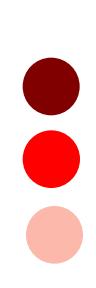


Arquitetura de Software

Engenharia Software III

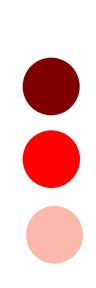
Prof. Me. Emerson Ap. Mouco Jr.

Fonte: Profª. Fabiana Masson



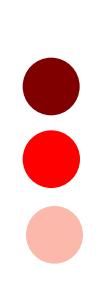
Arquitetura de Software

- Projeto de Arquitetura de Software
- Decisões de projeto de arquitetura
- Visões de arquitetura
- Padrões de arquitetura
- Arquiteturas de aplicações



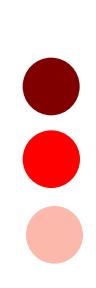
Arquitetura de Software

- Arquitetura em Pequena escala (*software*)
- Arquitetura em larga escala (*sistemas distribuídos*)



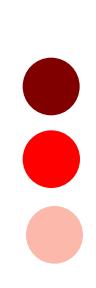
Arquitetura de Software

- Por quê se preocupar com a arquitetura de software?
- Quais as vantagens de projetar/documentar a arquitetura?



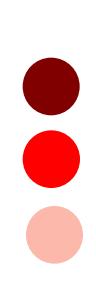
Arquitetura de Software

- Apoio à comunicação com os stakeholders
- Apoio à análise de sistemas
- Apoio ao reuso em larga escala



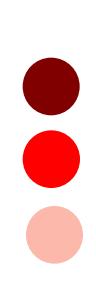
Arquitetura de Software

- Alta coesão
- Baixo acoplamento



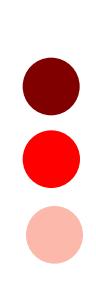
Arquitetura de Software

- Coesão = Divisão de responsabilidades
- Acoplamento = Dependência dos componentes



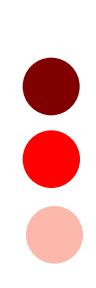
Arquitetura de Software

- A Coesão está ligada ao princípio da responsabilidade única, que diz que uma classe deve ter uma e apenas uma responsabilidade e realizá-la de maneira satisfatória, isto é, a força funcional relativa de um módulo ou componente de software.
- O acoplamento está ligado ao grau de conexão entre módulos em uma estrutura de software.



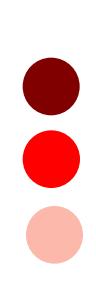
Projeto de Arquitetura de Software

- Busca compreender como um sistema deve ser organizado e com a estrutura geral desse sistema.
- É o primeiro estágio no processo de projeto de software
- É o elo entre o projeto e a engenharia de requisitos



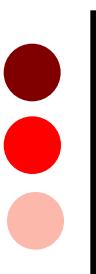
Projeto de Arquitetura de Software

- Qual será a estrutura geral do sistema
- Busca identificar quais são os principais componentes estruturais do sistema e o relacionamento entre eles

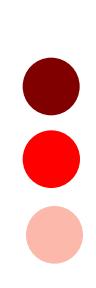


Objetivo do Projeto de Arquitetura

- Identificar os principais componentes estruturais de um sistema e os relacionamentos entre eles.
- Resultado → é um modelo de arquitetura que descreve como o sistema está organizado em um conjunto de componentes de comunicação.

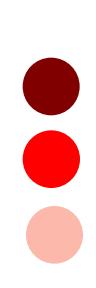


Decisões de projeto de arquitetura



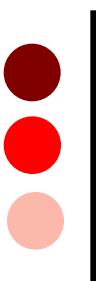
Decisões do projeto de arquitetura

- As decisões a serem tomadas pelos engenheiros de software são de aspectos **estruturais**, que afetam o sistema e seu processo de desenvolvimento.
- Baseado em **conhecimentos** e **experiência** dos engenheiros de software são consideradas as seguintes **questões** fundamentais sobre o sistema:



Questões fundamentais

1. Existe uma arquitetura genérica de aplicação que pode atuar como um modelo para o sistema que está sendo projetado?
2. Como o sistema será distribuído por meio de um número de núcleos ou processadores?
3. Que padrões ou estilos de arquitetura podem ser usados?
4. Qual será a abordagem fundamental para se estruturar o sistema?

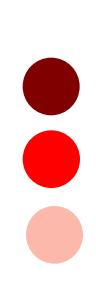


Questões fundamentais

5. Como os componentes estruturais do sistema serão decompostos em subcomponentes?
6. Que estratégia será usada para controlar o funcionamento dos componentes do sistema?
7. Qual a melhor organização de arquitetura para satisfazer os requisitos não funcionais do sistema?
8. Como o projeto de arquitetura será avaliado?
9. Como a arquitetura do sistema deve ser documentada?

