Arquitetura de Software !

Estilos de Arquitetura

Prof. Galdir Reges

Tópicos

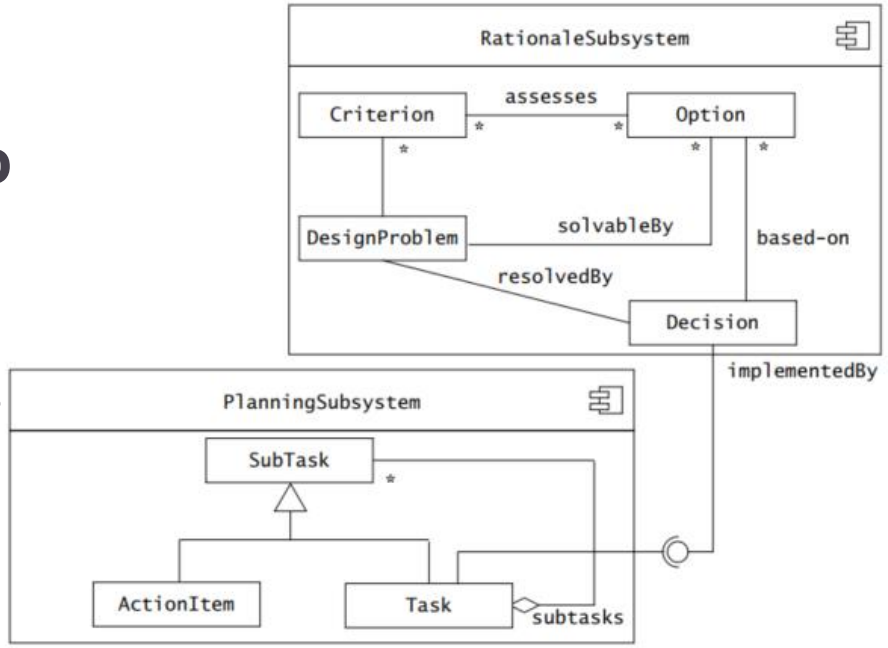
- ✧ Visão geral
- ✧ Estilos
- ✧ Atividades

Pesquisa diagnóstica

- ✧ Brainstorming: Que estilos de arquitetura vocês lembram?
- ✧ Quais suas vantagens?

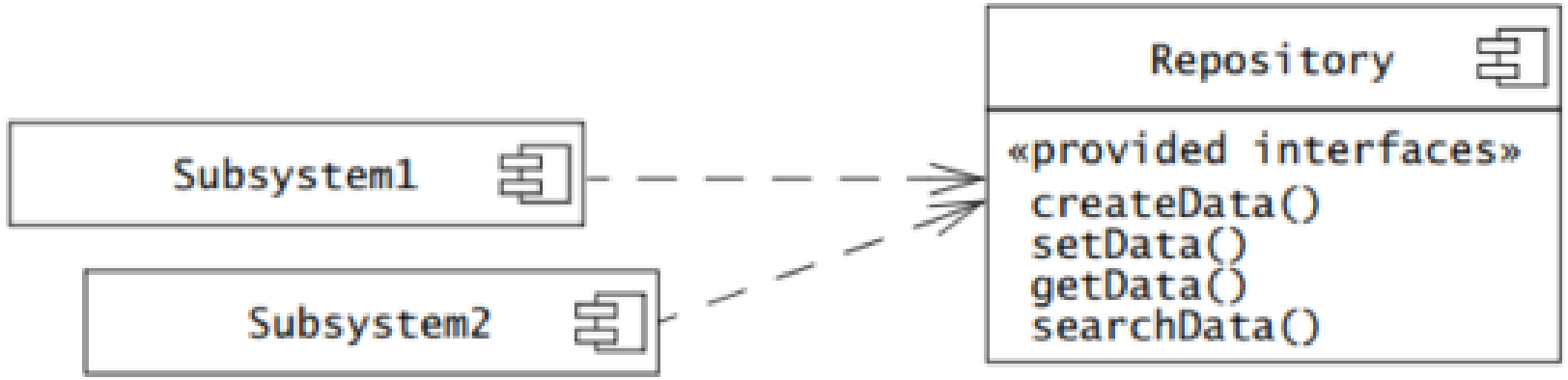
Estilos de arquitetura de software

- ✧ A decomposição do sistema é crítica!
  - É difícil modificar ou corrigir uma decomposição fraca após o início do desenvolvimento, pois a maioria das interfaces do subsistema precisaria mudar.
- ✧ Uma arquitetura de software inclui **decomposição do sistema**, **fluxo de controle global**, **manipulação de condições de contorno** e **protocolos de comunicação entre subsistemas** [Shaw & Garlan, 1996].
- ✧ Estilos de arquitetura podem ser **usados como base para a arquitetura de diferentes sistemas**.



