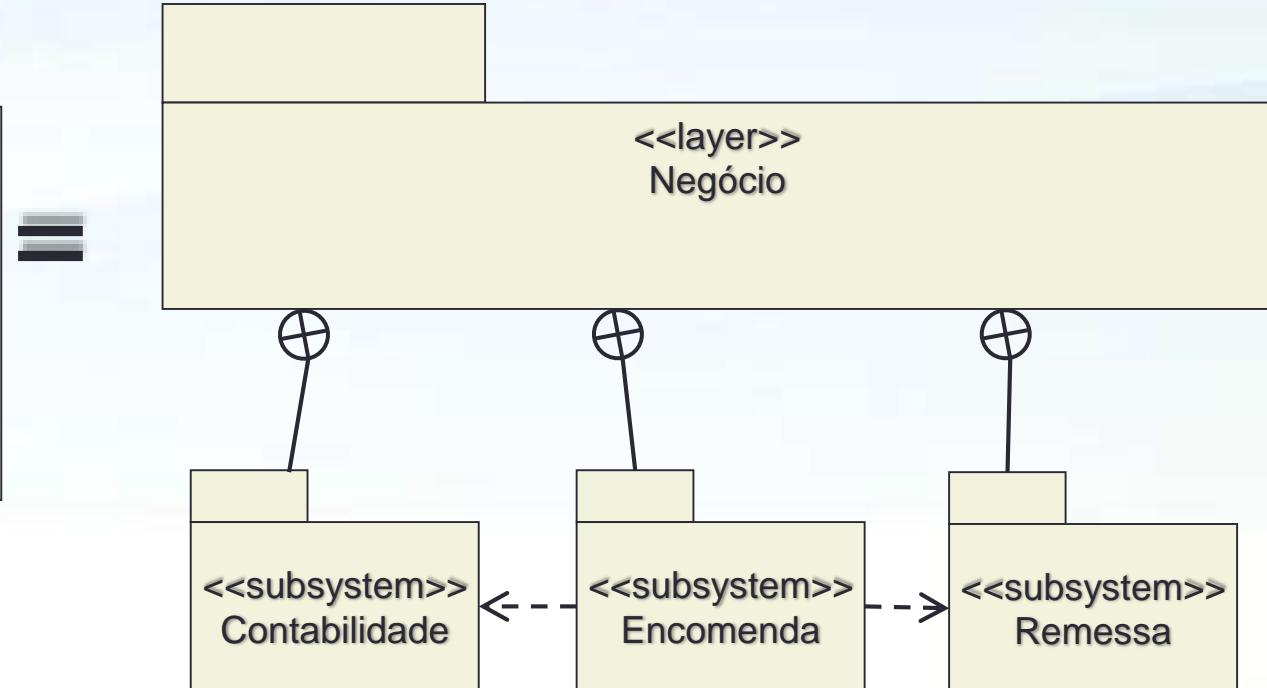
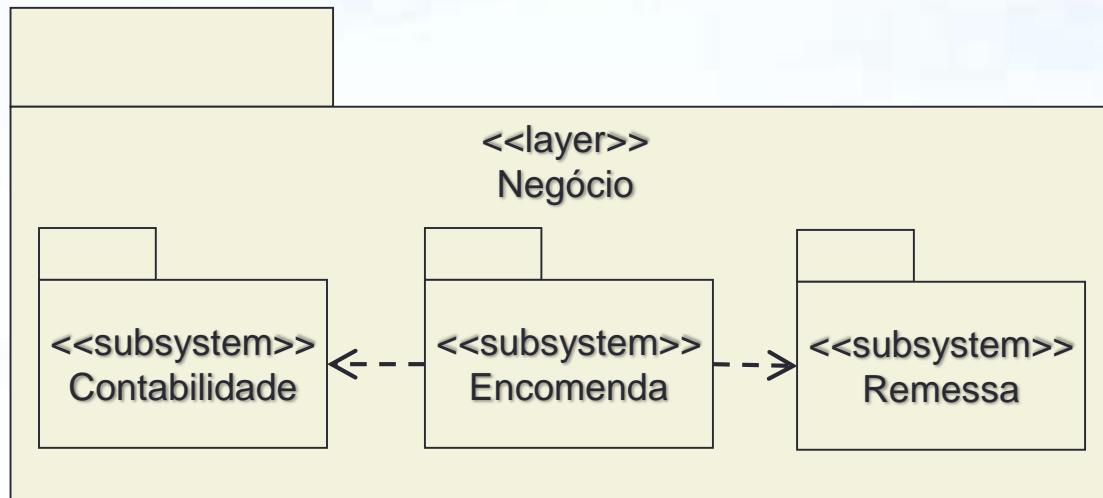


Diagrama de Pacotes

Pacotes contendo outros pacotes

- Estes 2 diagramas representam a mesma coisa:



Estilos Arquiteturais

- Existem diversos modelos ou padrões predefinidos para elaborar a arquitetura lógica de um sistema:
 - Cliente-Servidor
 - Pipe-filter
 - 3 Layers (3 Camadas)
 - 3 Tiers (3 Camadas)
 - MVC
 - MVP
 - MVVM
 - etc.

Tópicos

- Arquitetura de Software
- **Modelo 3 Camadas**
- Modelo MVC

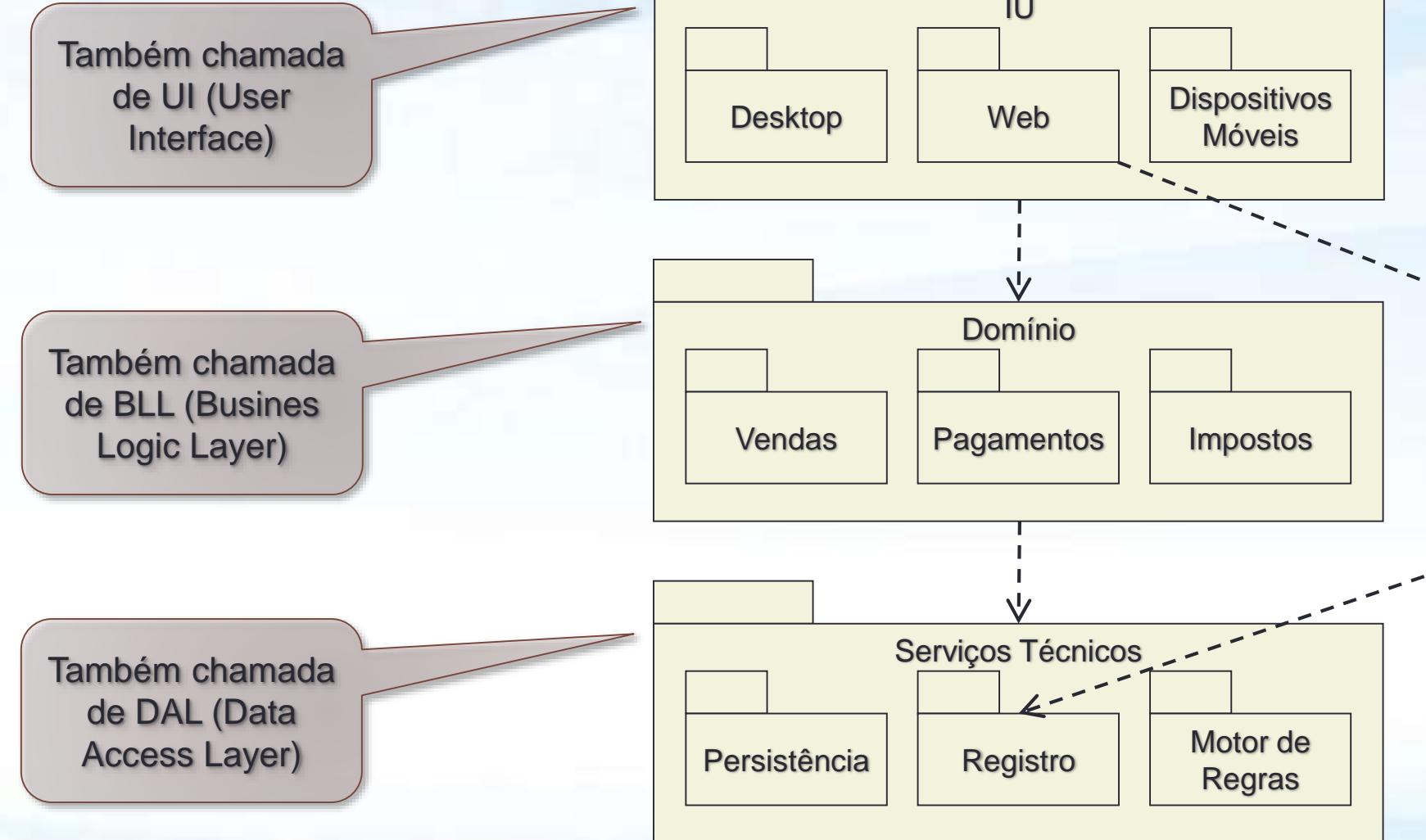
Estilos Arquiteturais

Modelo 3 Camadas (3 Layers)

- Organiza logicamente um sistema nas seguintes camadas:
 - **Interface com o Usuário (IU ou Apresentação)**
 - **Lógica da Aplicação e Objetos do Domínio** (Funcionalidades Principais): objetos de software representando conceitos do domínio (ex., uma classe de software Venda) que satisfazem requisitos da aplicação, como calcular um total de venda.
 - **Serviços Técnicos** (Funcionalidades Secundárias): objetos de propósito geral e subsistemas que fornecem serviços técnicos de apoio, como interfaceamento com um banco de dados ou registro de erros
 - Esses serviços, geralmente, são independentes da aplicação e reusáveis entre diversos sistemas
 - As camadas são organizadas de maneira **que as “mais altas”** (como a camada de IU) **solicitem serviços das “mais baixas”**