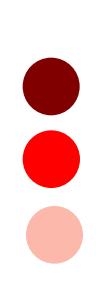


Visões da arquitetura



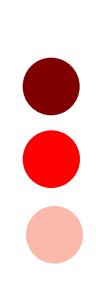
Visões da arquitetura

- As visões auxiliam a criação dos modelos no desenvolvimento do projeto. Cada modelo mostra apenas uma visão ou perspectiva do sistema.
- Kruchten (1995), sugere que deve haver quatro visões fundamentais de arquitetura, relacionadas usando-se casos de uso ou cenários:
 - visão lógica,
 - processo,
 - desenvolvimento e
 - física



Visões da arquitetura

- **Visão lógica** - mostra as abstrações fundamentais do sistema como objetos ou classes de objetos. Os requisitos de sistema são relacionados com as entidades.
- **Visão de processo** - mostra como no tempo de execução, o sistema é composto de processos interativos, como por exemplo as características não funcionais do sistema, como desempenho e disponibilidade.

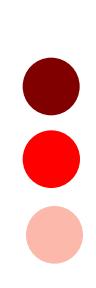


Visões da arquitetura

- **Visão de desenvolvimento** - apresenta a distribuição do software em componentes que são implementados pelos desenvolvedores (sua decomposição). Essa visão é útil para gerentes de software e programadores.
- **Visão física** - mostra o hardware do sistema e como os componentes de software são distribuídos entre os processadores. Essa visão é útil para os engenheiros de sistemas que estão planejando uma implantação do sistema.

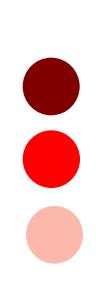


Padrões de arquitetura



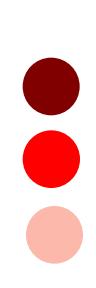
Padrão de arquitetura

- É uma descrição abstrata, estilizada, de boas práticas experimentadas e testadas em diferentes sistemas e ambientes.
- A descrição estilizada de padrão inclui o **nome do padrão**, uma breve **descrição** (com um **modelo gráfico associado**), e um **exemplo** do tipo de sistema em que o padrão é usado.
- Também pode incluir informações sobre quando o padrão deve ser usado e suas vantagens e desvantagens.



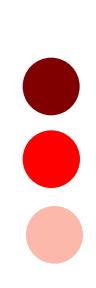
Tipos de arquitetura

- Camadas
- Repositório
- Cliente-servidor
- Duto e filtro



Arquitetura em Camadas

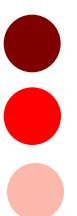
- Nesse tipo de arquitetura as funcionalidades do sistema são organizadas em camadas separadas. Cada camada só depende dos recursos e serviços oferecidos pela camada imediatamente abaixo dela.
- É realizada uma separação e independência das funcionalidade, pois permitem que alterações sejam localizadas.



Descrição estilizada do modelos em Camadas

Nome	Arquitetura em camadas
Descrição	Organiza o sistema em camadas com a funcionalidade relacionada associada a cada camada. Uma camada fornece serviços à camada acima dela; assim, os níveis mais baixos de camadas representam os principais serviços suscetíveis de serem usados em todo o sistema.
Exemplo	Um modelo em camadas de um sistema para compartilhar documentos com direitos autorais, em bibliotecas diferentes,
Quando é usado	É usado na construção de novos recursos em cima de sistemas existentes; quando o desenvolvimento está espalhado por várias equipes, com a responsabilidade de cada equipe em uma camada de funcionalidade; quando há um requisito de proteção multinível.
Vantagens	Desde que a interface seja mantida, permite a substituição de camadas inteiras. Recursos redundantes (por exemplo, autenticação) podem ser fornecidos em cada camada para aumentar a confiança do sistema.
Desvantagens	Na prática, costuma ser difícil proporcionar uma clara separação entre as camadas, e uma camada de alto nível pode ter de interagir diretamente com camadas de baixo nível, em vez de através da camada imediatamente abaixo dela. O desempenho pode ser um problema por causa dos múltiplos níveis de interpretação de uma solicitação de serviço, uma vez que são processados em cada camada.

Fonte: Sommerville, p. 124. 2011.



Arquitetura genérica do modelo em Camadas

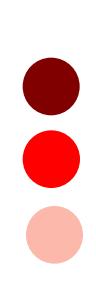
Interface de usuário

Gerenciamento de interface de usuário
Autenticação e autorização

Lógica de negócio principal/funcionalidade de aplicação
Recursos de sistema

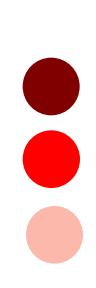
Apoio de sistema (SO, banco de dados etc.)

