Estilos Arquiteturais

Centro de Informática - Universidade Federal de Pernambuco
Engenharia da Computação
Kiev Gama
kiev@cin.ufpe.br

Slides elaborados pelo professor Marcio Cornélio

UNIVERSIDADE FEDERAL DE PERNAMBUCO

Estilos Arquiteturais

- A arquitetura de um sistema pode aderir a um ou mais estilos arquiteturais
  - Um estilo define os tipos de elementos que podem aparecer em uma arquitetura e as regras que regem a sua interconexão
- Esses estilos pode simplificar o problema de definição de arquiteturas de sistema.
- A maioria dos sistemas de grande porte adere a vários estilos
- Estilos arquiteturais = "modelos arquiteturais"

Exemplos de Estilos Arquiteturais

- Cliente-Servidor
- “Em camadas”
- Filtros e dutos (*pipes and filters)*
- *Baseado em repositório*
- *Orientado a eventos (publisher/subscriber)*
- *Transferência Representacional de Estado (REST)*
- *Objetos distribuídos*
- *C2*

Estilos Arquiteturais e Escolhas de Projeto

- Um estilo arquitetural representa um conjunto de escolhas de projeto
  - Conjunto de características comuns a diversos sistemas nos quais as mesmas escolhas foram feitas
    - Padrões arquiteturais
  - Um sistema aderente a determinado estilo “ganha" as características inerentes a ele
- Estilos podem ser usados para descrever uma determinada arquitetura
  - Foco nas soluções de projeto e não em sua documentação

Organização de sistema

- Reflete a estratégia básica que é usada para estruturar um sistema
- Exemplos:
  - O estilo de repositório de dados compartilhados;
  - Estilo de serviços e servidores compartilhados;
  - Estilo de máquina abstrata ou em camadas
  - Orientado a objetos (ou Objetos Distribuídos)
  - *Pipes and Filters ou Pipelining*
- *Classificação para fins de estudo*

Estilo de repositório

- Sistemas cujas partes precisam trocar dados com frequência:
  - Dados compartilhados podem ser mantidos em um banco de dados central e acessados por todos os subsistemas
  - Cada subsistema mantém seu próprio banco de dados e passa dados para outros subsistemas
    - Podem usar uma abstração de repositório centralizado
    - Implementação distribuída

Arquitetura de conjunto de ferramentas CASE

Figura 11.2

Arquitetura de um conjunto de ferramentas CASE integradas.

Características do Estilo Arquitetural de Repositório

- Vantagens
  - É uma maneira eficiente de compartilhar grandes quantidades de dados
  - Dados aderem a uma representação comum
  - Simplifica a projeto de aplicações fortemente baseadas em dados
    - Tanto para troca de info. quanto para armazenamento
- Desvantagens
  - Os subsistemas devem estar de acordo com um modelo de dados padronizado
  - A evolução de dados é difícil e dispendiosa;
  - Dificuldade para distribuir de forma eficiente.

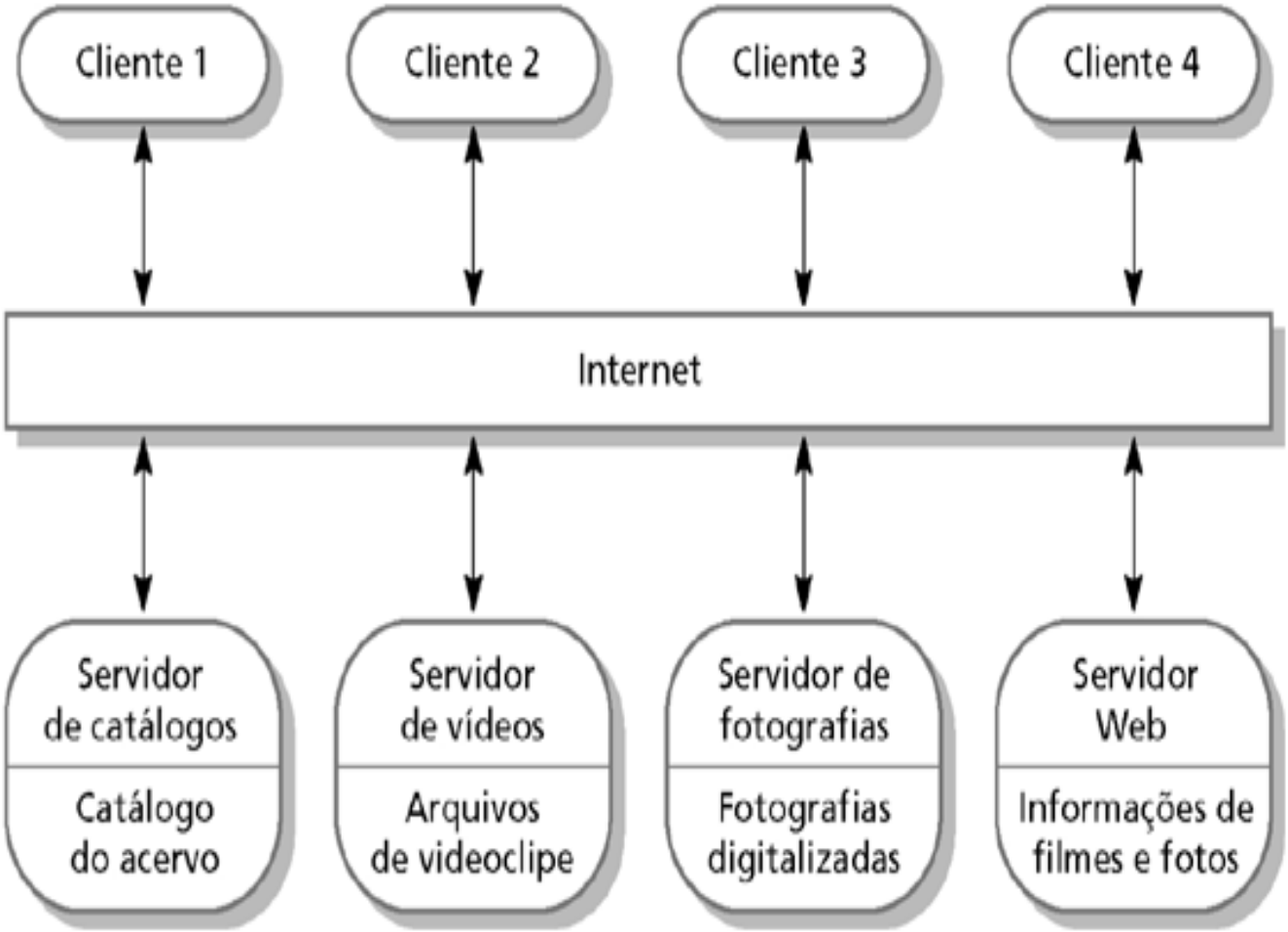
Estilo Cliente-Servidor

- Mostra como dados e processamento são distribuídos por uma variedade de componentes.
- Servidores independentes que fornecem serviços tais como impressão, transferência de arquivos, gerenciamento de dados, etc.
- Clientes utilizam esses serviços.
- Clientes e servidores normalmente se comunicam através de uma rede
  - Diversas tecnologias de comunicação são possíveis

Biblioteca de filmes e fotografias

Figura 11.3

Arquitetura de um sistema de acervo de filmes e fotografias.



