Design de Software

Arquitetura de Software

Baseado no material de Rubén Béjar

Habib Asseiss Neto

INSTITUTO FEDERAL
Mato Grosso do Sul

Nesta aula

- Isolamento de domínio
- Arquitetura em camadas
- Arquitetura Cliente-Servidor

Isolamento de domínio

- A parte de um software que resolve especificamente problemas de domínio geralmente é pequena
- É importante dissociar os objetos de domínio do restante da funcionalidade e dos aspectos técnicos do sistema

Isolamento de domínio

- UI (*User Interface*), acesso a banco de dados e código de suporte podem ser misturados com o código que expressa o domínio do problema
  - Por exemplo, incluir regras de domínio em componentes da UI ou consultas a bancos de dados
- A curto prazo, mesclar código resolve o problema. Porém se o código relacionado ao domínio estiver disperso e misturado com outro código:
  - É mais difícil localizá-lo e raciocinar sobre ele
  - Qualquer alteração superficial (na UI ou no banco de dados) pode alterar inadvertidamente a lógica do domínio
  - Uma alteração em alguma regra de negócio envolve a busca de código, que é misto e disperso
  - O teste automatizado se torna mais complicado

Arquitetura em camadas (Layered)

- A solução mais comum para esse problema é uma arquitetura em camadas, em que cada camada é especializada em um aspecto do software
- Normalmente, define-se três ou quatro camadas
- Em alguns projetos, a camada da interface do usuário e a camada do aplicativo não são separadas
- O que é crucial é que a camada de domínio seja separada