Fonte: Sommerville, p. 125. 2011.



Padrão MVC

- O padrão MVC (Modelo-Visão-Controlador, em inglês, *Model-View-Controller*) separa os elementos de um sistema, permitindo mudá-los de forma independente.
- É utilizado em muitos projetos.
- **Benefícios:** isolar as regras de negócios da lógica de apresentação, possibilitando várias interfaces para o usuário.
- Assista o vídeo: “**Aprendendo MVC na prática**”
<https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>



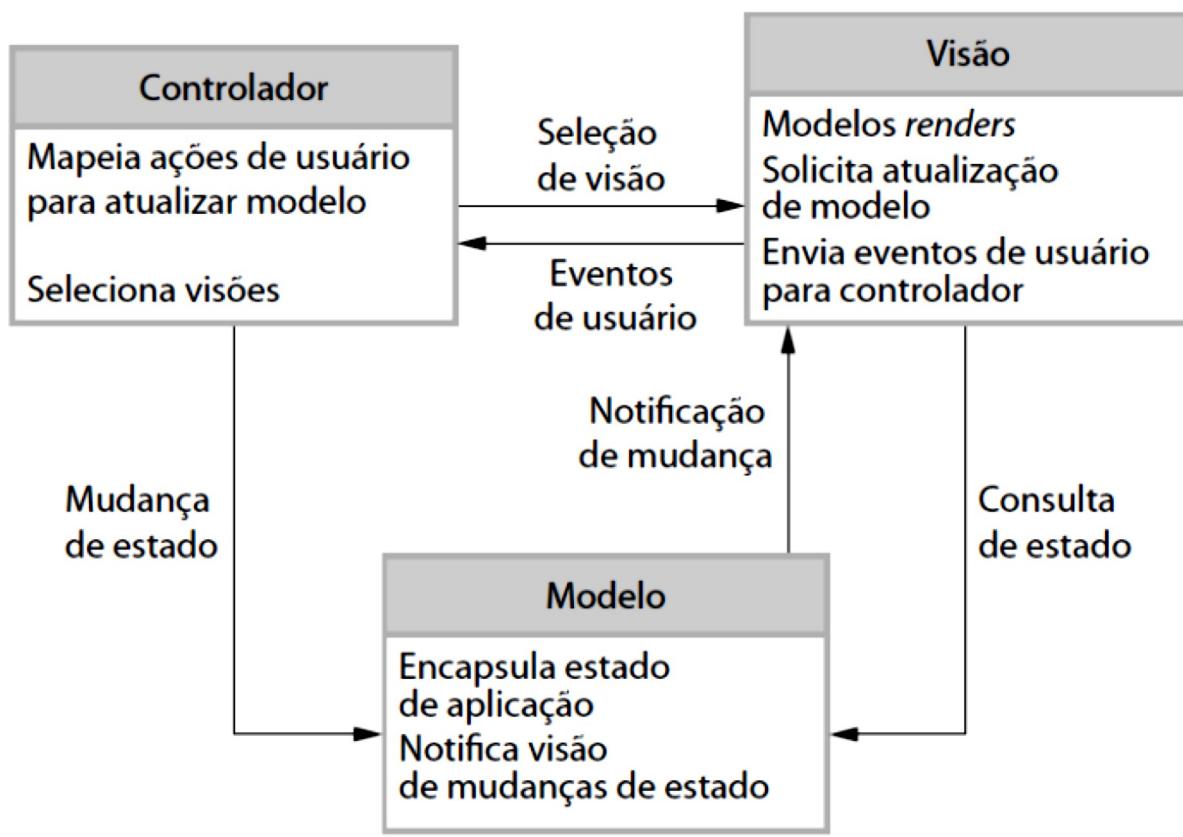
Padrão do modelo MVC

Nome	MVC (Modelo-Visão-Controlador)
Descrição	Separa a apresentação e a interação dos dados do sistema. O sistema é estruturado em três componentes lógicos que interagem entre si. O componente Modelo gerencia o sistema de dados e as operações associadas a esses dados. O componente Visão define e gerencia como os dados são apresentados ao usuário. O componente Controlador gerencia a interação do usuário (por exemplo, teclas, cliques do mouse etc.) e passa essas interações para a Visão e o Modelo. Veja a Figura 6.2.
Exemplo	A Figura 6.3 mostra a arquitetura de um sistema aplicativo baseado na Internet, organizado pelo uso do padrão MVC.
Quando é usado	É usado quando existem várias maneiras de se visualizar e interagir com dados. Também quando são desconhecidos os futuros requisitos de interação e apresentação de dados.
Vantagens	Permite que os dados sejam alterados de forma independente de sua representação, e vice-versa. Apoia a apresentação dos mesmos dados de maneiras diferentes, com as alterações feitas em uma representação aparecendo em todas elas.
Desvantagens	Quando o modelo de dados e as interações são simples, pode envolver código adicional e complexidade de código.