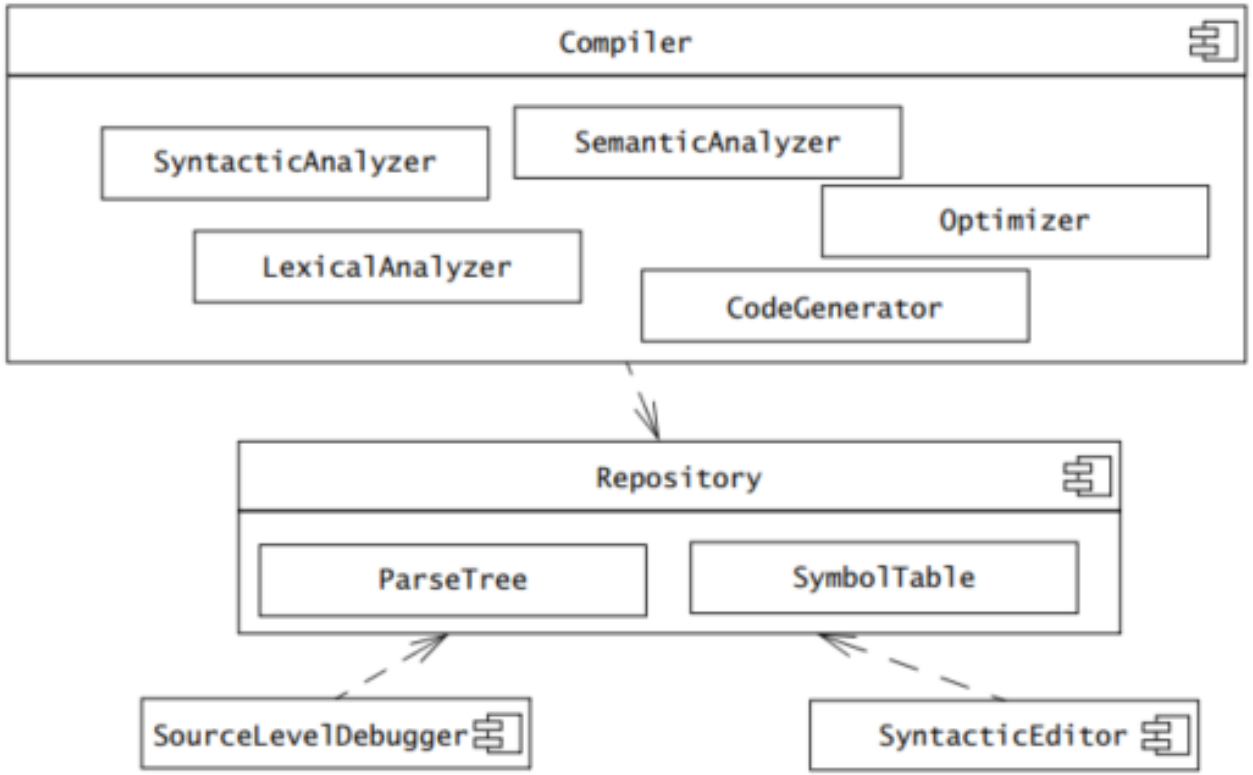
O estilo Repositório

- Os subsistemas acessam e modificam uma **única estrutura de dados** chamada repositório central.
- Os subsistemas são relativamente independentes e **interagem apenas através do repositório.**
- O **fluxo de controle** pode ser **ditado pelo repositório central** (por exemplo, gatilhos nos dados invocam sistemas periféricos) **ou pelos subsistemas** (por exemplo, fluxo independente de controle e sincronização através de bloqueios no repositório)



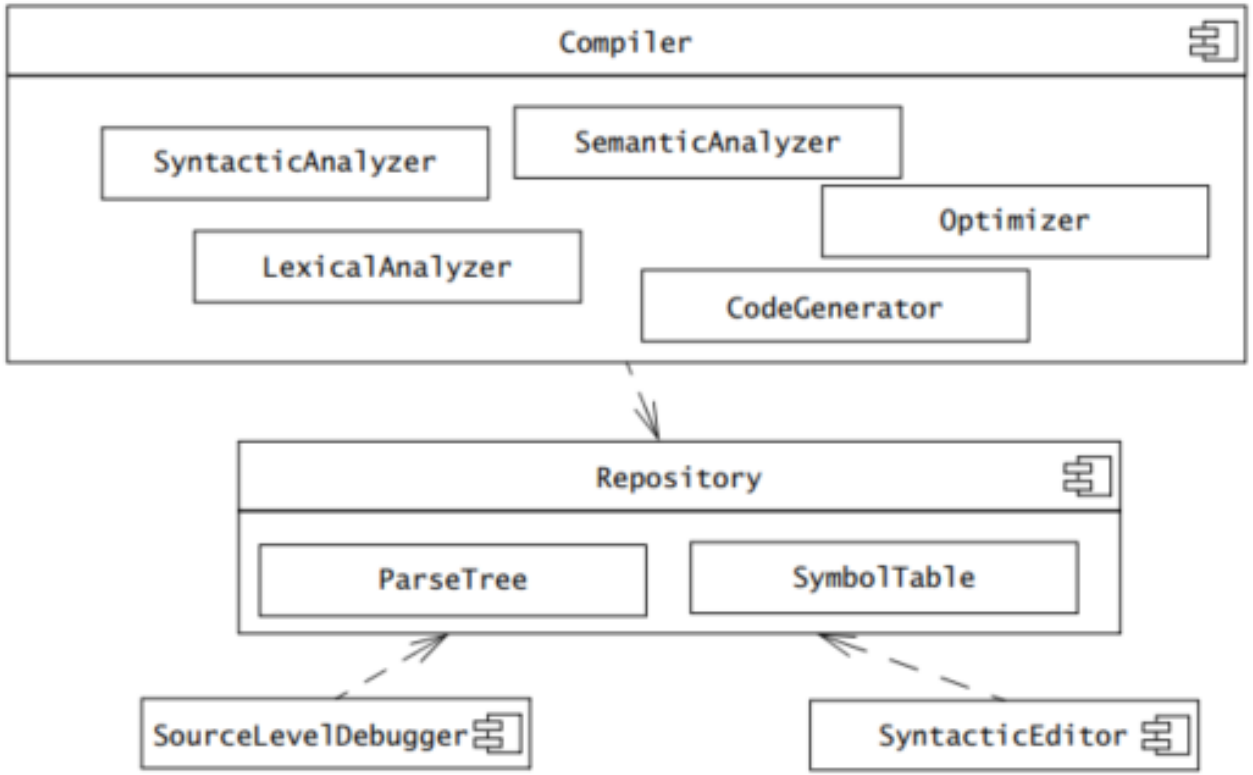
Repositório (2)