Estilos Arquiteturais

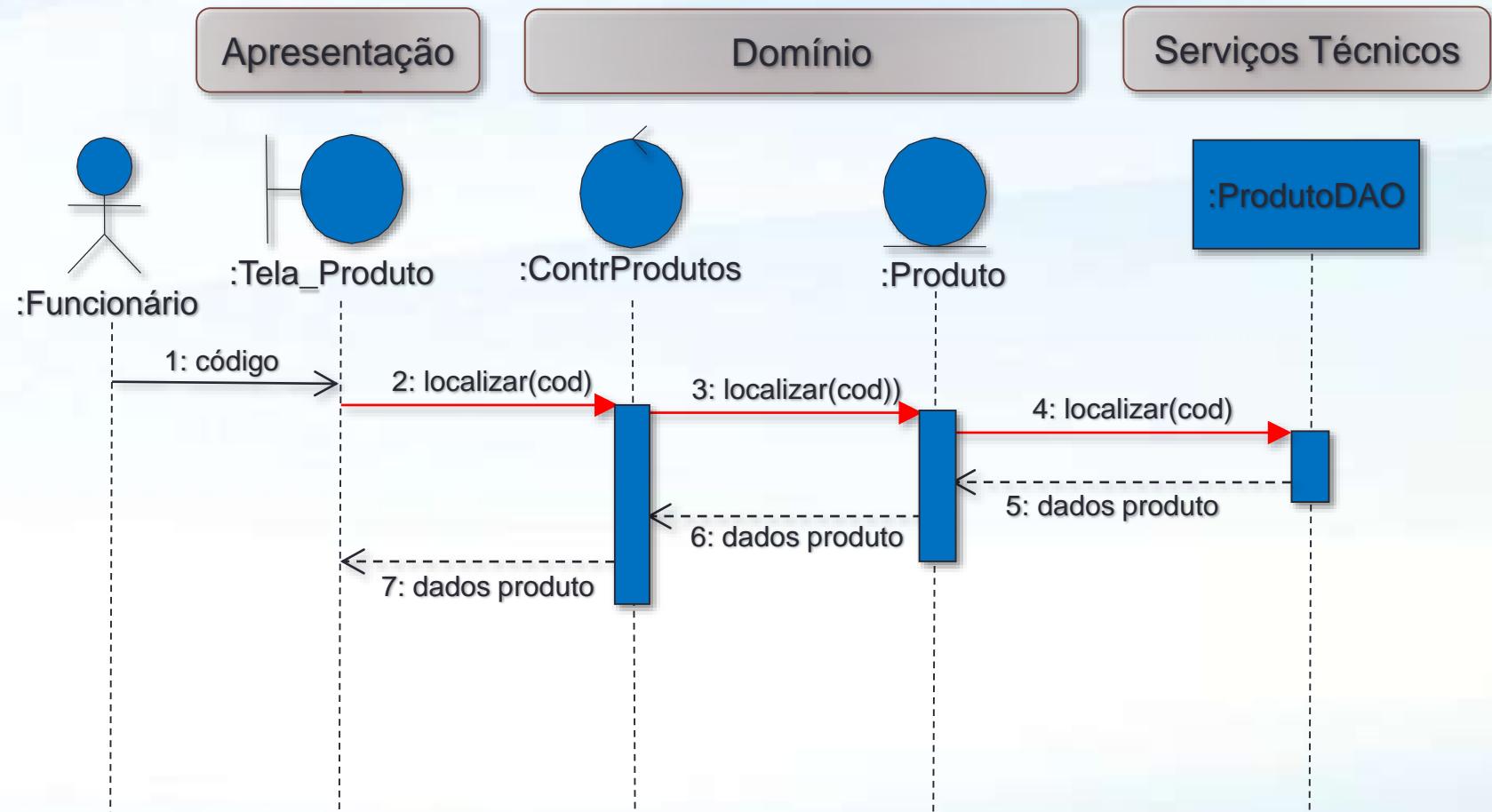
Modelo 3 Camadas – Exemplo



Estilos Arquiteturais

Modelo 3 Camadas – Exemplo: Localizando um Produto

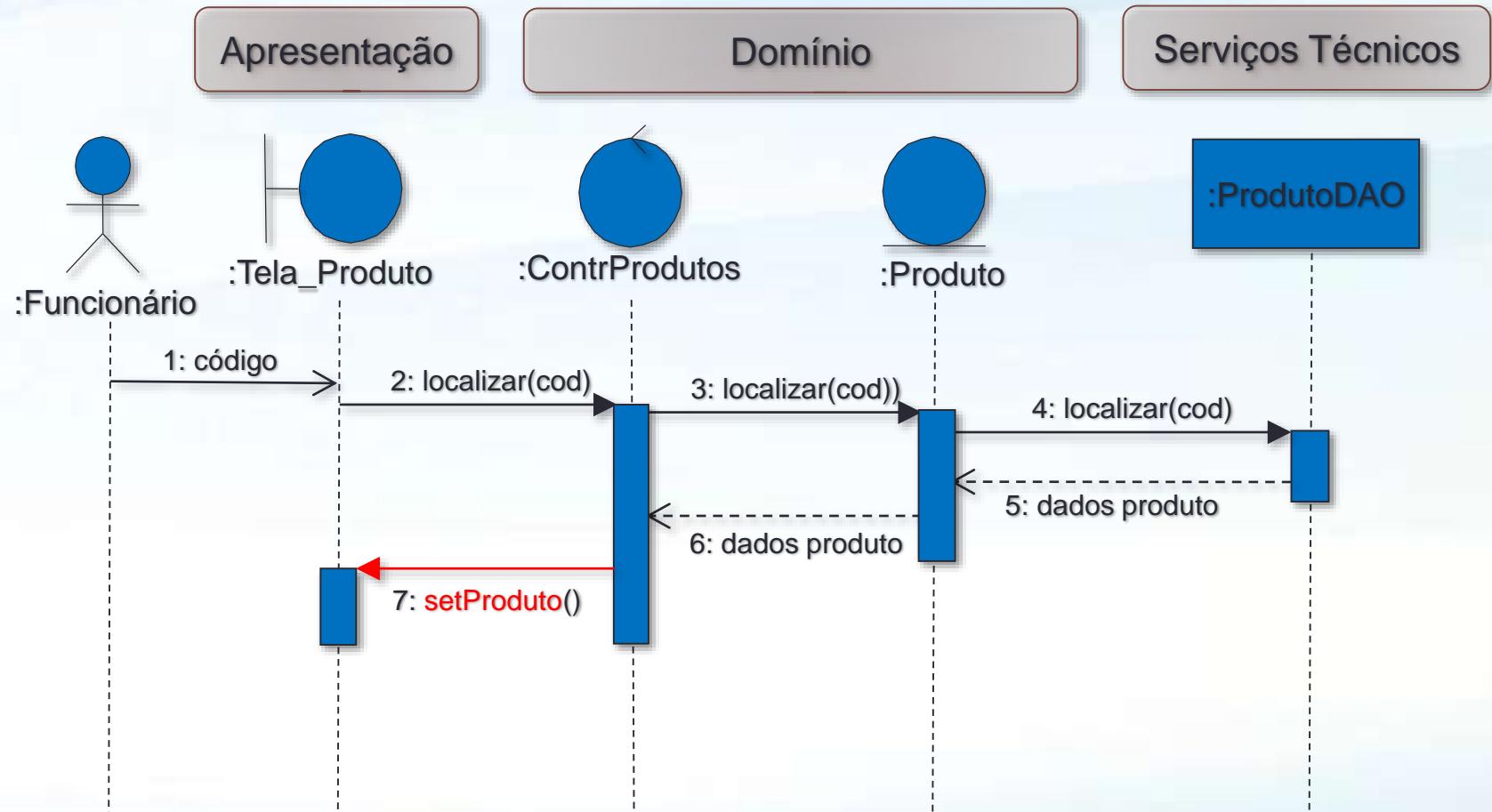
- As camadas mais altas **estão chamando** as camadas mais baixas.



Estilos Arquiteturais

Modelo 3 Camadas – Exemplo que não é 3 camadas

- Uma camada mais baixa **está chamando** uma camada mais alta.



Estilos Arquiteturais

Layers x Tiers

- Os termos *layer* e *tier* são ambos traduzidos como *camada*, mas há uma diferença entre eles:
 - **Layer** refere-se a uma **camada lógica**, um agrupamento de componentes dentro da lógica da aplicação (**arquitetura lógica**). Ex.:
 - Modelo 3 Layers (ou 3 Camadas).
 - **Tier** refere-se a uma **camada física**, uma distribuição física dos componentes do sistema (**arquitetura de implantação**). Ex.:
 - **Modelo 2 Tier**: Organiza o sistema em 2 camadas físicas. Ex.:
 - A aplicação encontra-se distribuída parte nos computadores clientes (**camada cliente**) e parte no servidor de banco de dados (**camada servidor**).
 - **Modelo 3 Tier**: Organiza o sistema em 3 camadas físicas. Ex.:
 - A aplicação encontra-se distribuída parte nos computadores clientes (**camada cliente**), parte no servidor de aplicação (camada de aplicação) e parte no servidor de banco de dados (**camada servidor**).