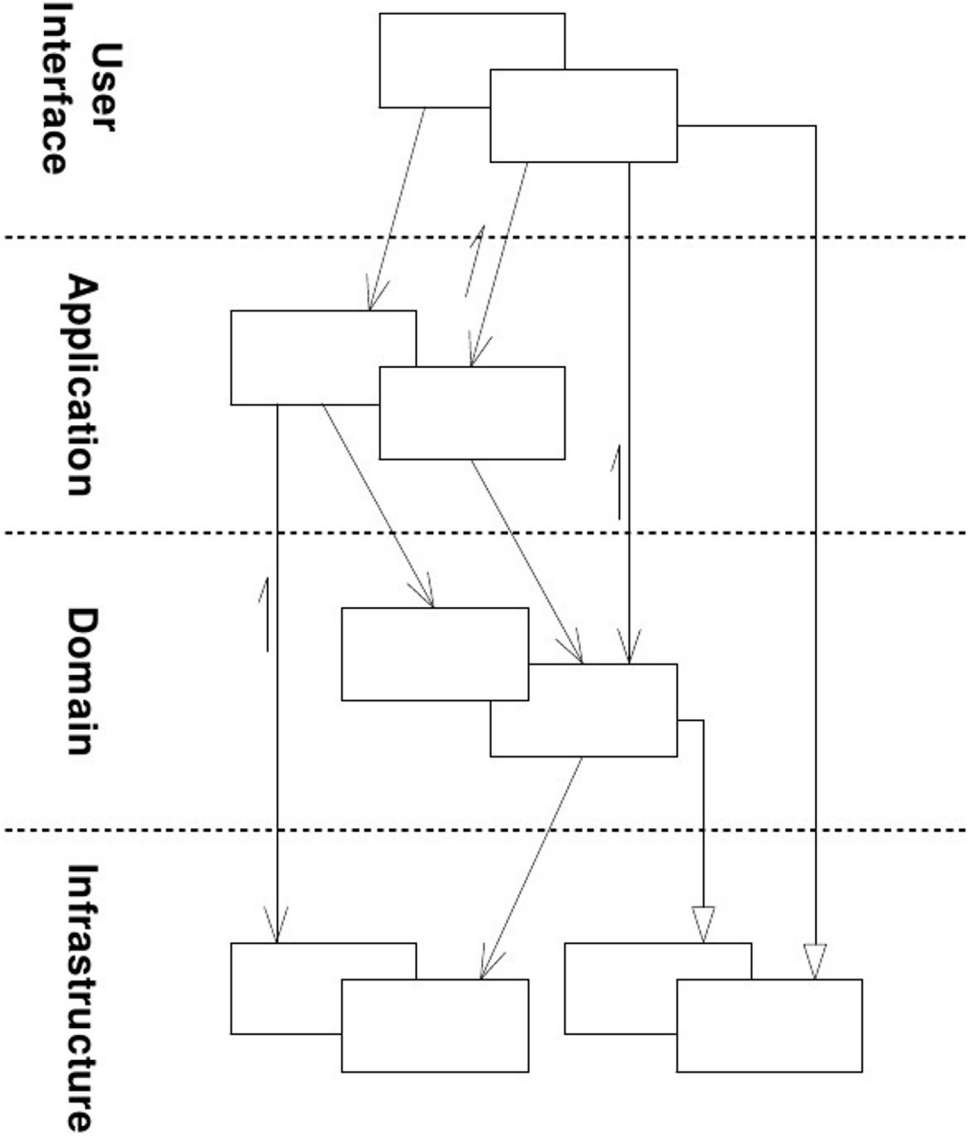
Arquitetura em camadas

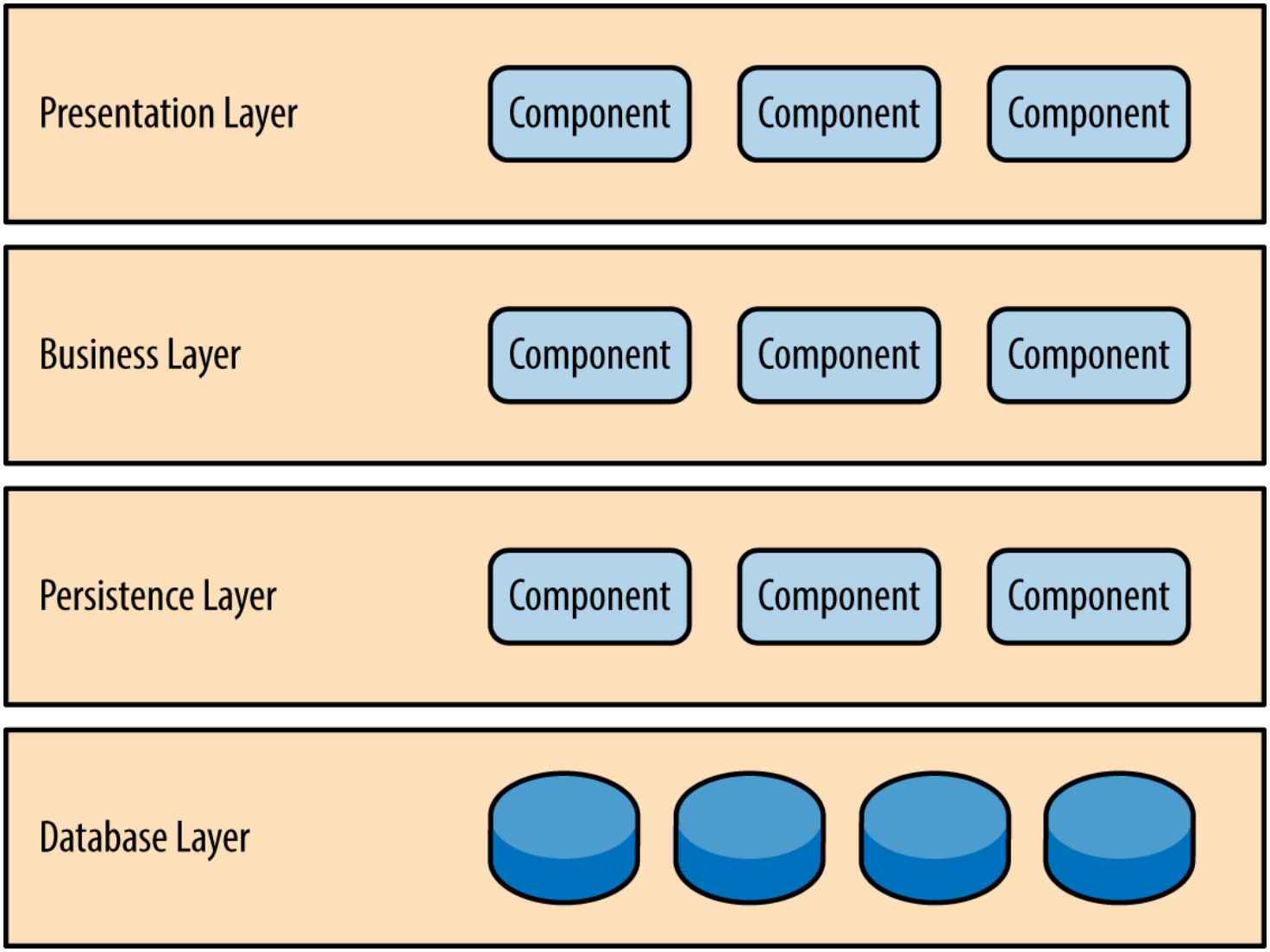
- **Interface do usuário (camada de apresentação)**
  - Exibe informações ao usuário e interpreta comandos do usuário
- **Camada de aplicação**
  - Define o que o programa que estamos criando faz e atribui tarefas específicas a objetos de domínio
  - Não contém conhecimento de domínio, é essencialmente uma camada de coordenação
  - Não armazena status do domínio, mas pode armazenar o status que reflete o progresso de um usuário ou tarefa do programa