Características do Estilo Cliente-Servidor

- Vantagens
  - Separação de interesses
  - Inerentemente distribuído
    - Balanceamento de carga, tolerância a falhas
  - É fácil adicionar novos servidores ou atualizar servidores existentes.
- Desvantagens
  - Gerenciamento redundante em cada servidor;
  - Nenhum registro central de nomes e serviços – pode ser difícil descobrir quais servidores e serviços estão disponíveis
  - Requisições e respostas casadas

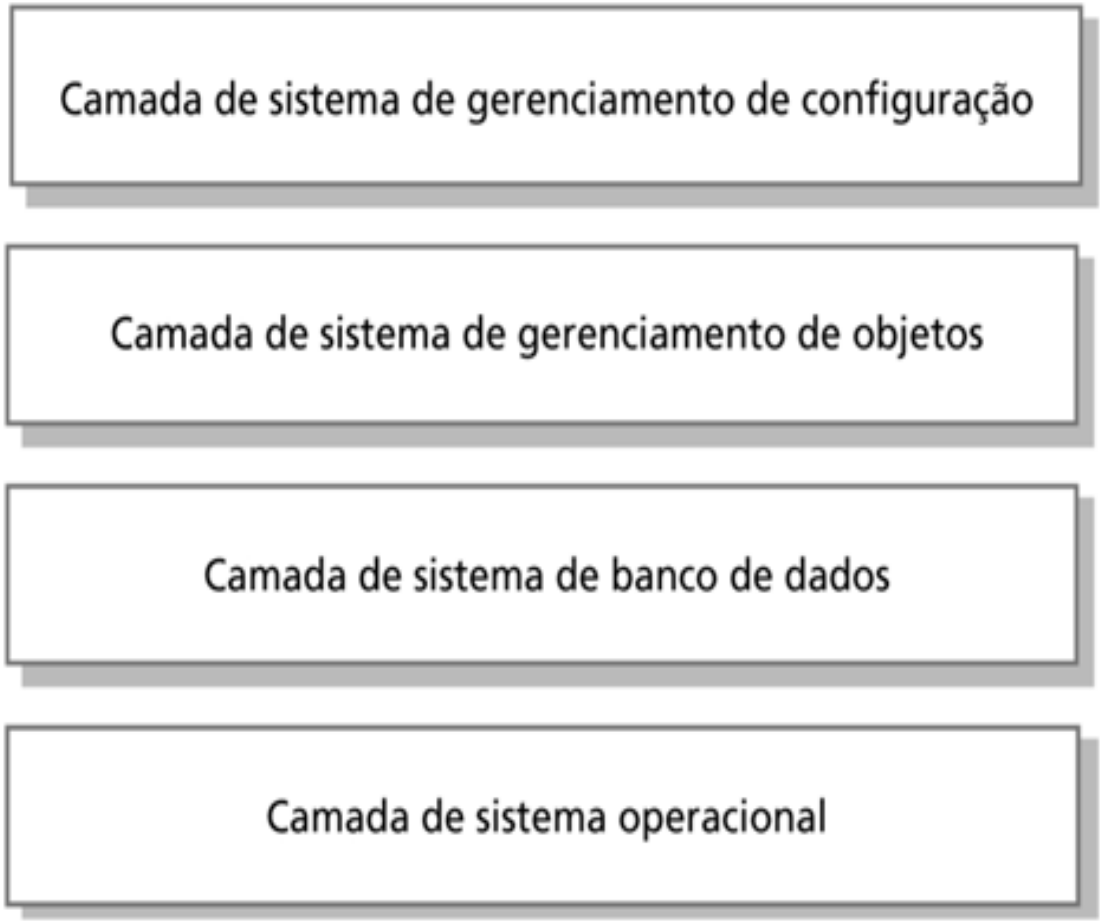
Modelo de Máquina Abstrata (Em Camadas)

- Organiza o sistema em um conjunto de camadas (ou máquinas abstratas)
  - Cada uma fornece um conjunto de serviços
  - Cada camada é cliente da camada subjacente
- Generalização do estilo Cliente-Servidor
  - Não precisa ser distribuído
- Apóia o desenvolvimento incremental dos subsistemas em camadas diferentes.
  - Ex. Se mudarmos a camada de negócios, só as camadas acima precisam ser modificadas

Sistema de gerenciamento de versões

Figura 11.4

Modelo em camadas de um sistema de gerenciamento de versões.



Características do Estilo em Camadas

- Vantagens
  - Facilidade de compreensão
  - Facilidade de manutenção
  - Desenvolvimento independente
- Desvantagens
  - Duplicação de funcionalidade
  - Às vezes é difícil estruturar um sistema através de camadas
    - É comum que a estruturação seja violada
  - *Overhead* de implementação e desempenho

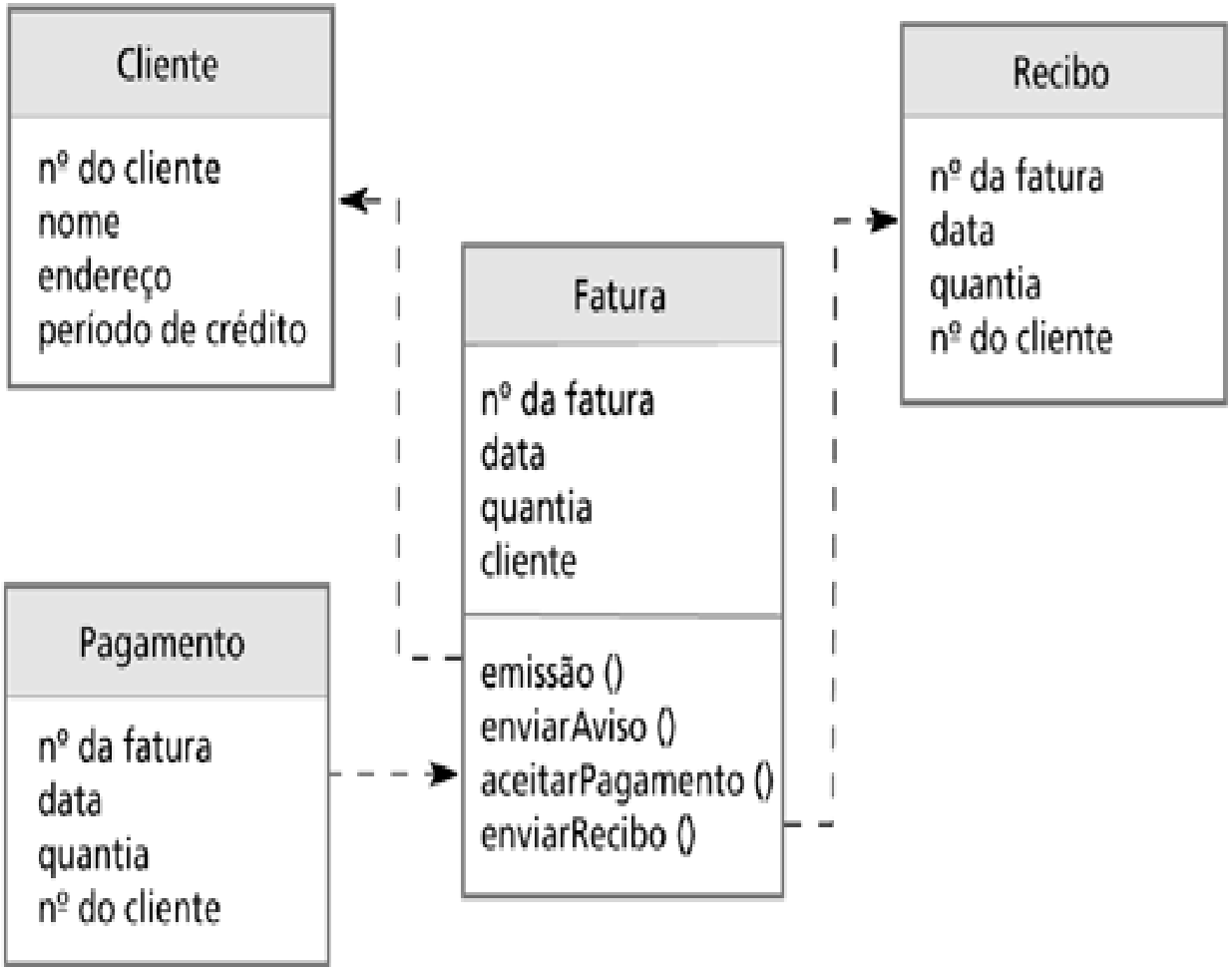
Estilo Arquitetural de Objetos

- Sistema como um conjunto de objetos fracamente acoplados e com interfaces bem definidas
  - Cada objeto oferece um conjunto de serviços
- No nível arquitetural, é frequentemente empregado na construção de sistemas distribuídos
  - Objetos distribuídos
- Uma implementação OO não implica em uma arquitetura OO

Sistema de processamento de faturas

Figura 11.5

Modelo de objeto de um sistema de processamento de faturas.