Fonte: Sommerville, p. 123. 2011.

Organização do modelo MVC

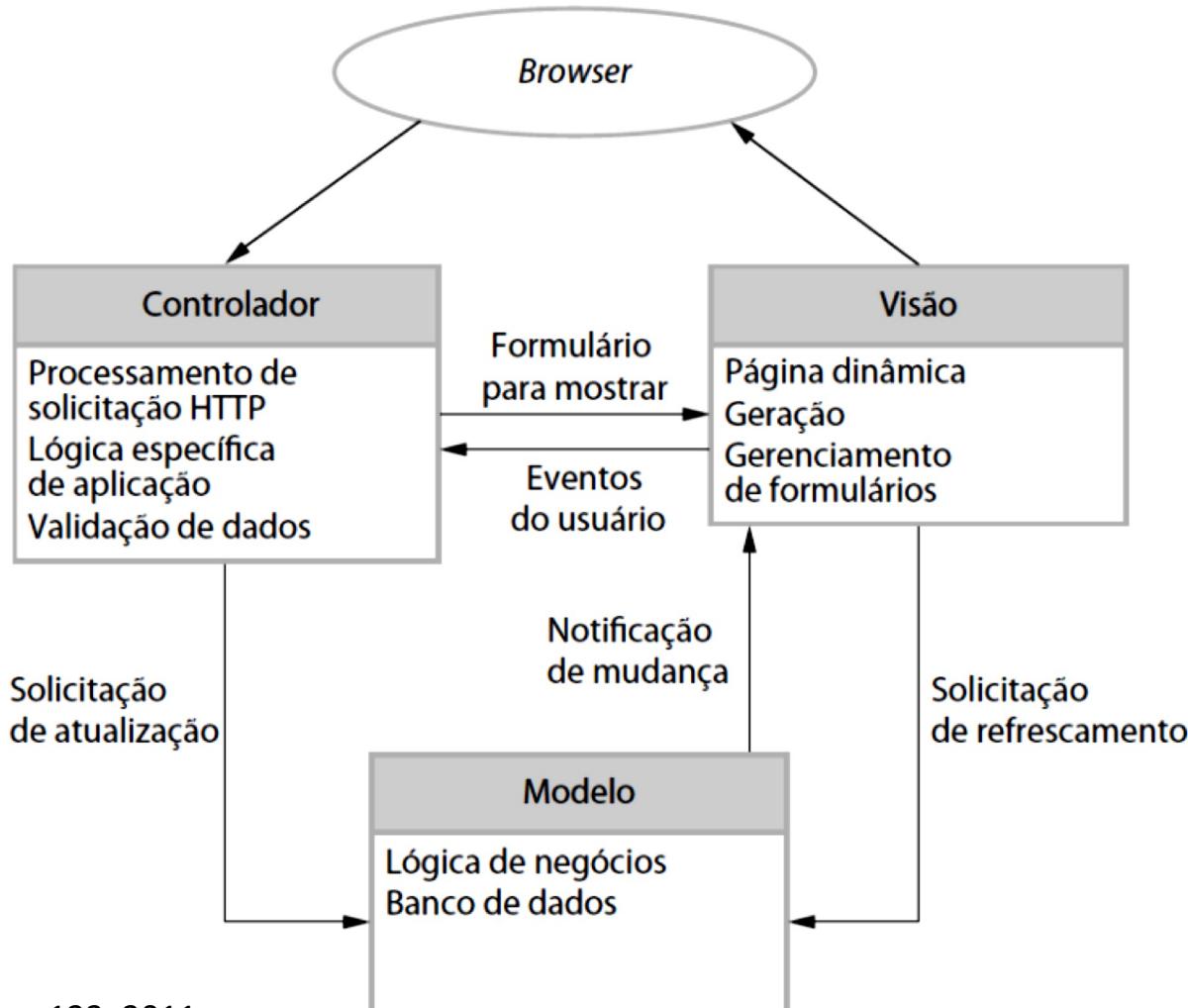
Figura 6.2



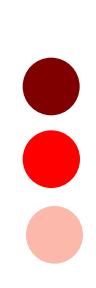
Fonte: Sommerville, p. 123. 2011.