- Normalmente usados para sistemas de gerenciamento de banco de dados, como um sistema de folha de pagamento ou sistema bancário.
  - Facilitam simultaneidade e integridade
- Também usados em compiladores e IDEs
- Também pode ser usado para implementar o **fluxo de controle global.**
  - garante que os acessos simultâneos sejam serializados



Repositório (3)

- ✧ Adequados para aplicativos com tarefas de processamento de dados complexas e em **constante mudança**
  - bem definido, podemos facilmente adicionar novos serviços na forma de subsistemas adicionais
- ✧ A principal desvantagem dos sistemas de repositório é que o repositório central pode rapidamente se tornar um gargalo
  - Gargalo de que?
  - Acoplamento?

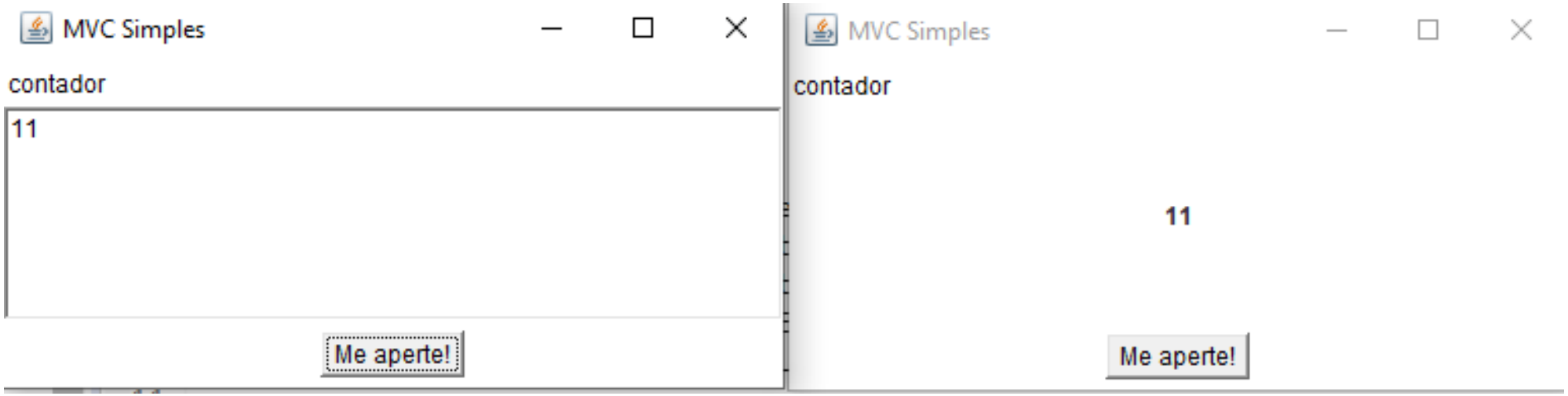


Estilo MVC

- Os subsistemas são classificados em três tipos:
  - Modelo: mantêm o conhecimento do domínio
  - Visão: exibem o modelo para o usuário
  - Controlador: gerenciam a sequência de interações com o usuário
- É um caso especial do repositório
  - Modelo implementa a estrutura central de dados e os objetos de controle determinam o fluxo de controle.