Tópicos

- Arquitetura de Software
- **Modelo MVC**

Modelo MVC (Model-View-Controller)

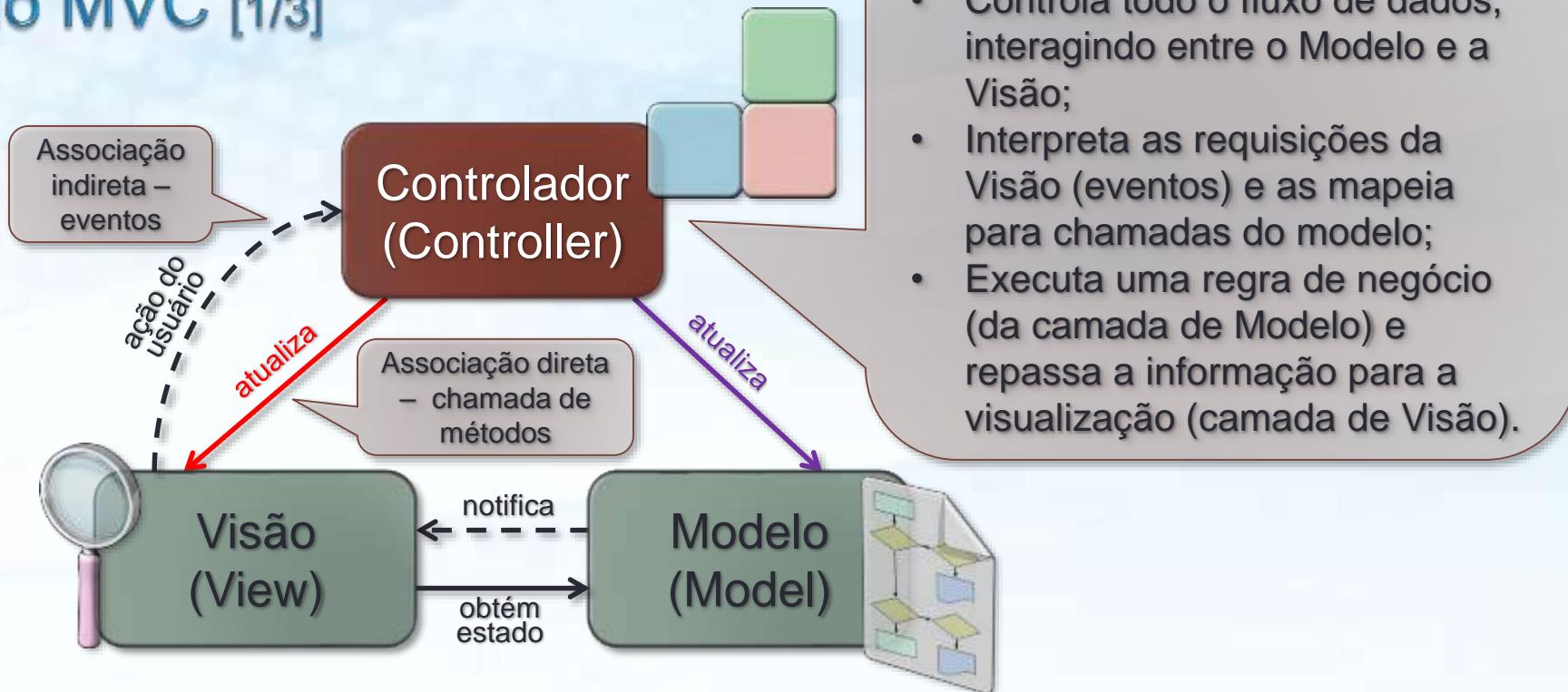
Introdução

- Padrão de arquitetura (**estilo arquitetural**) criado na década de 70 por Trygve M. H. Reenskaug, como uma solução para oferecer **maior controle sobre os dados** de um sistema;
- Fornece uma **separação das responsabilidades** na aplicação (**visão e regras de negócio**), facilitando a manutenção e a reutilização;
- Utilizado inicialmente na linguagem *Smalltalk*

Modelo MVC Tradicional

Arquitetura do MVC [1/3]

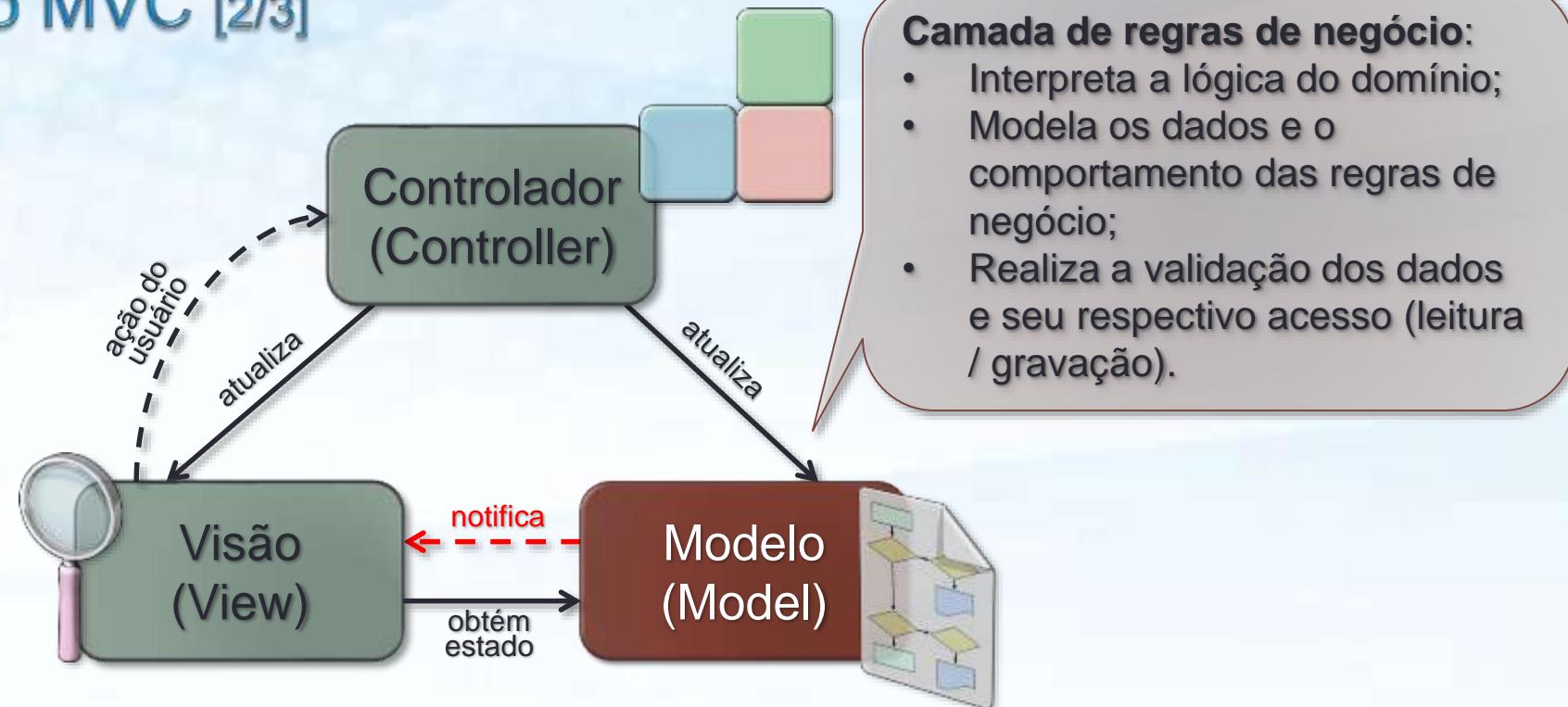
30



- Um **controlador** pode **enviar comandos para sua visão** associada para alterar a apresentação da visão do modelo
 - Ex.: percorrendo um documento;
- Também pode **enviar comandos para o modelo** para atualizar o estado do deste
 - Ex.: editando um documento;

Modelo MVC Tradicional

Arquitetura do MVC [2/3]



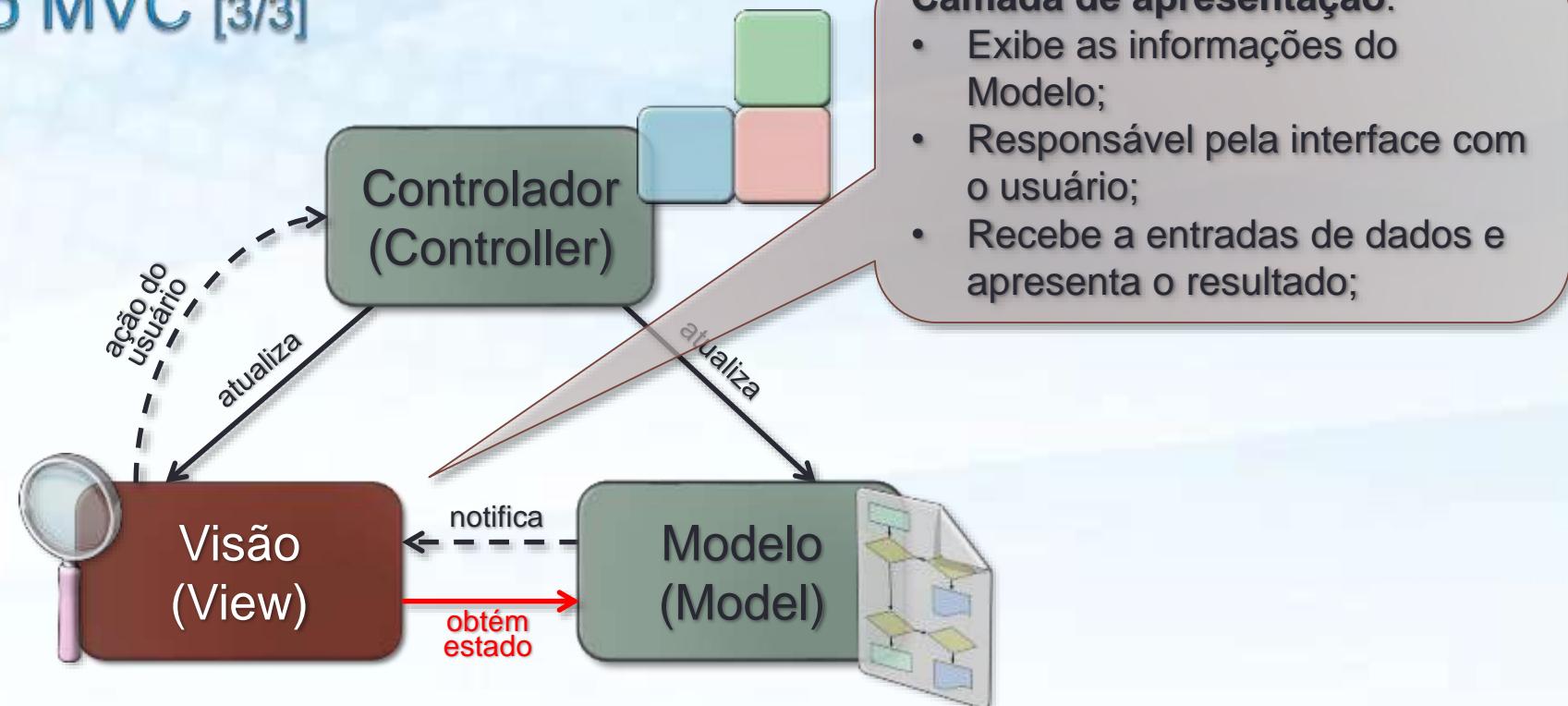
- Um modelo **notifica suas visões** quando há uma mudança em seu estado.
 - Esta notificação permite que as visões produzam saídas atualizadas e que os controladores alterem o conjunto de comandos disponíveis.