Características do Estilo Arquitetural de Objetos

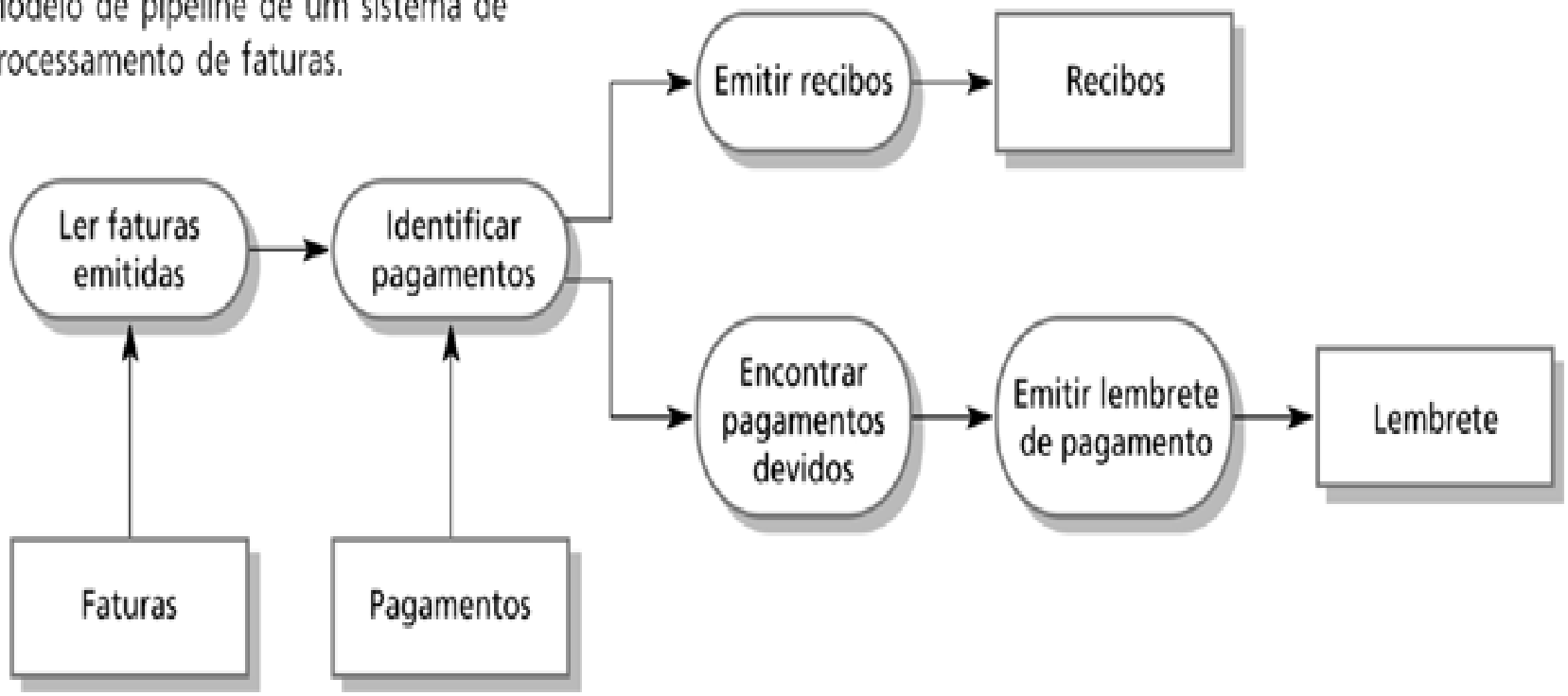
- Vantagens
  - Objetos são fracamente acoplados devido ao uso de interfaces
  - Linguagens de implementação orientada a objeto são amplamente usadas.
- Desvantagens
  - Mudanças de interface têm alto impacto
  - Não envolve restrições topológicas, o que pode dificultar a manutenção
    - Dependências entre objetos não são limitadas

Estilo Dutos e Filtros (*Pipelining*)

- Originário de sistemas operacionais UNIX e do projeto de compiladores
- Transformações funcionais processam entradas para produzir saídas.
  - Componentes são chamados de filtros
  - Conectores são dutos (*pipes*)
- Útil para aplicações de processamento de informação que interagem pouco com usuários
- Variação distribuída: processamento de *streams*

Sistema de processamento de faturas

Figura 11.6 Modelo de pipeline de um sistema de processamento de faturas.



Características do Estilo Dutos e Filtros

- Vantagens
  - Apóia reuso de transformações.
  - É fácil adicionar novas transformações.
  - É relativamente simples implementar como sistema concorrente ou seqüencial.
- Desvantagens
  - Requer um formato comum para a transferência de dados ao longo do *pipeline*
  - Não é apropriado para aplicações interativas
    - Mais especificamente: só é apropriado para realizar processamento sequencial

Fluxo de Controle

- Perspectiva de estilos arquiteturais em relação ao fluxo de controle entre os componentes arquiteturais
  - Controle centralizado
    - Um subsistema tem responsabilidade global pelo controle e inicia e pára outros sistemas
  - Controle baseado em eventos
    - Cada componente responde a eventos gerados por outros subsistemas