Utilização do padrão MVC em sistemas web

Figura 6.3



Fonte: Sommerville, p. 123. 2011.



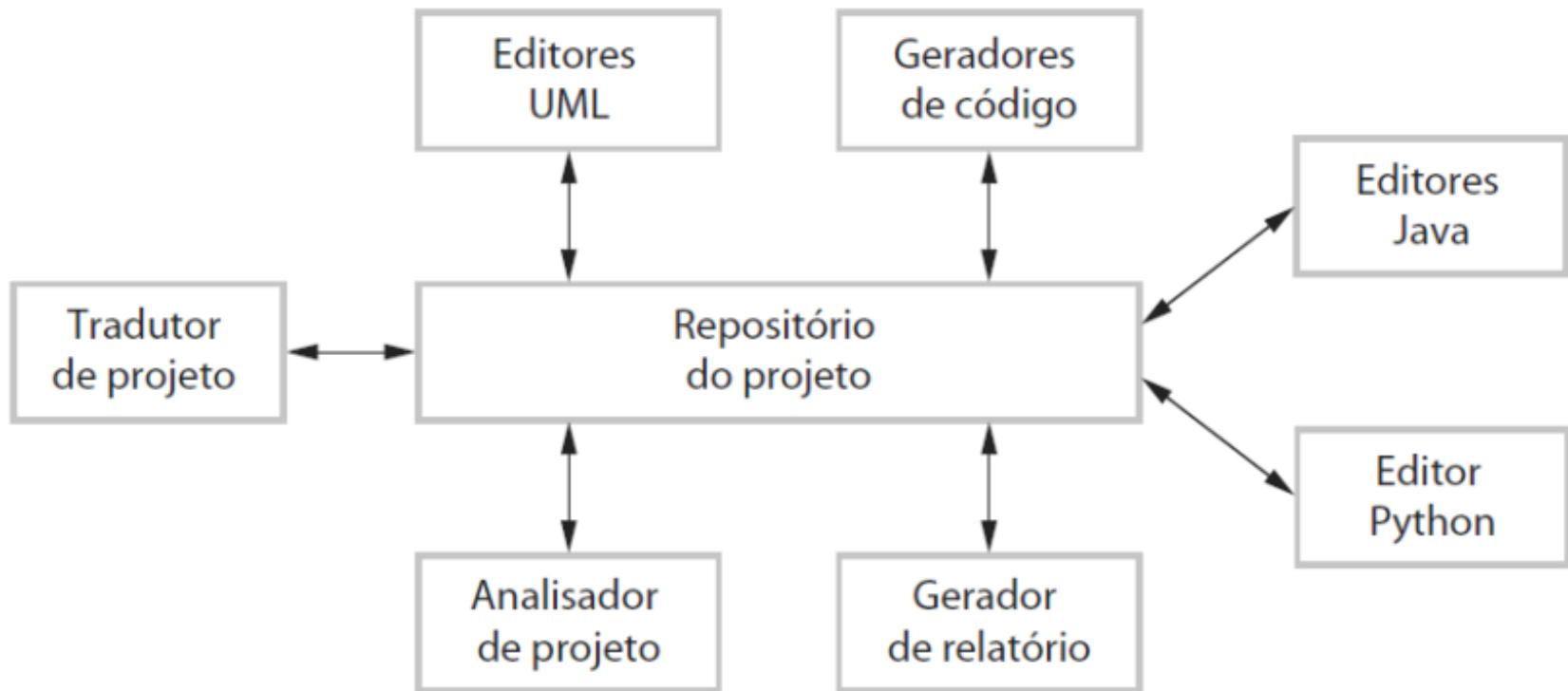
Arquitetura de Repositório

- Descreve como um conjunto de componentes que interagem podem compartilhar dados.
- Esse modelo é adequado para aplicações nas quais os dados são gerados por um componente e usados por outro.
- Esse padrão está preocupado com a estrutura estática de um sistema e não mostra sua organização de tempo de execução.
- Exemplo: em um ambiente controlado por versões, que mantém o controle das alterações de software e permite a reversão para versões anteriores.