Modelo MVC Tradicional

Arquitetura do MVC [3/3]

32

Modelo MVC



Camada de apresentação:

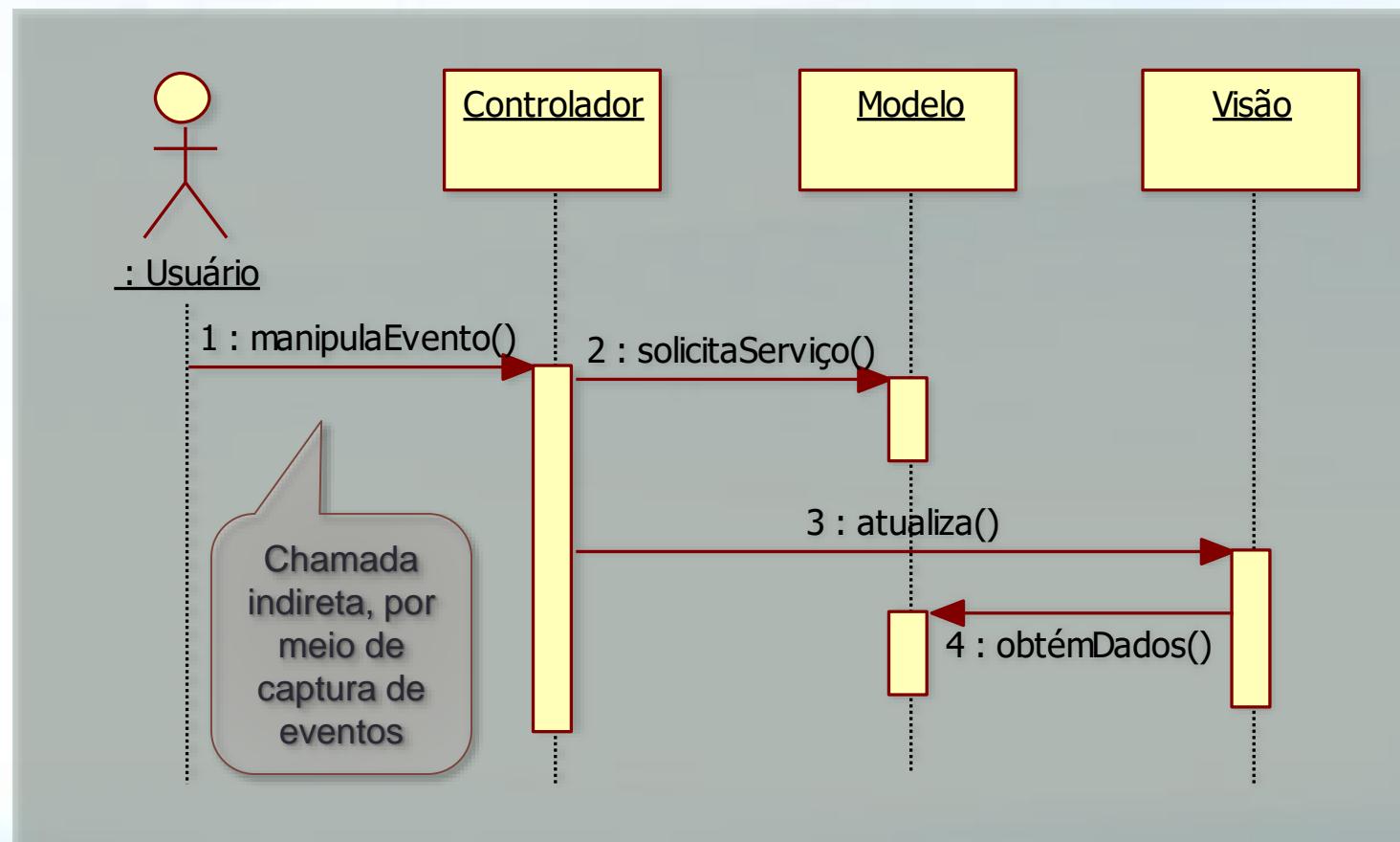
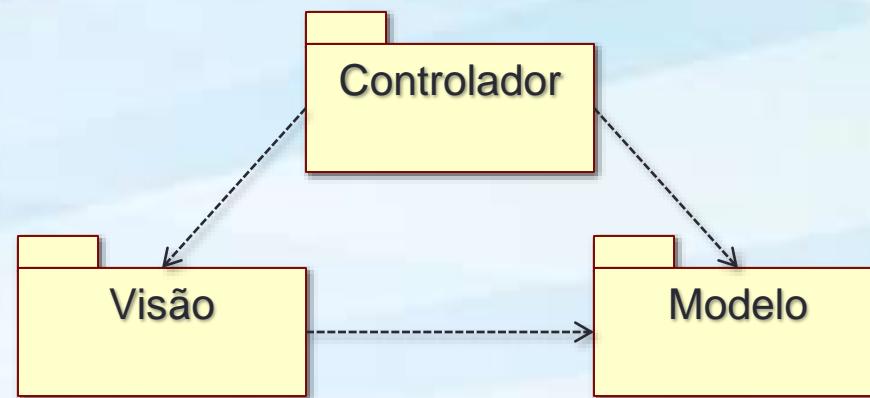
- Exibe as informações do Modelo;
- Responsável pela interface com o usuário;
- Recebe a entradas de dados e apresenta o resultado;

- A visão **solicita do modelo** a informação que ela necessita (mudanças de estado) para gerar uma representação de saída.