MVC (2)

✧ Exemplo prático no eclipse

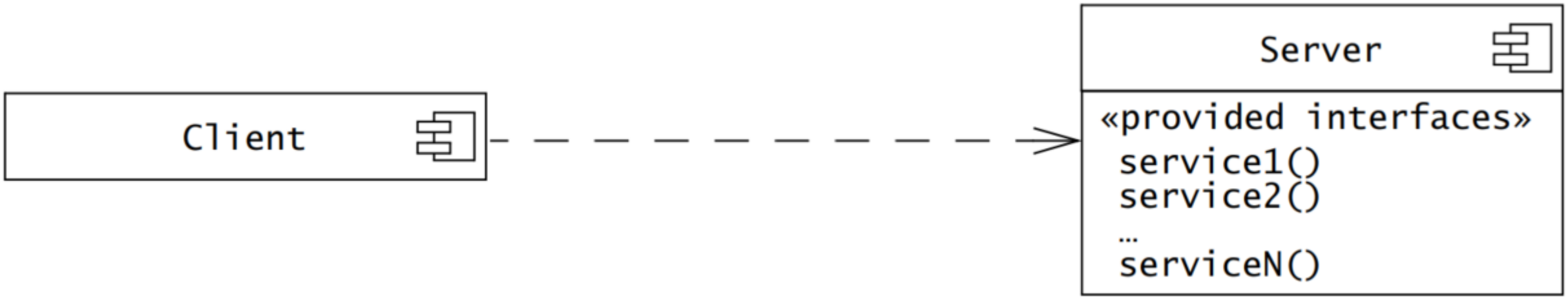


MVC (3)

- A mecânica de **assinatura e notificação** associada a essa sequência de eventos geralmente é realizada com o padrão de design do **Observer. (subscribe/publish)**
- A lógica da separação MVC é que as interfaces de usuário, ou seja, a **View e o Controller, são muito mais sujeitas a alterações** do que o conhecimento de domínio, ou seja, o Model.
- MVC é **adequado para sistemas interativos**, especialmente quando são necessárias **várias visualizações** do mesmo modelo.
- Como **desvantagem**, apresenta o **mesmo gargalo de desempenho** que outros estilos de repositório

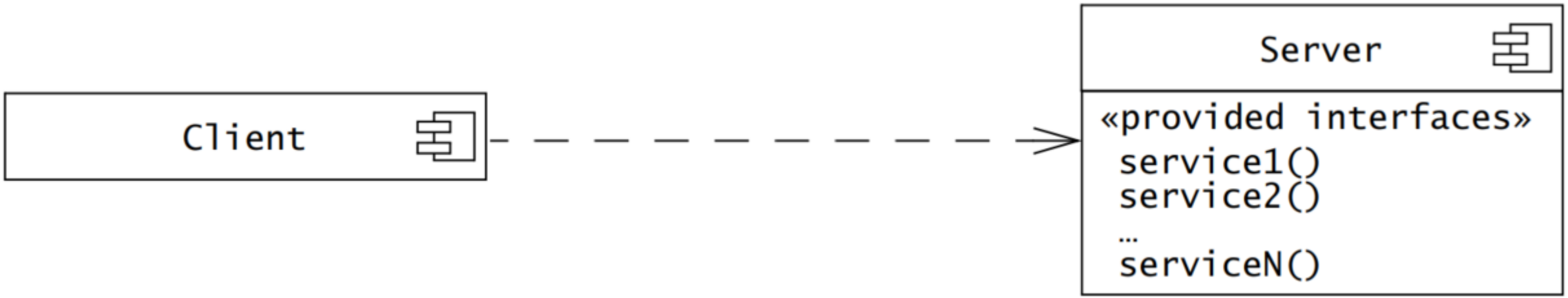
Estilo cliente/servidor

- ✧ um subsistema, o **servidor**, **fornece serviços** para instâncias de outros **subsistemas chamados clientes**, responsáveis por interagir com o usuário.
- ✧ A solicitação de um serviço geralmente é feita por meio de um mecanismo de **chamada de procedimento remoto** ou um intermediador de objeto comum (por exemplo, CORBA, Java RMI ou HTTP)
- ✧ O **fluxo de controle nos clientes e nos servidores é independente**, exceto pela sincronização para gerenciar solicitações ou receber resultados.



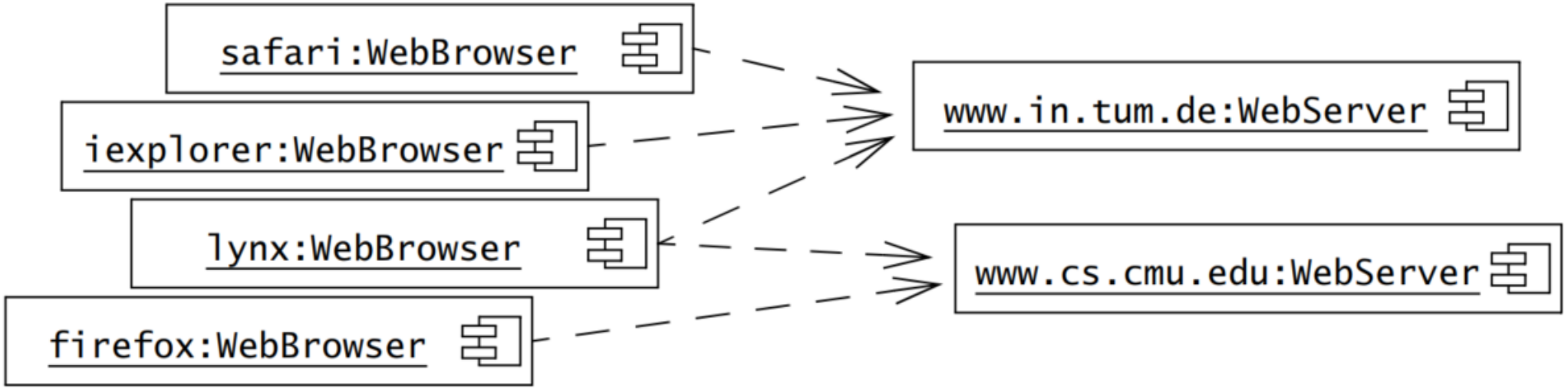
cliente/servidor (2)