Arquitetura em camadas

- **Camada de domínio (camada de modelo ou camada de lógica de negócio)**
  - Contém os conceitos e as regras do domínio e o status do domínio
  - Esse estado é controlado e usado aqui, mas os detalhes técnicos do armazenamento são delegados à infraestrutura
  - Essa camada é o núcleo do software
- **Camada de infraestrutura**
  - Tecnologia genérica para dar suporte às camadas superiores
  - Envio de mensagens, persistência

Layered Architecture




Presentation Layer
Component
Component
Component
Business Layer
Component
Component
Component
Persistence Layer
Component
Component
Component
Database Layer

Arquitetura em camadas

- Cada camada deve ser coesa e estar conectada às outras camadas, mas não muito intimamente ligada a elas
- Os objetos em uma camada devem ter dependências apenas com objetos em uma camada inferior
  - Chamada de suas interfaces públicas e armazenamento de referências a elas

Arquitetura em camadas

- Se um objeto em uma camada inferior precisar se comunicar com um objeto em uma camada superior (além, é claro, de responder quando for invocado de cima), ele deverá fazer isso por meio de métodos indiretos
  - Por exemplo, usando retornos de chamada ou o padrão observador (“*observer*”)
- A conexão entre a camada de interface do usuário e a camada de domínio normalmente utiliza alguma variante do padrão modelo-visão-controlador (MVC)

Arquitetura em camadas

- A camada de infraestrutura pode fornecer serviços genéricos (por exemplo, envio de e-mails) para a camada de aplicativos
    - Os objetos da camada de aplicativos sabem quando enviar um e-mail, mas não necessariamente sabem como fazê-lo
- Nem toda infraestrutura pode ser separada de forma limpa na forma de serviços

Diagramas e Implementação

- Para que o design orientado por domínio funcione, o modelo de domínio precisa ser único, mesmo que o expressemos de diferentes maneiras e em diferentes níveis de detalhes
  - Por exemplo, se nossos diagramas UML dizem uma coisa e nosso código faz outra, não temos um bom modelo de domínio
- Todas as associações entre classes são igualmente fáceis de desenhar em um diagrama, mas algumas são mais complicadas de implementar do que outras