Modelo MVC Tradicional

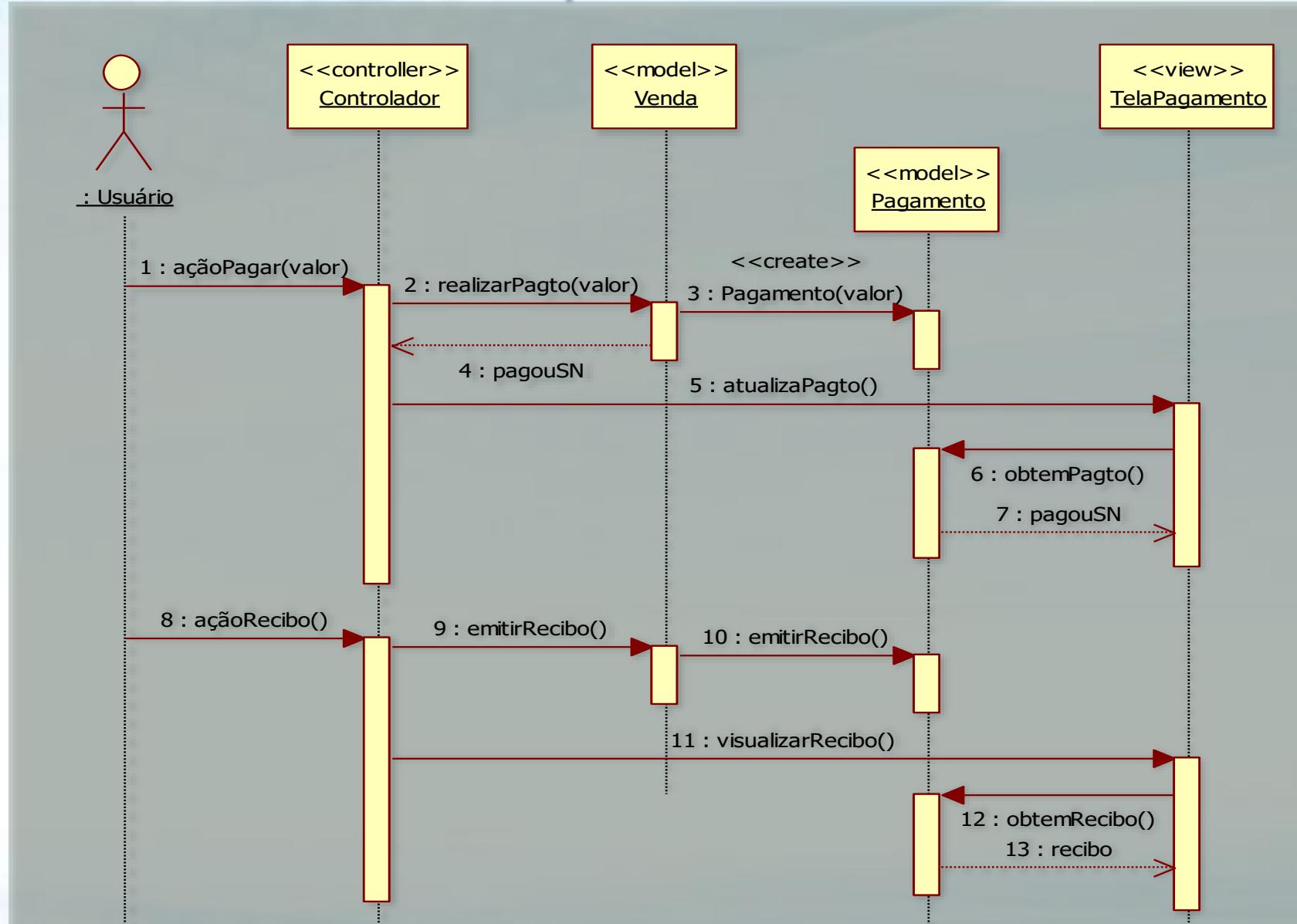
Exemplos de Diagramas

33



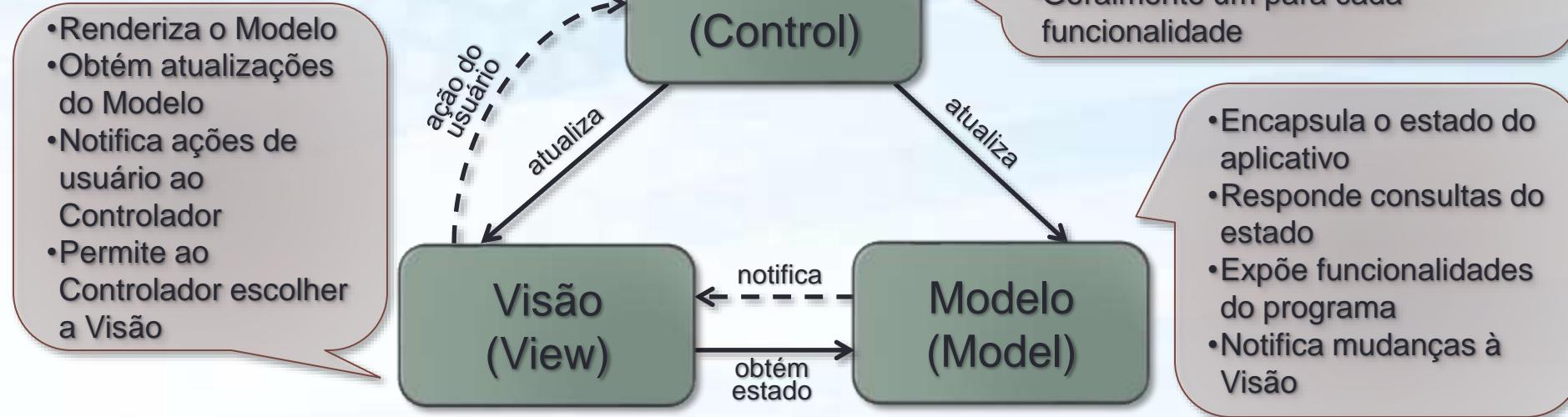
Exemplo: Pagamento

MVC Tradicional considerando padrões GRASP



Modelo MVC Tradicional

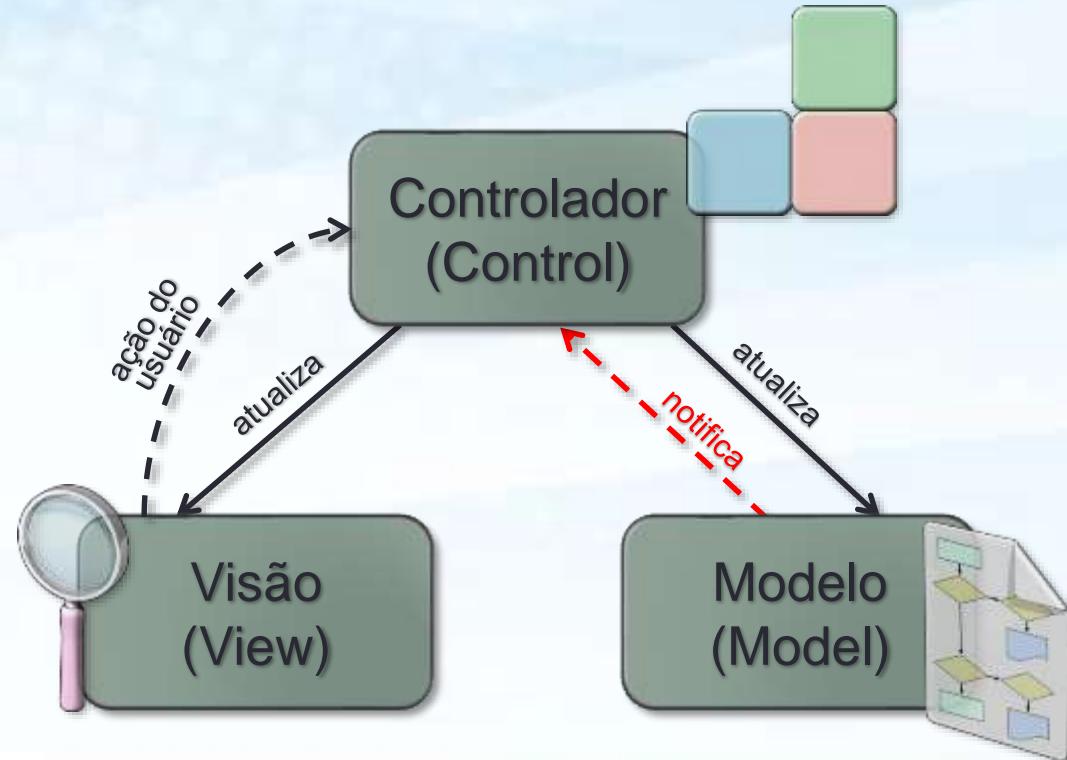
Resumindo...



- O Controlador recebe as requisições de usuário da Visão e as envia para o Modelo;
 - O Controlador conhece a Visão e o Modelo;
- O Modelo atende à requisição:
 - Persiste dados, valida informações etc. e atualiza os dados para serem visualizados novamente pela Visão.
- A Visão observa o Modelo:
 - Quando o Modelo é atualizado, a Visão exibe os dados atualizados para o usuário.

Modelo MVC Web/Model 2

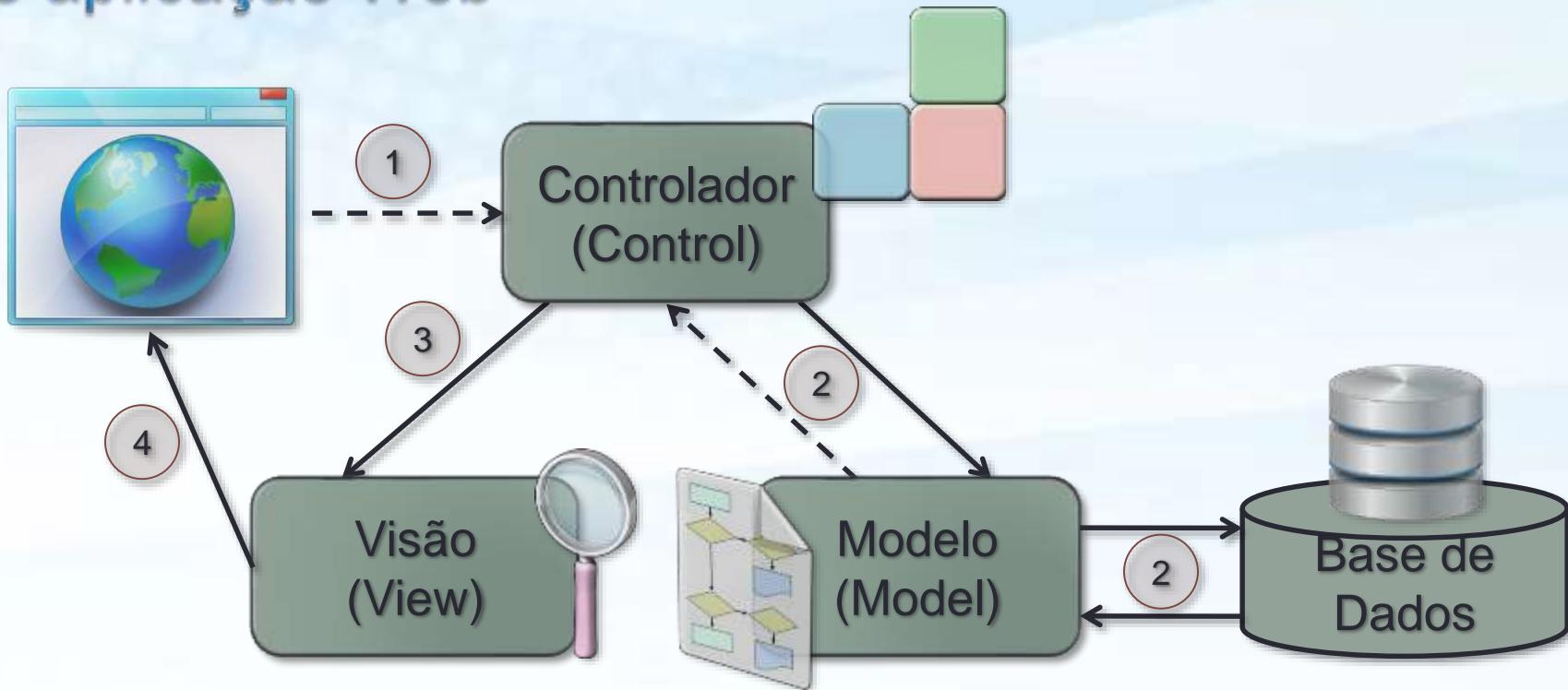
Arquitetura do MVC Web



- O controlador recebe a **notificação do modelo** e atualiza a visão;
- A visão torna-se menos acoplada ao modelo;
- Esta versão atende o desenvolvimento de aplicações web.