- ✧ Um exemplo é sistema de informações com um banco de dados central.
- ✧ **Os clientes são responsáveis** por receber entradas do usuário, executar verificações de intervalo e iniciar transações do banco de dados quando todos os dados necessários forem coletados.
- ✧ **O servidor é responsável** por executar a transação e garantir a integridade dos dados.



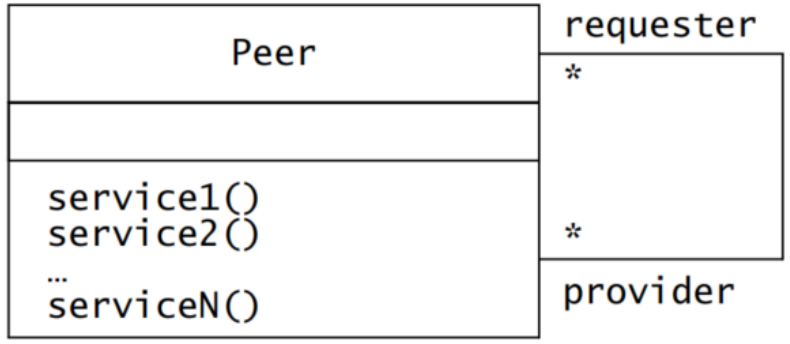
cliente/servidor (3)

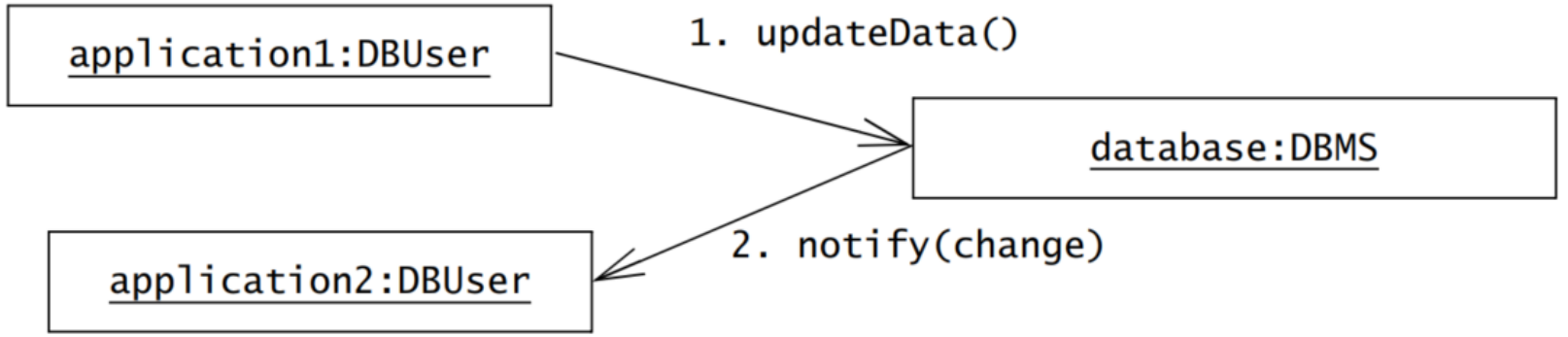
- A arquitetura cliente / servidor é um caso especial do estilo repositório no qual a estrutura de dados central é gerenciada por um processo.
- No entanto, não há restrição a um único servidor.
  - Na World Wide Web, um único cliente pode acessar facilmente dados de milhares de servidores diferentes
- Os estilos arquiteturais de cliente / servidor são adequados para sistemas distribuídos que gerenciam grandes quantidades de dados.



Estilo ponto a ponto (peer-to-peer/P2P)