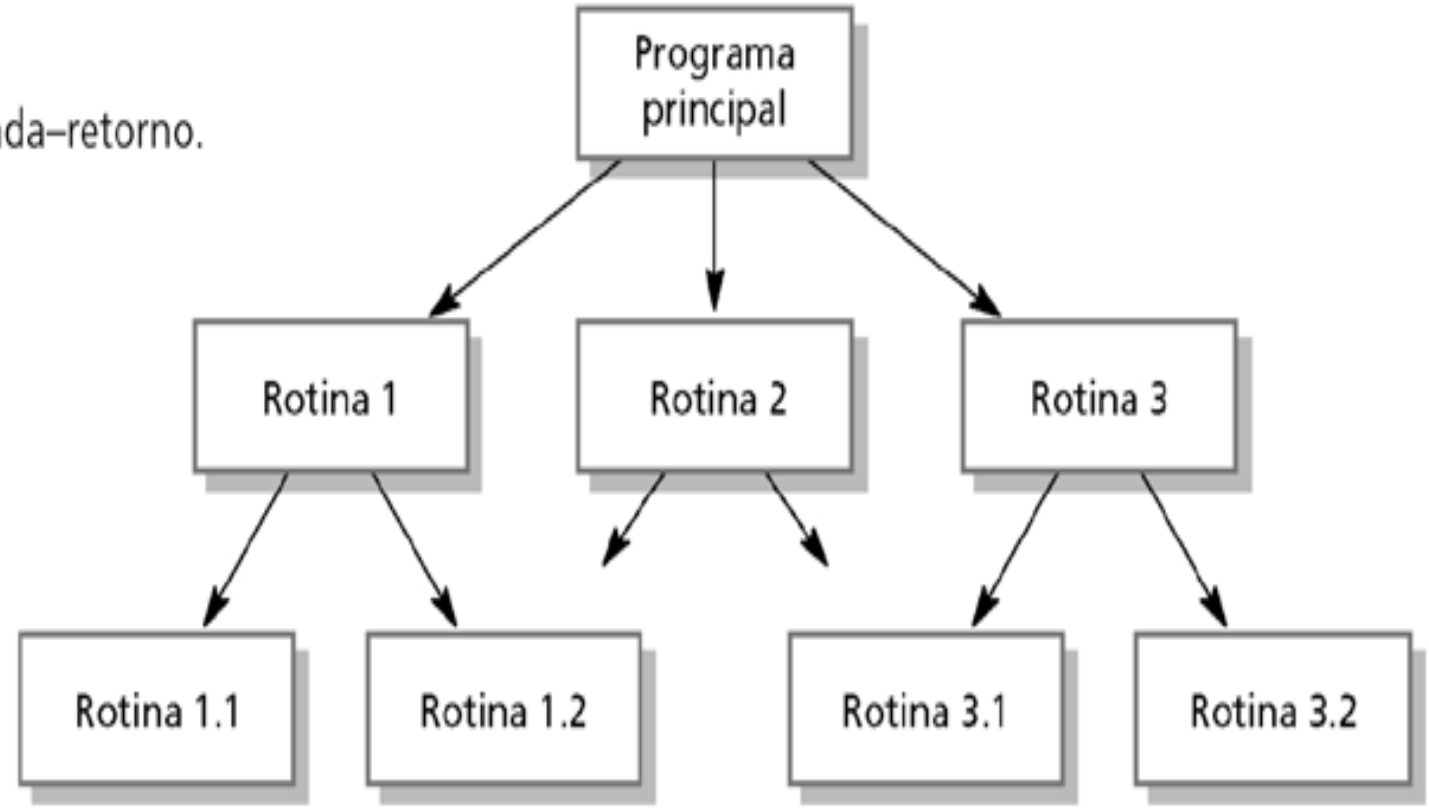
Controle centralizado

- Um componente é responsável pelo gerenciamento da execução de outros componente.
  - O estilo Chamada-Retorno
    - Controle se inicia no topo de uma hierarquia de subrotinas e move-se para baixo na hierarquia.
    - Pode ser sequencial ou concorrente
  - O estilo de Gerenciador
    - Aplicável a sistemas concorrentes e de tempo real
    - Um componente controla a parada, o início e a coordenação de outros processos de sistema

Chamada-Retorno

Figura 11.7

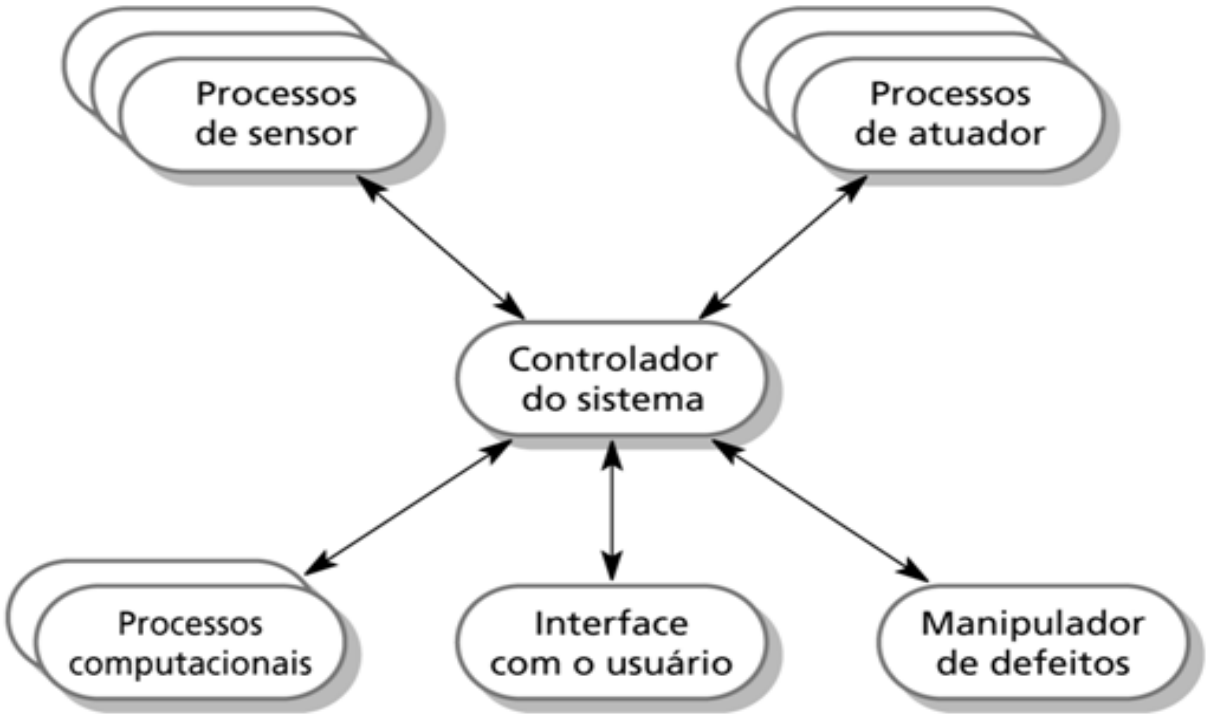
Modelo de controle chamada–retorno.



Gerenciador para um Sistema Tempo Real

Figura 11.8

Modelo de controle centralizado para um sistema de tempo real.



Comunicação entre o Controlador e os outros componentes pode ser baseada em eventos, chamadas de procedimentos, etc.

Sistemas orientados a eventos

- Dirigidos por eventos gerados externamente
  - O *timing* dos eventos está fora do controle dos componentes que os processam
- Estilo *Publisher/Subscriber*
  - Eventos são transmitidos a todos os componentes.
  - Qualquer componente interessado pode respondê-los
- Estilo Orientado a Interrupções
  - Usado em sistemas de tempo real
  - Interrupções são detectadas por tratadores e passadas por outro componente para processamento.

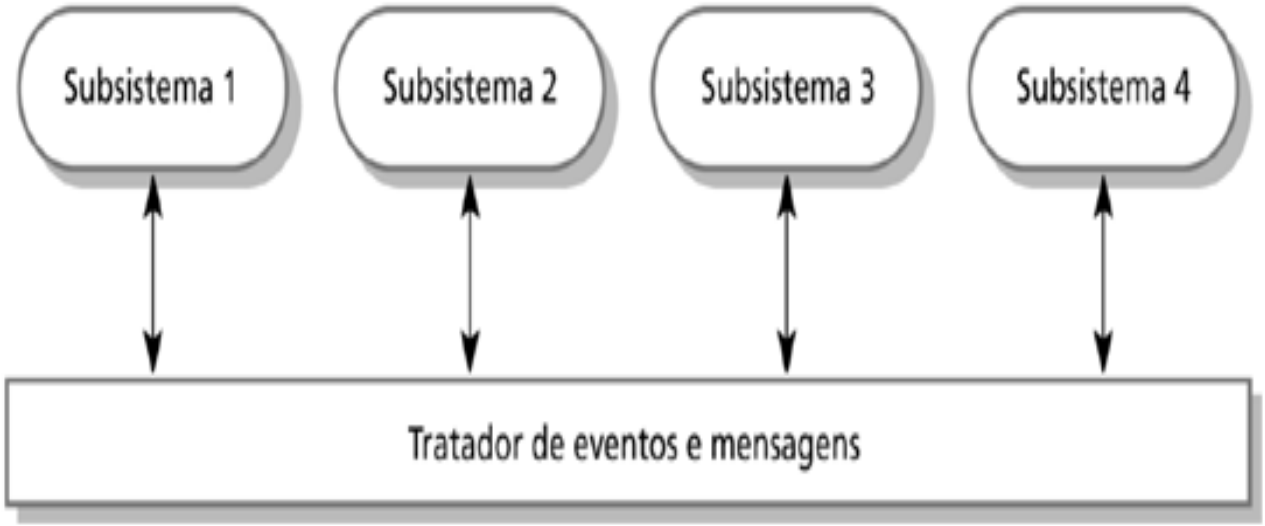
Modelo Publisher/Subscriber

- É efetivo na integração de componentes em computadores diferentes em uma rede
  - Desacoplamento espacial e temporal
  - Componentes não sabem se um evento será tratado e nem quando será.
- Alguns componentes (*publishers*) publicam eventos
- Componentes (*subscribers*) registram interesse em eventos específicos e podem tratá-los
- A política de controle não é embutida no tratador de eventos e mensagens

Publisher/Subscriber

Figura 11.9

Modelo de controle baseado em broadcast seletivo.