Padrão do modelo Repositório

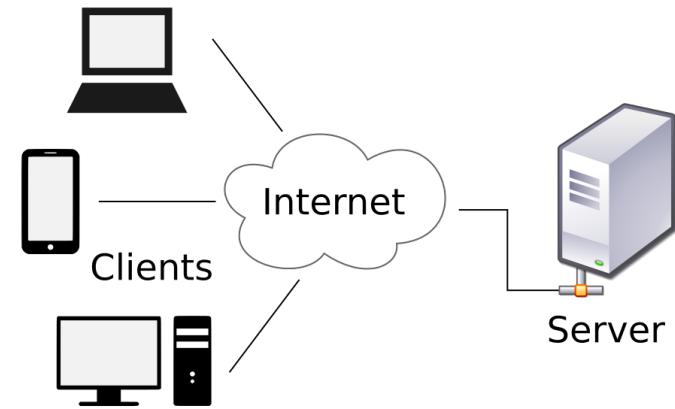
Nome	Repositório
Descrição	Todos os dados em um sistema são gerenciados em um repositório central, acessível a todos os componentes do sistema. Os componentes não interagem diretamente, apenas por meio do repositório.
Exemplo	A Figura 6.6 é um exemplo de um IDE em que os componentes usam um repositório de informações sobre projetos de sistema. Cada ferramenta de software gera informações que ficam disponíveis para uso por outras ferramentas.
Quando é usado	Você deve usar esse padrão quando tem um sistema no qual grandes volumes de informações são gerados e precisam ser armazenados por um longo tempo. Você também pode usá-lo em sistemas dirigidos a dados, nos quais a inclusão dos dados no repositório dispara uma ação ou ferramenta.
Vantagens	Os componentes podem ser independentes — eles não precisam saber da existência de outros componentes. As alterações feitas a um componente podem propagar-se para todos os outros. Todos os dados podem ser gerenciados de forma consistente (por exemplo, <i>backups</i> feitos ao mesmo tempo), pois tudo está em um só lugar.
Desvantagens	O repositório é um ponto único de falha, assim, problemas no repositório podem afetar todo o sistema. Pode haver ineficiências na organização de toda a comunicação através do repositório. Distribuir o repositório através de vários computadores pode ser difícil.

Exemplo de utilização da arquitetura Repositório

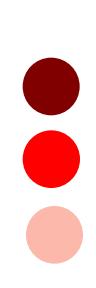


Arquitetura Cliente-Servidor

- É organizado como um conjunto de serviços e servidores associados e clientes que acessam e usam os serviços.
- É muito usada para sistemas distribuídos, em tempo de execução.
- O modelo lógico de serviços independentes funciona em servidores separados e pode ser implementado em um único computador.



Fonte: [CC BY-SA-NC](#)



Componentes do modelo Cliente-Servidor

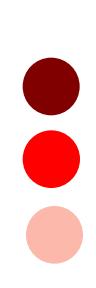
- **Servidores** - oferecem serviços a outros componentes, exemplo: de impressão, arquivos, compiladores, etc.
- **Cientes** – solicitam os serviços oferecidos pelos servidores.
- **Rede** - permite aos clientes acessar esses serviços, a maioria conectados através de protocolos de Internet, como HTTP.