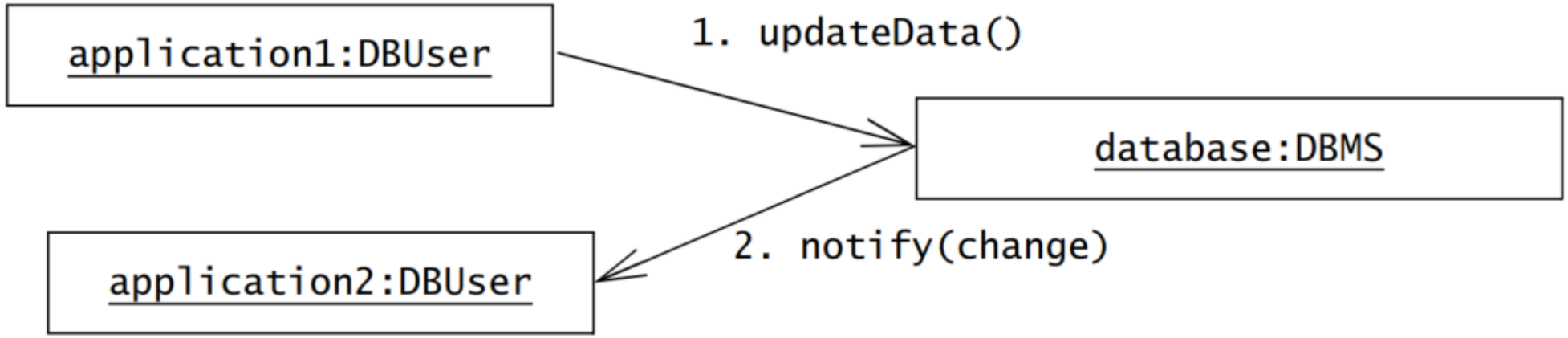
- ✧ É uma **generalização do estilo de arquitetura cliente / servidor** no qual os **subsistemas podem atuar como cliente ou como servidores,** no sentido de que cada subsistema pode solicitar e fornecer serviços.
- ✧ O **fluxo de controle dentro de cada subsistema é independente** dos outros, exceto para sincronizações nas solicitações.
- ✧ Um exemplo é um sistema de vídeo conferência onde cada câmera é servidor para todos os outros usuários, e todo usuário é cliente para todo câmera.





Ponto a ponto (peer-to-peer/P2P) (2)

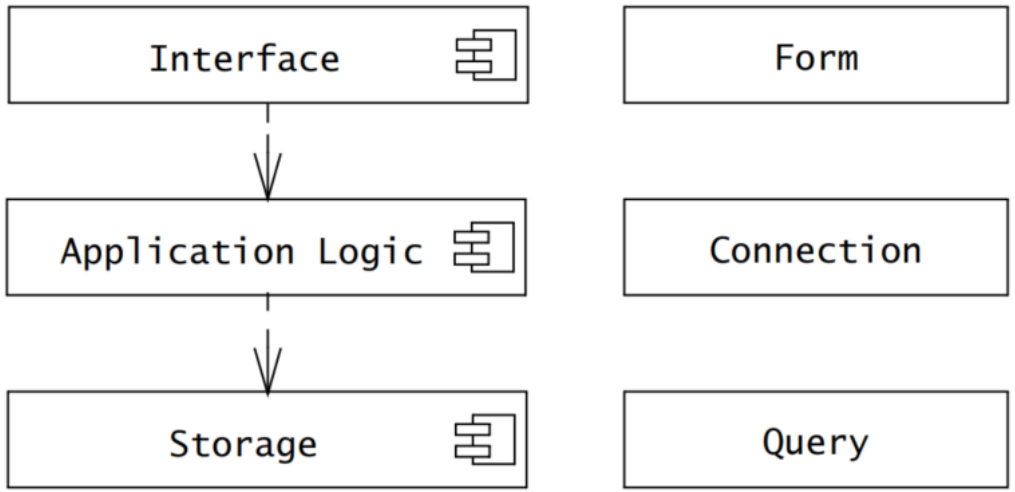
- São **mais difíceis de projetar** do que os sistemas cliente / servidor porque eles introduzem a possibilidade de **deadlocks** e complicam o fluxo de controle.
- **Callbacks** são operações temporárias e personalizadas para uma finalidade específica. Por exemplo,
  - um par DBUser pode dizer ao par DBMS **qual operação chamar (função/método) após uma notificação de alteração**.
  - **O DBMS usa a operação de retorno de chamada especificada** por cada DBUser para notificação quando ocorre uma alteração.