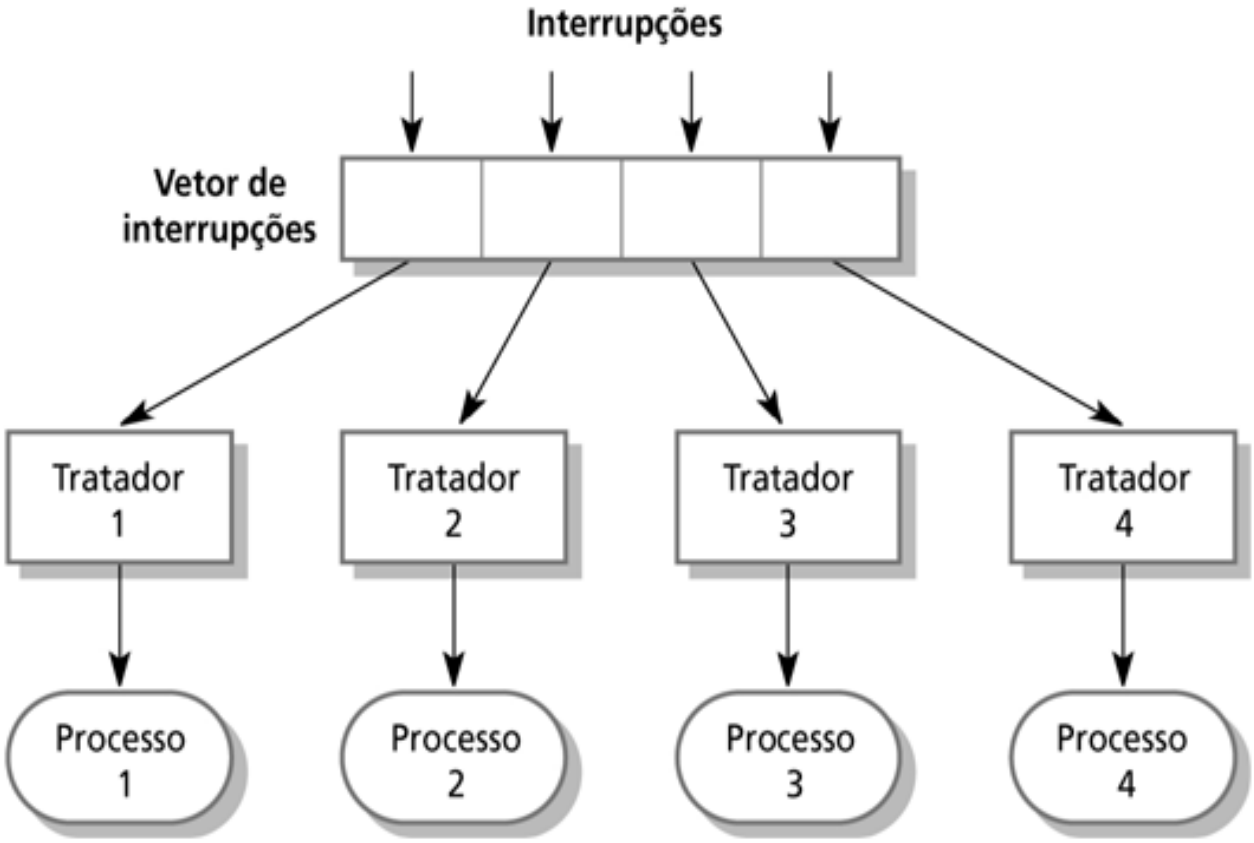
Estilo Orientado a Interrupções

- Usado em sistemas de tempo real onde a resposta rápida para um evento é essencial
- Existem tipos de interrupções conhecidos
  - Um tratador definido para cada tipo
- Cada tipo é associado a uma localização da memória
  - Uma chave de hardware causa a transferência de controle para um tratador.
- Permite respostas rápidas, mas é complexo para programar e difícil de validar.

Controle dirigido a interrupções

Figura 11.10

Modelo de controle orientado a interrupções.



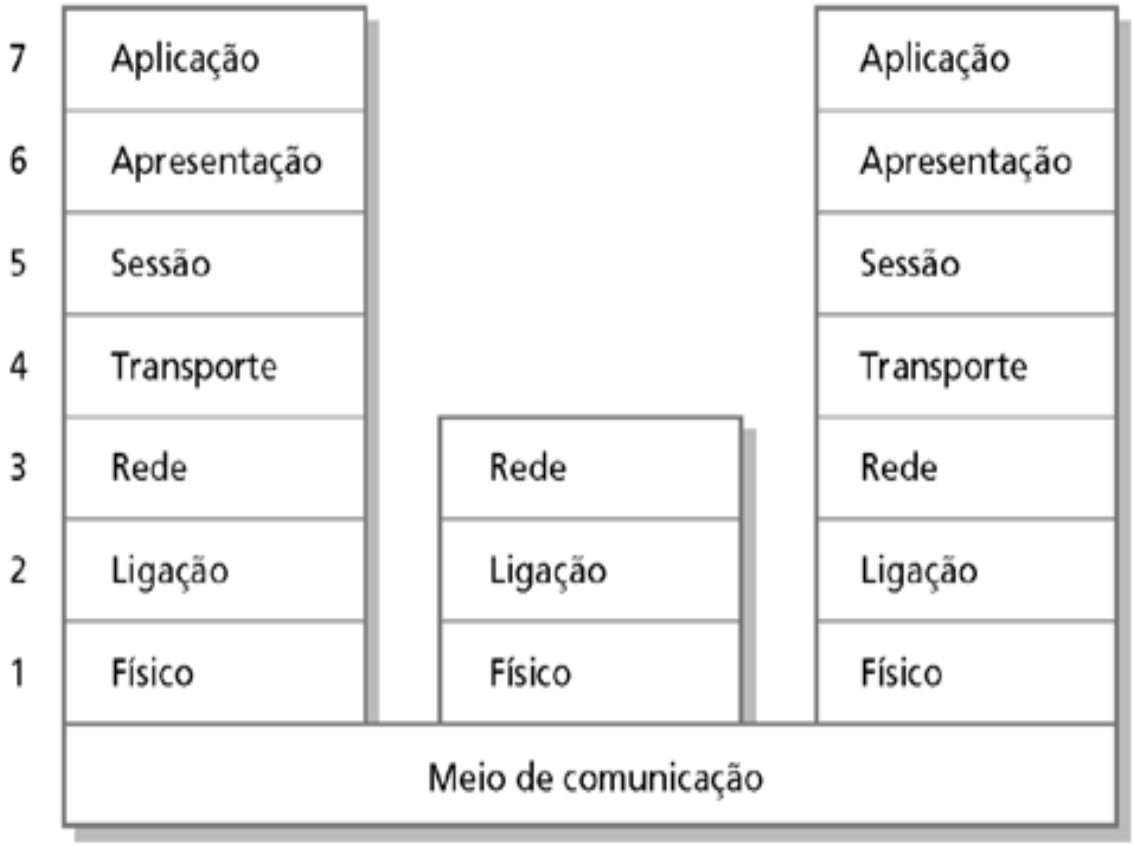
Arquiteturas de Referência

- Derivadas de um estudo de domínio de aplicação, ao invés de sistemas existentes.
- Podem ser usadas como base para a implementação de sistemas (não é o uso principal) ou comparação de sistemas diferentes.
  - Atua como um padrão com relação ao qual os sistemas podem ser avaliados.
- Exs.
  - Modelo OSI para sistemas de comunicação
  - Organização tradicional de compiladores em vanguarda e retaguarda (e seus elementos internos)