Padrão do modelo Cliente-Servidor

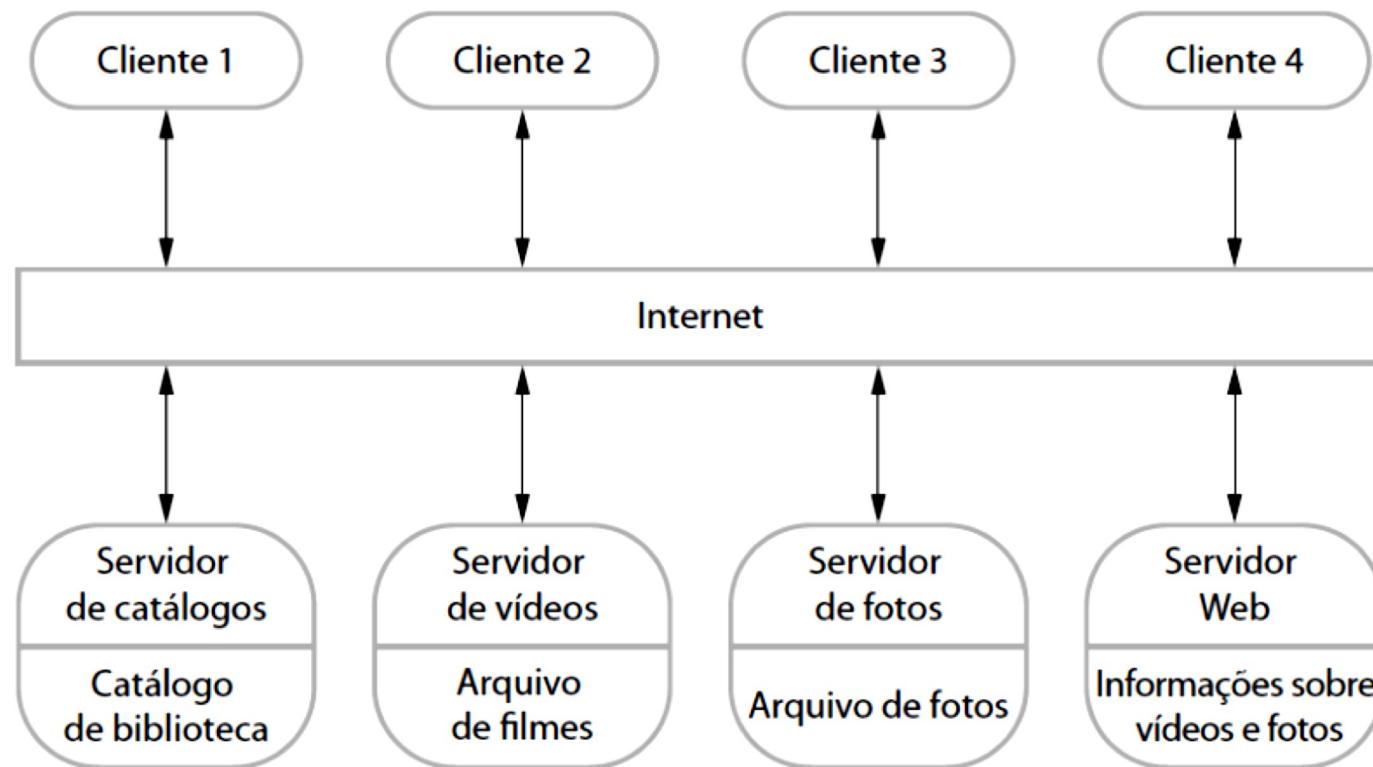
Nome	Cliente-servidor
Descrição	Em uma arquitetura cliente-servidor, a funcionalidade do sistema está organizada em serviços — cada serviço é prestado por um servidor. Os clientes são os usuários desses serviços e acessam os servidores para fazer uso deles.
Exemplo	A Figura 6.7 é um exemplo de uma biblioteca de filmes e vídeos/DVDs, organizados como um sistema cliente-servidor.
Quando é usado	É usado quando os dados em um banco de dados compartilhado precisam ser acessados a partir de uma série de locais. Como os servidores podem ser replicados, também pode ser usado quando a carga em um sistema é variável.
Vantagens	A principal vantagem desse modelo é que os servidores podem ser distribuídos através de uma rede. A funcionalidade geral (por exemplo, um serviço de impressão) pode estar disponível para todos os clientes e não precisa ser implementada por todos os serviços.
Desvantagens	Cada serviço é um ponto único de falha suscetível a ataques de negação de serviço ou de falha do servidor. O desempenho, bem como o sistema, pode ser imprevisível, pois depende da rede. Pode haver problemas de gerenciamento se os servidores forem propriedade de diferentes organizações.

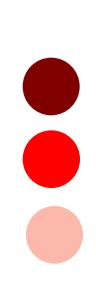
Fonte: Sommerville, p. 127. 2011.



Exemplo de utilização da arquitetura Cliente-Servidor

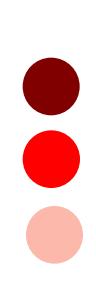
- Figura 6.7





Arquitetura de duto e filtro

- Esse é um modelo de organização em tempo de execução de um sistema no qual as transformações funcionais processam suas entradas e produzem saídas.
- Os dados de entrada fluem por meio de transformações até serem convertidos em saídas. As transformações podem ser executadas sequencialmente ou em paralelo. Os dados podem ser processados por cada item de transformação ou em um único lote.
- O nome ‘duto e filtro’ vem do sistema original Unix, em que foi possível vincular os processos usando ‘dutos’.