Estilo três camadas

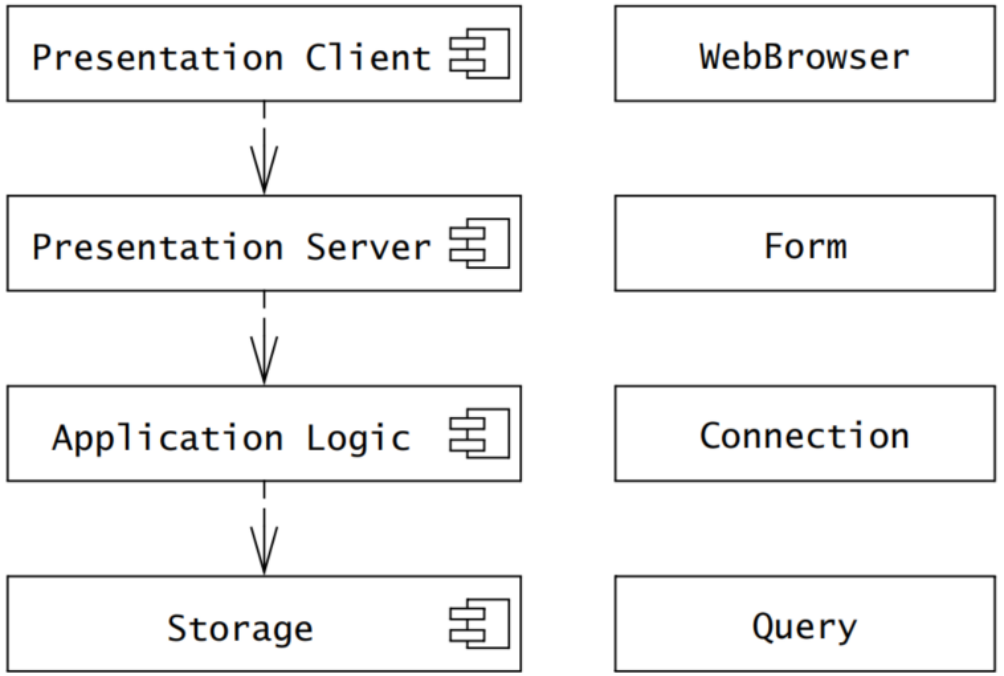
- Subsistemas em 3 camadas:
  - A camada da interface inclui todos os objetos de limite que lidam com o usuário, incluindo janelas, formulários, páginas da web etc.
  - A camada lógica do aplicativo inclui todos os objetos de controle e entidade, realizando o processamento, a verificação de regras e a notificação exigida pelo aplicativo.
  - A camada de armazenamento realiza o armazenamento, a recuperação e a consulta de objetos persistentes.

A camada de **armazenamento**, análoga ao Repositório, **pode ser compartilhada** por vários aplicativos diferentes que operam nos mesmos dados.
A separação entre a camada da interface e a camada lógica da aplicação permite o **desenvolvimento de diferentes UI.**



Estilo quatro camadas

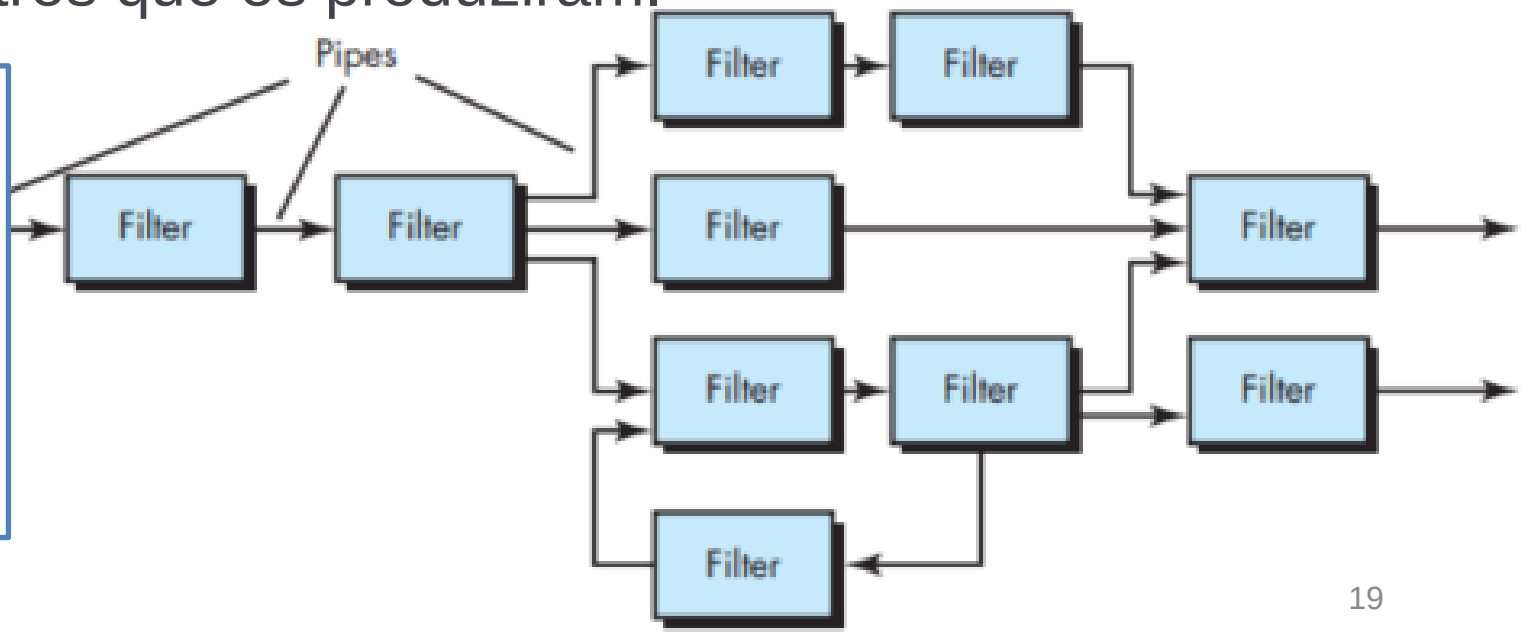
- É uma arquitetura de três camadas, na qual a camada Interface é decomposta em uma camada **Apresentação do Cliente (usuários)** e uma do **Apresentação do Servidor (servidores)**.
  - permite uma ampla variedade de **clientes de apresentação diferentes no aplicativo**, enquanto reutiliza alguns dos objetos de apresentação entre os clientes.
  - Exemplo: bancos com interface de navegador da Web, Caixa Eletrônico e um cliente de aplicativo para funcionários do banco



Duto e filtro (pipe and filter)

- os subsistemas processam dados recebidos de um conjunto de entradas e enviam resultados para outros subsistemas por meio de um conjunto de saídas.
- Os subsistemas são chamados de "**filtros**" e as associações entre os subsistemas são chamadas de "**dutos**"
- Cada filtro conhece apenas o conteúdo e o formato dos dados recebidos nos canais de entrada, não os filtros que os produziram.