Modelo MVC Web/Model 2

Exemplo de aplicação Web

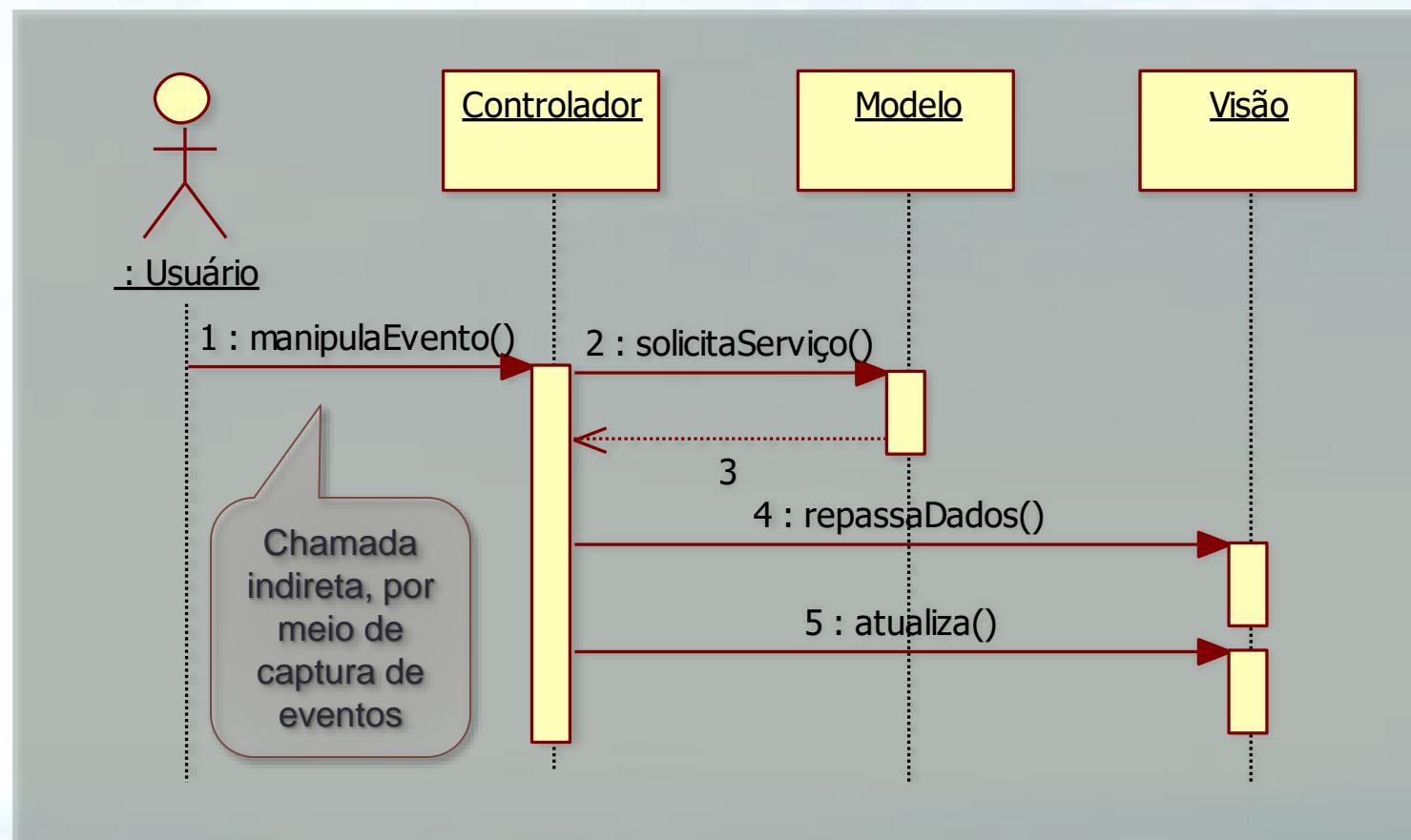


1. Browser envia requisição ao controlador
2. Controlador interage com o modelo e este com a base de dados
3. Controlador invoca a visão
4. Visão renderiza a próxima tela do browser