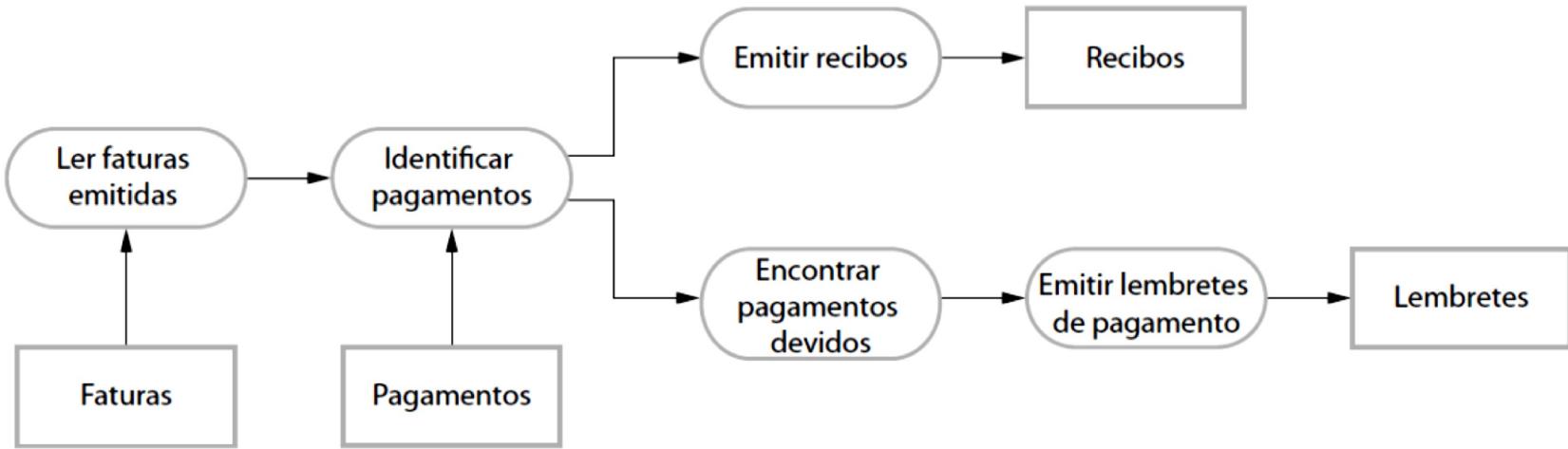


Padrão do modelo dutos e filtro

Nome	Duto e filtro
Descrição	O processamento dos dados em um sistema está organizado de modo que cada componente de processamento (filtro) seja discreto e realize um tipo de transformação de dados. Os dados fluem (como em um duto) de um componente para outro para processamento.
Exemplo	A Figura 6.8 é um exemplo de um sistema de duto e filtro usado para o processamento das faturas.
Quando é usado	Comumente, é usado em aplicações de processamento de dados (tanto as baseadas em lotes como as baseadas em transações) em que as entradas são processadas em etapas separadas para gerarem saídas relacionadas.
Vantagens	O reúso da transformação é de fácil compreensão e suporte. Estilo de <i>workflow</i> corresponde à estrutura de muitos processos de negócios. Evolução por adição de transformações é simples. Pode ser implementado tanto como um sistema sequencial quanto concorrente.
Desvantagens	O formato para transferência de dados tem de ser acordado entre as transformações de comunicação. Cada transformação deve analisar suas entradas e gerar as saídas para um formato acordado. Isso aumenta o <i>overhead</i> do sistema e pode significar a impossibilidade de reúso de transformações funcionais que usam estruturas incompatíveis de dados.

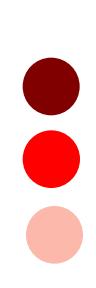
Fonte: Sommerville, p. 128. 2011.

Exemplo de utilização da arquitetura dutos e filtros



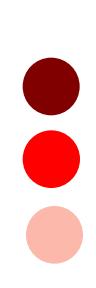


Arquiteturas de aplicações



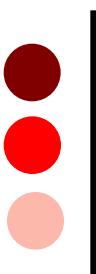
Arquiteturas de aplicações

- Essa arquitetura está presente nos sistemas de aplicações são projetados para atender a uma necessidade organizacional.
- Os sistemas de aplicações são projetados para atender a uma necessidade organizacional.
- Pode ser reimplementada no desenvolvimento de novos sistemas, mas, para sistemas de muitas empresas, o reuso de aplicações é possível sem a reimplementação.



Exemplos

- Os sistemas *Enterprise Resource Planning* (ERP) de empresas como SAP e Oracle, e pacotes de software verticais (COTS) para aplicações especializadas em diferentes áreas de negócio.
- Nesses sistemas, um sistema genérico é configurado e adaptado para criar uma aplicação específica de negócio.



Bibliografia

- BOOCH, G.; RUMBAUGH, J; JACOBSON, I. **UML: Guia do usuário.** Elsevier, 2006.
- GOES, W.M. **Aprenda UML por meio de Estudos de Caso.** São Paulo: Novatec, 2014.
- GUEDES, G. **UML 2 – Uma Abordagem Prática.** São Paulo: Novatec, 2009.
- LARMAN, C. **Utilizando UML e padrões: uma introdução à análise e projeto orientados a objetos e ao desenvolvimento iterativo.** Bookman, 2007.
- PAULA FILHO, W. P. **Engenharia de Software:** Fundamentos, Métodos e Padrões. 3. ed. Rio de Janeiro: LTC, 2012.
- SOMERVILLE, I. **Engenharia de Software.** São Paulo: Addison Wesley Brasil, 2007.