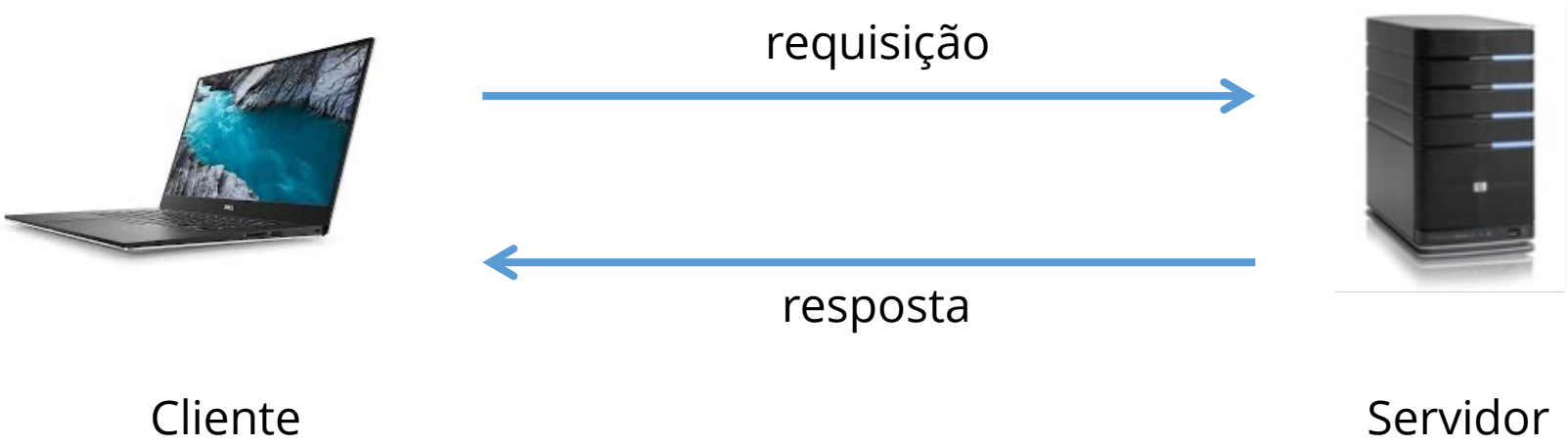
Arquiteturas de software

- Camadas (Layered)
- Microsserviços
- Publicador-Inscrito (Pub/Sub)
- Cliente-Servidor (Client-Server)
- Orientado a Eventos (Event-driven)

Modelo Cliente-Servidor

- É uma estrutura de aplicação distribuída
- Separa a aplicação para duas diferentes entidades:
  - **Servidor**: fornecedor de um recurso ou serviço
  - **Cliente**: requerentes ou consumidores de um recurso ou serviço

Modelo Cliente-Servidor



Cliente

- Na maioria das vezes, é o navegador do usuário
- É capaz de exibir páginas HTML
  - CSS adiciona estilos visuais
  - Linguagem JavaScript oferece algum dinamismo
- Pode, também, ser um aplicativo móvel
- **Realiza requisições ao servidor e espera respostas**

Servidor

- Sempre espera por uma requisição
- Quando uma requisição é recebida, processa e gera uma resposta
- Respostas podem ser arquivos HTML, JSON, imagens, vídeos, etc

Servidor

- Servidores HTTP mais usados:
  - Apache, Nginx
- Módulos permitem ao servidor a execução de códigos dinâmicos com uma linguagem de programação
- Muitos outros serviços podem ser invocados pelo servidor

Exercícios

Pesquisar e responder:

- Um software pode utilizar a Arquitetura em Camadas (Layered) e a Arquitetura Cliente-Servidor, ambas ao mesmo tempo? Se sim, como?

- O conceito de Arquitetura de Software nem sempre foca apenas na estrutura de código, mas também envolve decisões de alto nível sobre a organização do sistema. Quais podem ser essas outras decisões envolvidas?