Modelo MVC Web/Model 2

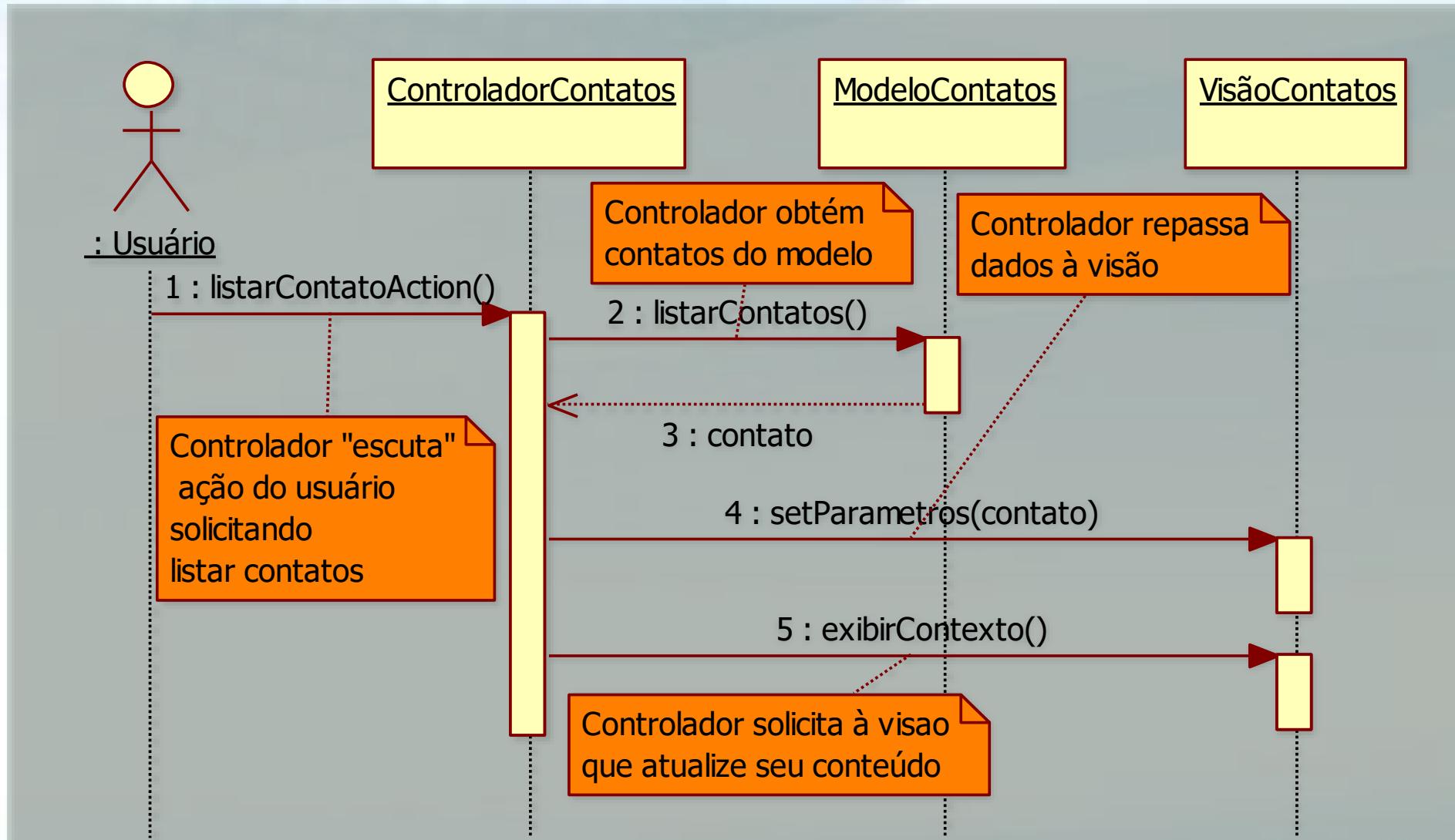
Exemplos de Diagramas

38



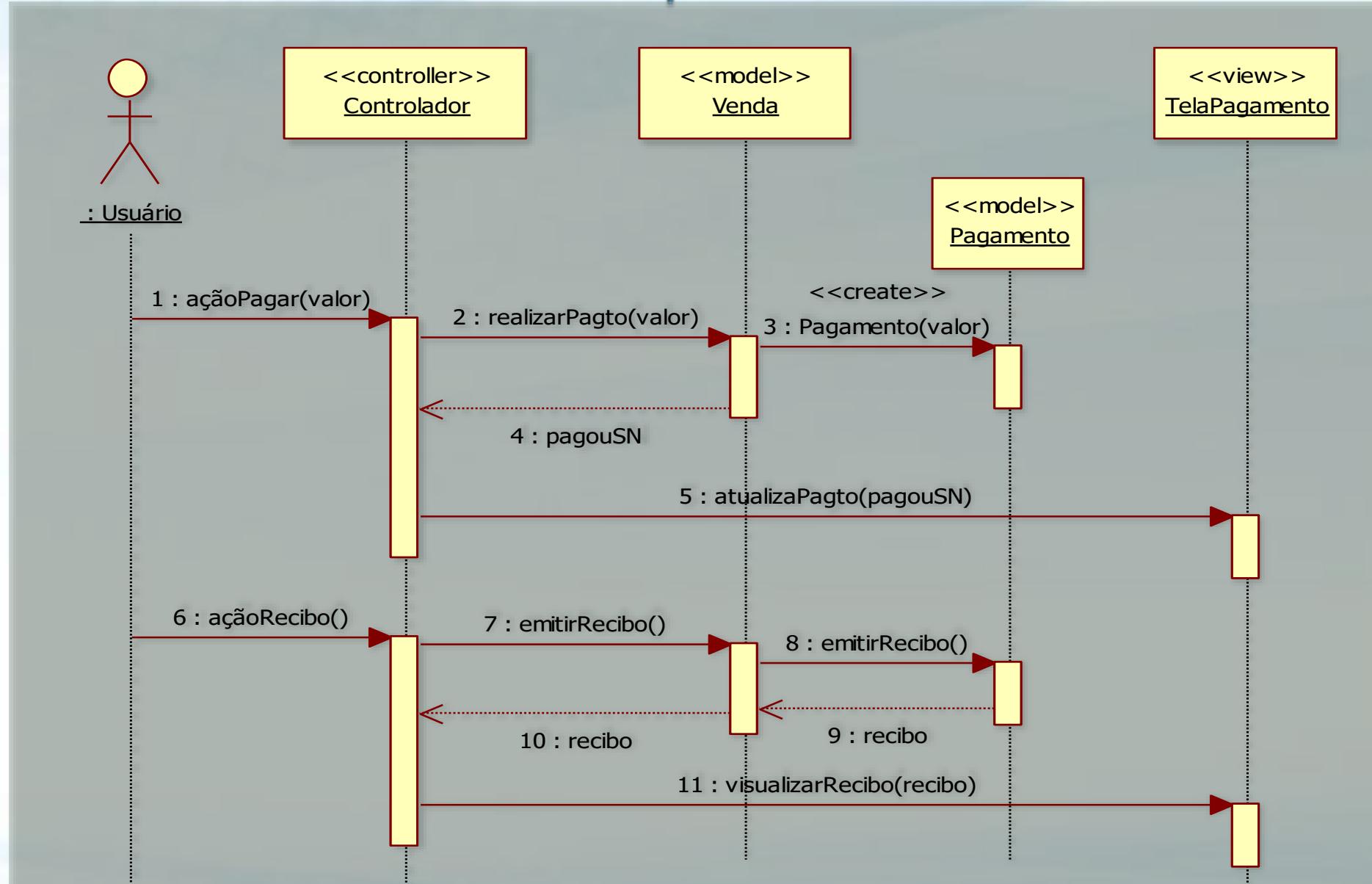
Exemplo: Listar Contatos

MVC Web/Model 2



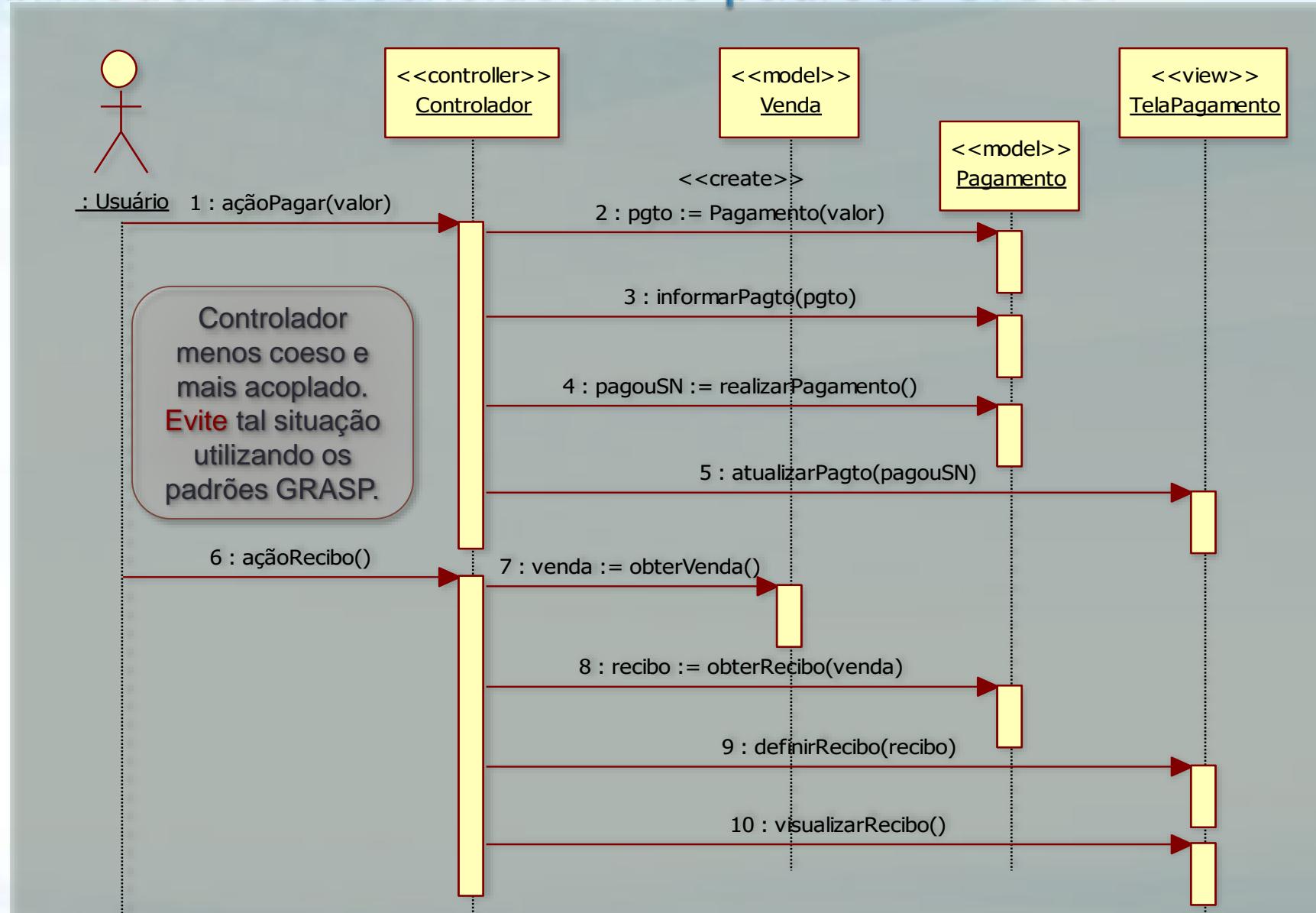
Exemplo: Pagamento

MVC Web/Model 2 considerando padrões GRASP



Exemplo: Pagamento

MVC Web/Model 2 desconsiderando padrões GRASP



Classes/Objetos Especiais

Business Objects e Value Objects [1/5]

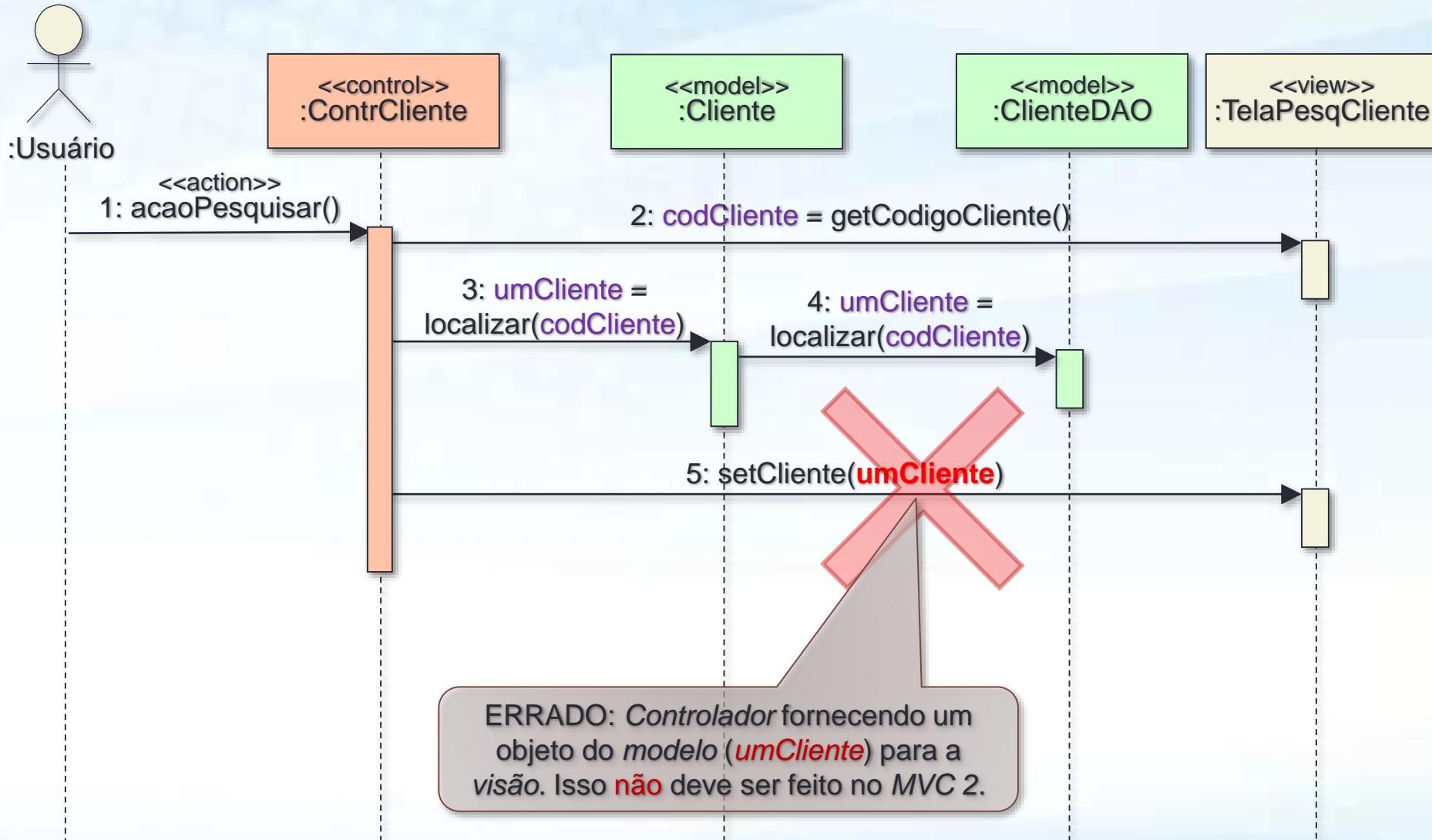
- Como no **MVC 2** a visão não enxerga o modelo, o controlador não deve devolver à uma visão nenhum objeto pertencente ao modelo.
- Considere o exemplo:
 - O usuário informa o código do cliente e clica em *Pesquisar*.
 - O *controlador*, observando a visão, obtém dela o código do cliente, solicita ao *modelo* a pesquisa e fornece à visão os dados do cliente.
 - Mas o *controlador* não deve fornecer à visão um objeto da classe *Cliente*, pois ela pertence ao *modelo*.

Pesquisar Clientes

Código 1035	Pesquisar	<input type="button" value="X"/>
Nome		<input type="button" value="foto"/>
RG	CPF	
<input type="text"/>	<input type="text"/>	
Data Nasc.	E-Mail	
<input type="text"/>	<input type="text"/>	

Classes/Objetos Especiais

Business Objects e Value Objects [2/5]



Classes/Objetos Especiais

Business Objects e Value Objects [3/5]

- Para resolver a questão, em vez de fornecer à vião um objeto do modelo, o controlador pode fornecer:
 - Cada dado do cliente individualmente
 - Pode se tornar inviável dependendo da quantidade de dados
 - Uma estrutura de registro (como o struct do C# ou o JSON)
 - Mas nem toda linguagem trabalha com registro
 - O Java não tem struct
 - Estruturas de ponteiros
 - Trabalhar com os padrões **Business Objects** e **Value Objects**.

Classes/Objetos Especiais

Business Objects e Value Objects [4/5]

- **Business Objects** (Objetos de Negócio):

- Objetos que implementam regras de negócios, a exemplo das classes de entidade.
- Ex.:

UsuarioBO
-codigo: int
-nome: String
-...
+regras de negócios...