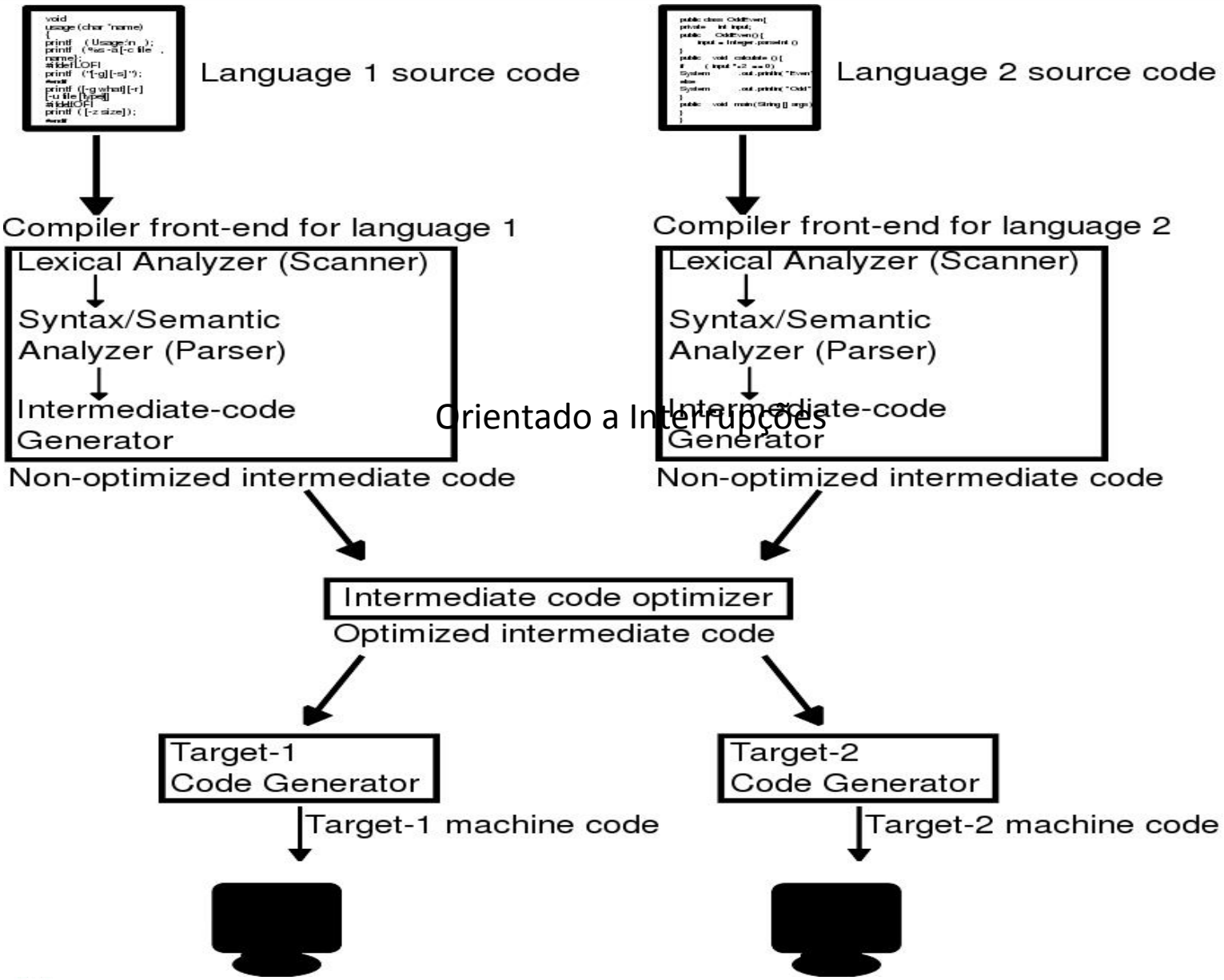
Modelo de referência OSI

Figura 11.11

Arquitetura de modelo de referência OSI.



Arquitetura de um Compilador



Referências

- SOMMERVILLE, I. Engenharia de Software. 9ª. Ed. São Paulo: Pearson Education, 2011
  - Capítulo 11

Projeto de Software Orientado a Objetos

Centro de Informática - Universidade Federal de Pernambuco
Engenharia da Computação
Kiev Gama
kiev@cin.ufpe.br

Slides elaborados pelos professors Fernando Castor e Marcio Cornélio

UNIVERSIDADE FEDERAL DE PERNAMBUCO

Desenvolvimento Orientado a Objetos

- Análise, projeto e programação orientados a objetos estão relacionados, mas são distintos.
- Análise => Modelo de domínio
- Projeto => Modelo para implementar os requisitos
- Programação => materialização do projeto em uma linguagem que o computador entende

Desenvolvimento Orientado a Objetos

- Análise, projeto e programação orientados a objetos estão relacionados, mas são distintos.
- Análise => Modelo de domínio
- Projeto => Modelo para implementar os requisitos
- Programação => materialização do projeto em

O projeto normalmente está dividido em duas etapas:

- Projeto arquitetural (visto nas últimas aulas)
- Projeto detalhado (assunto desta aula)

O que é Projeto Orientado a Objetos?

Conjunto de atividades e artefatos que visa fornecer uma solução para o problema proposto pelos requisitos do sistema

- Envolve a modelagem do sistema
  - Ênfase em resolução de problemas
  - Pensar antes para evitar dificuldades durante a implementação
    - Ignorando “detalhes”
  - De acordo com princípios bem estabelecidos

Características do Projeto Orientado a Objetos

- Objetos são abstrações do mundo real ou entidades de sistema e gerenciam a si próprios
- Os objetos são independentes e englobam estados e informações de representação
- A funcionalidade do sistema é expressa em termos de serviços de objetos
- Áreas de dados compartilhados são eliminadas e os objetos se comunicam por meio da passagem de mensagens
- Objetos podem ser distribuídos e executados seqüencialmente ou em paralelo

Projetar é Difícil!

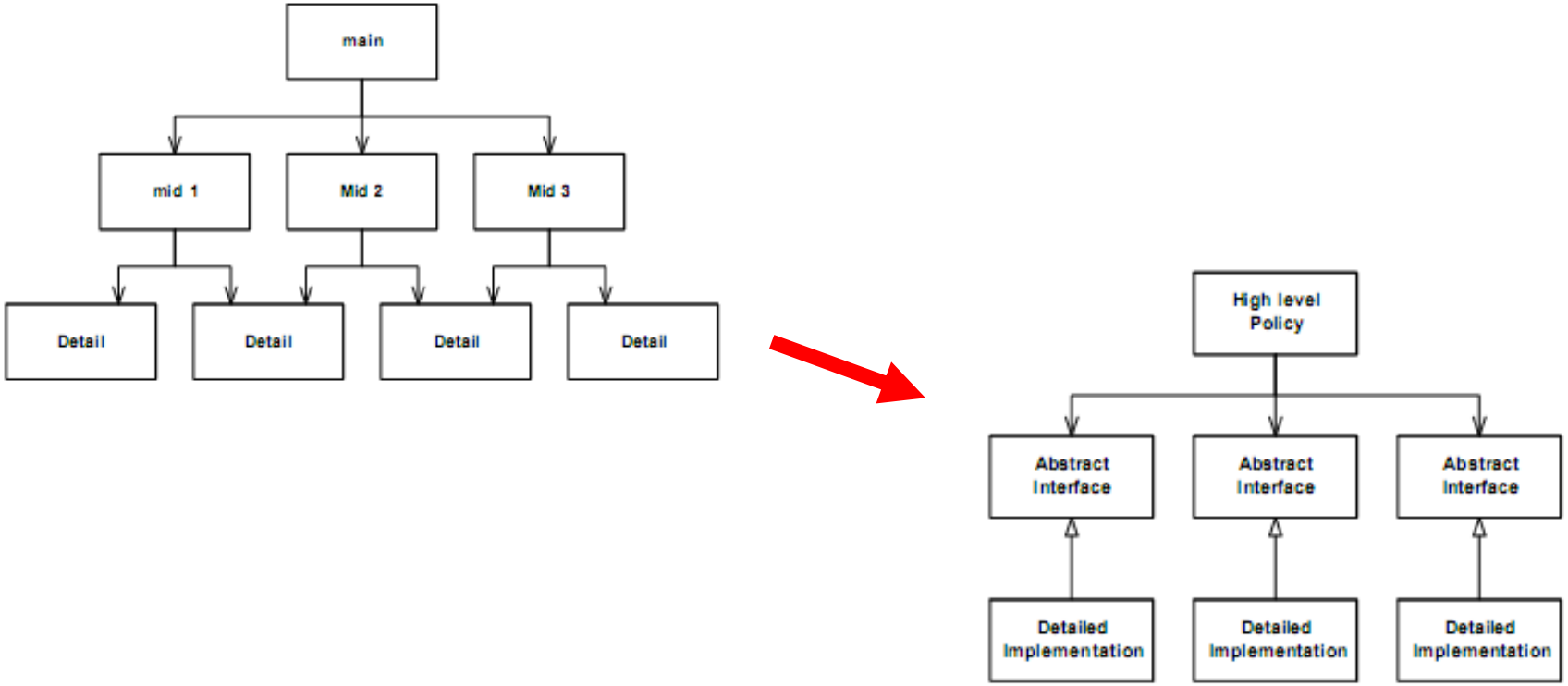
- Projetar envolve criatividade
- Exige a avaliação de diferentes opções
- Requer conhecimento sobre o contexto da aplicação
- Em suma, para cada sistema, um projeto diferente pode ser necessário
- Há soluções e princípios que são recorrentes, porém
  - Quase universalmente aceitos ou
  - Adequados para certas situações
- O foco dessa aula está necesses princípios e soluções gerais