Cada filtro é executado concorrentemente e a sincronização é realizada por meio dos dutos.
O estilo é modificável: os filtros podem ser substituídos por outros ou reconfigurados para atingir um objetivo diferente

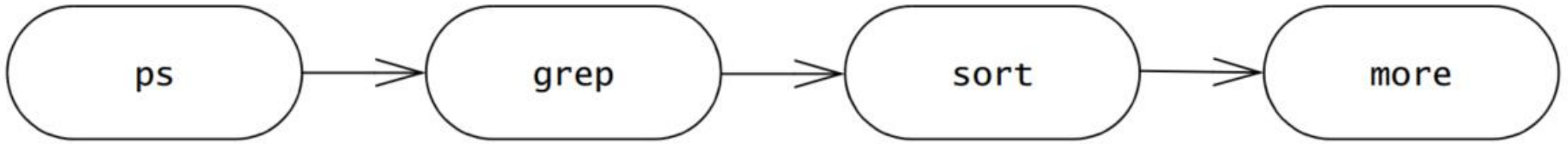


Estilo duto e filtro (pipe and filter) (2)

- ✧ O exemplo mais conhecido do estilo é o shell Unix [Ritchie & Thompson, 1974]
- ✧ A maioria dos filtros é escrita de forma que eles leiam suas entradas e escrevam seus resultados em dutos padrão.
  - Isso permite que um usuário Unix os combine de várias maneiras diferentes. Exemplo abaixo.
- ✧ O estilo é adequado para sistemas que aplicam **transformações a fluxos de dados sem intervenção dos usuários**.
  - não são adequados para sistemas que exigem interações mais complexas entre componentes, como um sistema de gerenciamento de informações ou um sistema interativo

```
% ps auxwww | grep dutoit | sort | more

dutoit   19737  0.2  1.6 1908 1500 pts/6    O 15:24:36  0:00 -tcsh
dutoit   19858  0.2  0.7  816  580 pts/6    S 15:38:46  0:00 grep dutoit
dutoit   19859  0.2  0.6  812  540 pts/6    O 15:38:47  0:00 sort
```



Tópicos

- ✧ Visão geral
- ✧ Estilos
- ✧ Atividades

Exercícios

- ✧ Você está desenvolvendo um sistema que armazena seus dados em um sistema de arquivos Unix. Você antecipa que irá portar versões futuras do sistema para outros sistemas operacionais que fornecem sistemas de arquivos diferentes. Baseado em um estilo de arquitetura, proponha uma decomposição do subsistema que antecipe essa mudança. Descreva o estilo, vantagens e desvantagens.

Exercícios

- Sugira uma arquitetura para um sistema como o Spotify, usado para distribuir música na Internet. Considere as seguintes funcionalidades:
  - Autenticar de usuários
  - Tocar músicas
  - Baixar músicas para a aplicação (arquivo protegido)
  - Marcar músicas como favoritas e listas as favoritas
  - Listas de músicas geradas por inteligência artificial baseada nas músicas favoritas e músicas ouvidas
  - Perfil de usuário com divulgação automática das músicas ouvidas
  - Interface de usuário pelo navegador, aplicativo de Windows, aplicativo de Android e aplicativo de IOS.
  - Usuários podem publicar seus áudios como podcasts.
  - Usuários podem fazer transmissão ao vivo de áudio (live de áudio)
- Liste os estilos de arquitetura que pretenda usar e desenhe uma arquitetura buscando reduzir o acoplamento. Use como base a descrição de componentes do Sistema de Bombeiros apresentado nos slides do Tema 1, e também os componentes dos estilos de arquitetura que queira usar. Arranje os componentes conforme os estilos de arquitetura que escolher.
- Desenvolva sua resposta no draw.io e exporte em um arquivo PDF.

Mãos na massa

- ✧ Professor faz e alunos copiam.
- ✧ Demonstração de MVC.

Referências

- BRUEGGE, B; DUTOIT, A. Object-oriented software engineering: using UML, patterns and Java. Harlow: Pearson Education, 2014.
- PRESSMAN, R. Engenharia de Software. [Recurso eletrônico, Minha Biblioteca]. 8ª ed. BOOKMAN, 2016.
- SOMMERVILLE, I. Engenharia de Software. [Recurso eletrônico, Biblioteca Virtual Universitária 3.0]. 9ª ed. SARAIVA, 2011
- DEITEL, P. Java: Como Programar. Pearson, 2016.