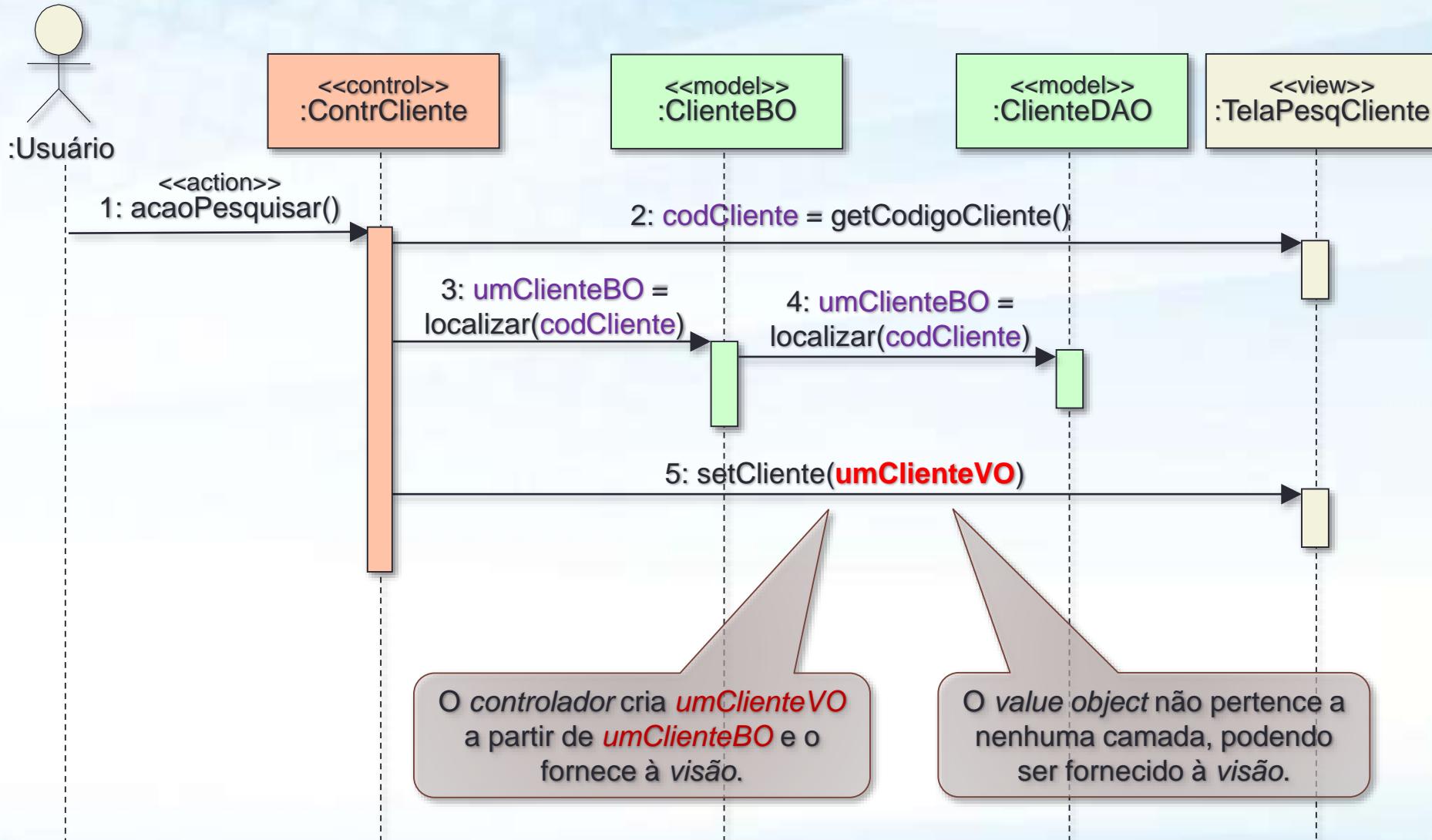
- **Value Objects** (Objetos de Valor):

- Objetos que transportam valores de uma camada para outra (também chamados de Transfer Objects).
- Não possuem métodos (mas podem possuir *gets* e *sets*).
- Seus atributos podem ser públicos.
- Na falta de um struct, é possível usar um *value object* no lugar.
- Ex.:

UsuarioVO
+codigo: int
+nome: String
+...
+gets/sets opcionais...

Classes/Objetos Especiais

Business Objects e Value Objects [5/5]



- **Separação Modelo-Visão:**

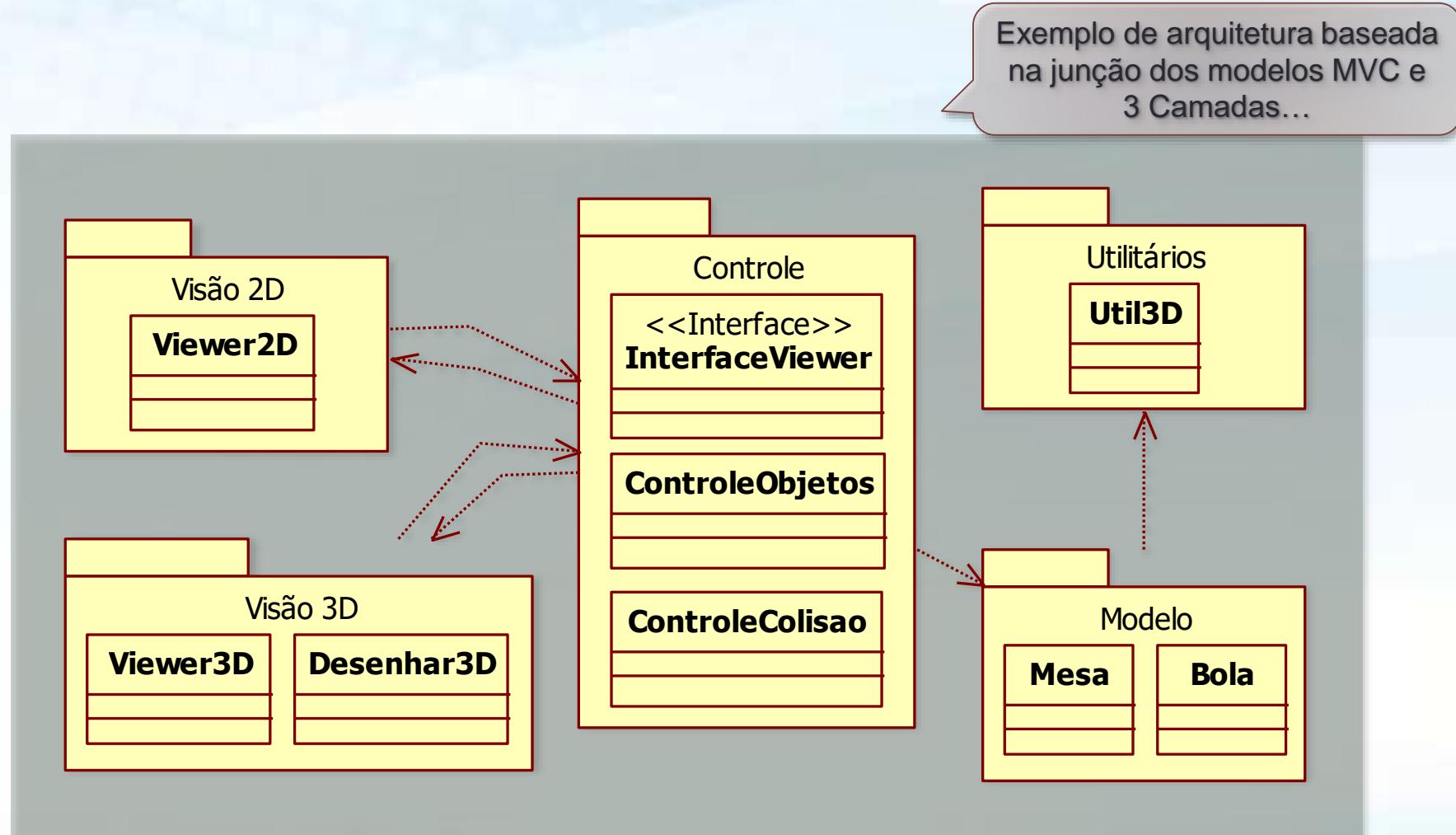
- O modelo de objetos (domínio) não deve ter conhecimento **direto** do dos objetos da *IU* (interface de usuário – visão);
 - Não conecte objetos que não são *IU* diretamente a objetos de *IU*;
 - Ex.: Um objeto *Venda* não deve enviar diretamente uma mensagem um objeto *JanelaVenda*, pedindo a ele para fazer algo, mudar a cor, fechar etc.;
 - As janelas estão relacionadas a uma aplicação em particular, e os objetos não-janela podem ser reutilizados em novas aplicações ou ligados a novas interfaces de usuário.
- Apenas as classes de domínio encapsulam a informação e o comportamento relacionados à lógica da aplicação;
 - Não coloque a lógica da aplicação (como um cálculo de imposto) em métodos de objetos de *IU*;
 - Objetos de *IU* devem apenas iniciar elementos *IU*, receber eventos *IU* (como um clique em um botão) e **delegar solicitações** da lógica da aplicação a objetos não-*IU* (tais como objetos de domínio).

- Apoiar definições **coesivas** de modelo que enfocam processos do **domínio** em vez de **IU**;
- Permitir o desenvolvimento **separado** das camadas de modelo e de IU;
- Minimizar o **impacto** das modificações de **requisitos da IU** sobre a camada de domínio;
- Permitir que novas visões sejam facilmente conectadas a uma camada de domínio existente **sem** afetar a camada de **domínio**;

- Permitir visões **simultâneas** e **múltiplas** do mesmo modelo de objeto
 - Ex.: Permitir tanto uma visão tabular quanto de diagrama de negócio das informações de vendas;
- Permitir a execução da camada de modelo **independente** da camada de *IU*;
 - Ex.: Um sistema que processe mensagens online ou em lote;
- Permitir fácil **portabilidade** da camada de modelo para outro framework de *IU*.
 - Ex.: Framework desktop, web e para dispositivos móveis.

Exemplo

Diagramas de Pacotes de um Jogo de Bilhar



Justificativas para a aplicação do MVC

- Atualmente há um aumento da complexidade de sistemas desktop, web e de dispositivos móveis;
- A arquitetura MVC foca em dividir um grande problema em vários problemas menores e de menor complexidade (camadas);
 - Assim, uma alteração em uma das camadas não interfere nas demais, facilitando a atualização de layouts, alteração nas regras de negócio e adição de novos recursos
- Em grandes projetos, o MVC facilita a divisão de tarefas entre a equipe

Modelo MVC

Vantagens

- Facilita o reaproveitamento de código;
- Facilidade na manutenção e adição de recursos;
- Maior integração da equipe e/ou divisão de tarefas;
- Facilidade em manter o seu código sempre limpo;
- Melhor visibilidade da camada de negócios;
- Diversas tecnologias estão adotando essa arquitetura:
 - Struts (Java);
 - CodeIgniter (PHP);
 - CakePHP (PHP);
 - Cocoa Touch (iOS – Apple);
 - Asp.net MVC (Microsoft);
 - Etc.