Princípios do Projeto Orientado a Objetos

- **Coesão** alta e **Acoplamento** baixo
  - Classes devem ter um conjunto pequeno e bem-definido de responsabilidades
  - Além disso, devem depender umas das outras o mínimo possível
  - **Implicam em facilidade de manutenção e compreensão**

Princípios do Projeto Orientado a Objetos

- Desenvolva para uma interface e não para uma implementação



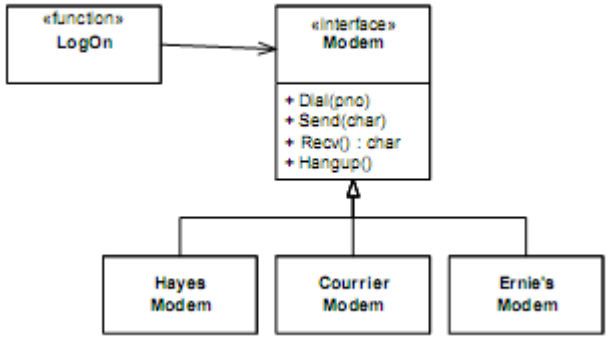
Princípios do Projeto Orientado a Objetos

- Prefira composição de objetos a herança
  - Reuso **caixa branca** vs. reuso **caixa preta**
    - Herança quebra o encapsulamento
  - Maior flexibilidade
    - Estrutura do sistema pode mudar em tempo de execução
  - Maior coesão
    - Classes mais focadas em um único objetivo
  - Composição implica em mais classes, porém

Princípios do Projeto Orientado a Objetos

- Um módulo deve ser aberto para extensão e fechado para modificação

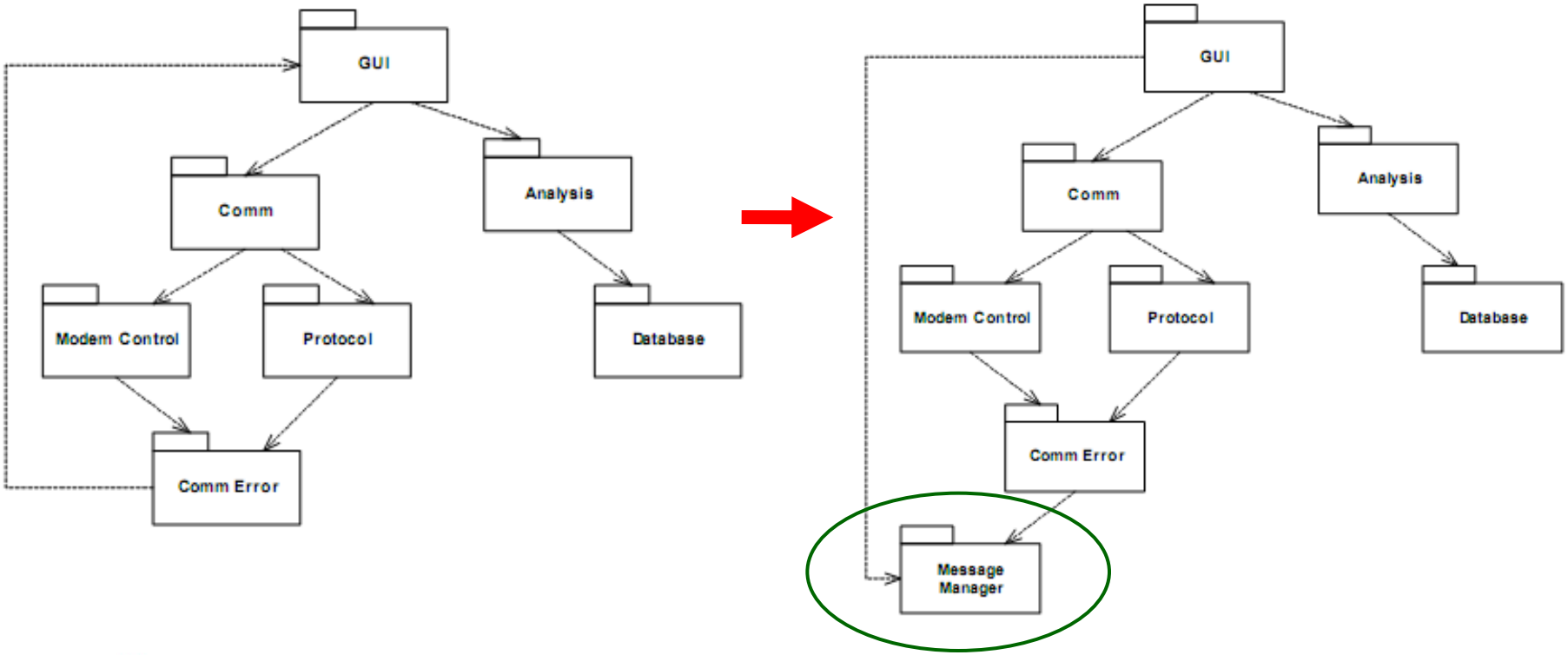
```
void LogOn(Modem& m,
           string& pno, string& user, string& pw)
{
  if (m.type == Modem::hayes)
    DialHayes((Hayes&)m, pno);
  else if (m.type == Modem::courrier)
    DialCourrier((Courrier&)m, pno);
  else if (m.type == Modem::ernie)
    DialErnie((Ernie&)m, pno)
  // ...you get the idea
}
```



- Subclasses devem ser capazes de substituir suas superclasses
  - Elipses vs. Círculos
  - Listas ligadas vs. Pilhas e Filas

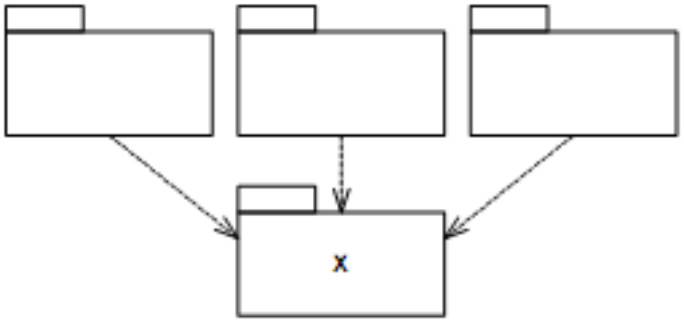
Princípios do Projeto Orientado a Objetos

- Dependências entre pacotes não devem formar ciclos

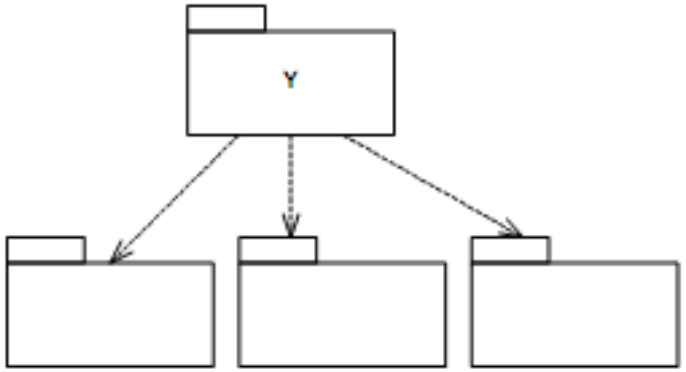


Princípios do Projeto Orientado a Objetos

- Dependa na direção da estabilidade



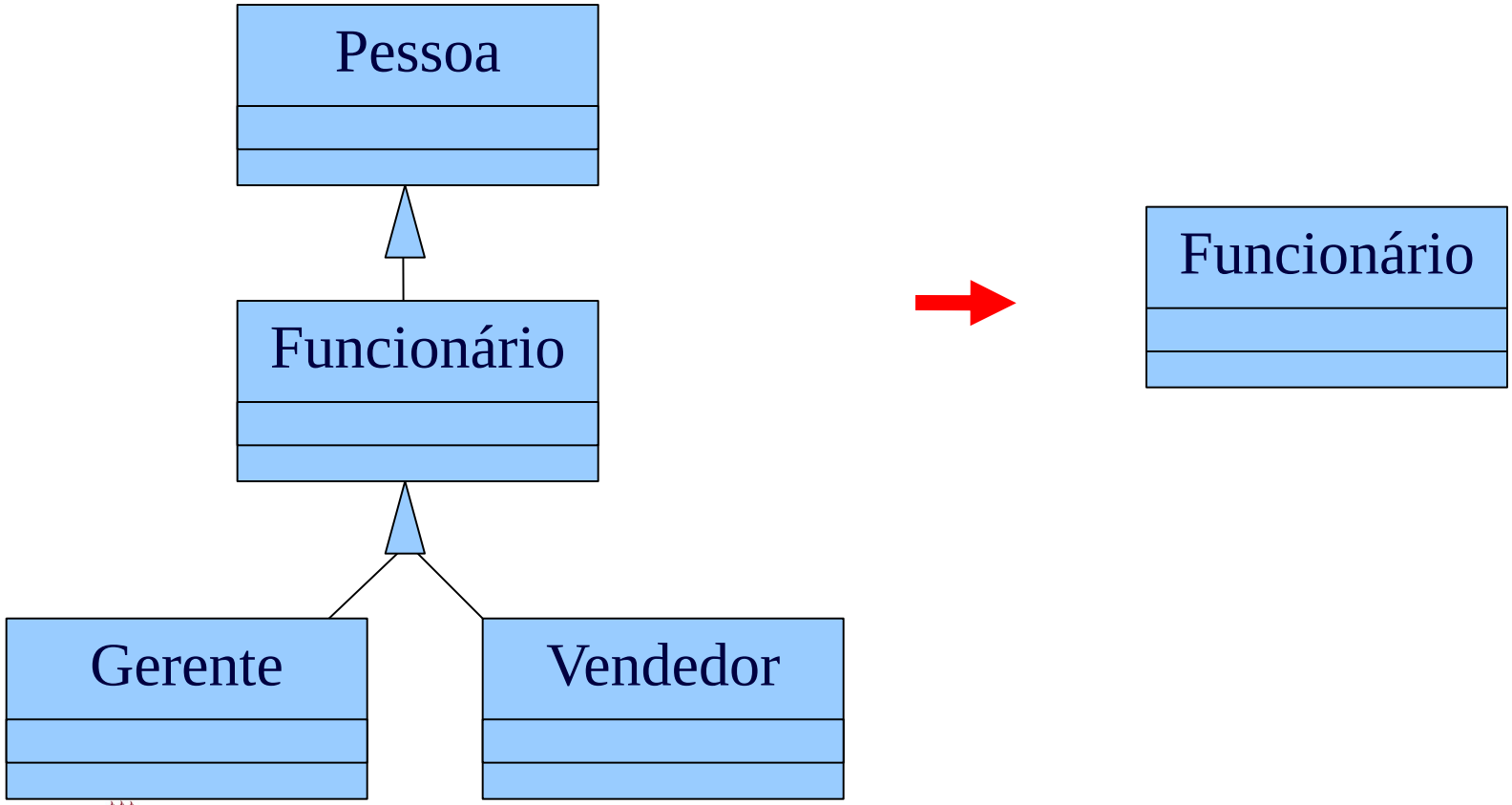
MELHOR!



PIOR!

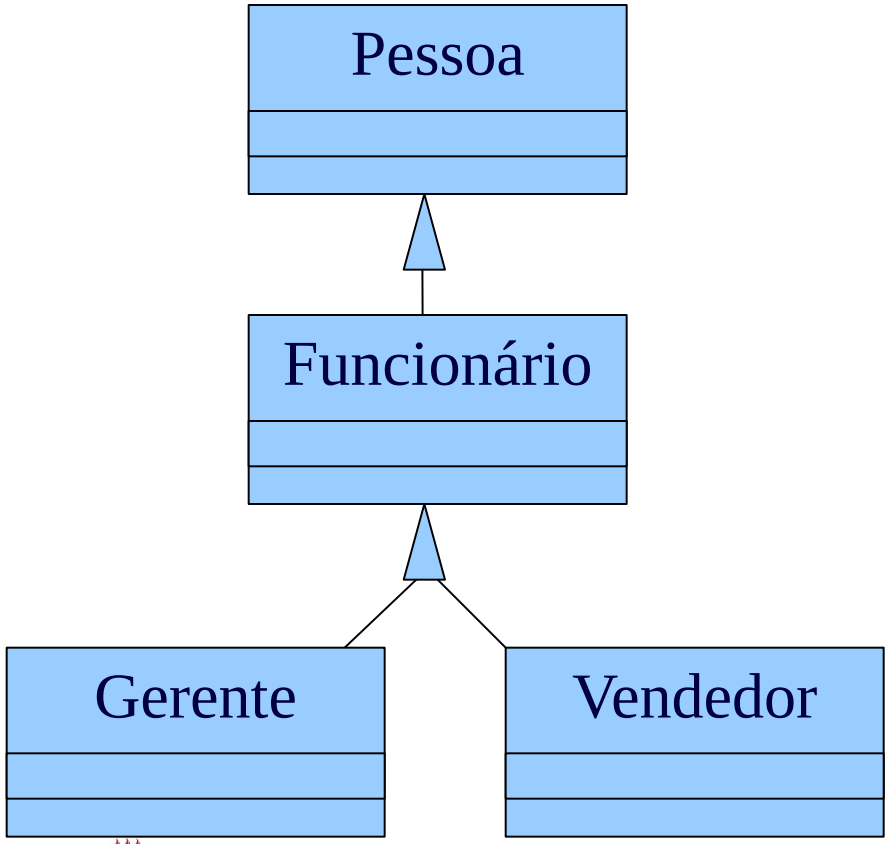
Princípios do Projeto Orientado a Objetos

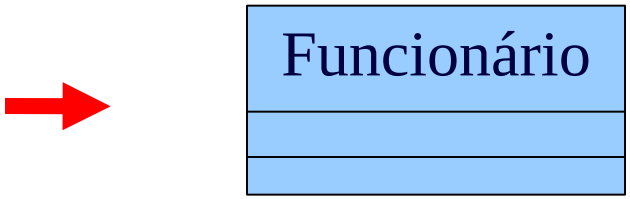
- Crie classes apenas quando necessário



Princípios do Projeto Orientado a Objetos

- Crie classes apenas quando necessário





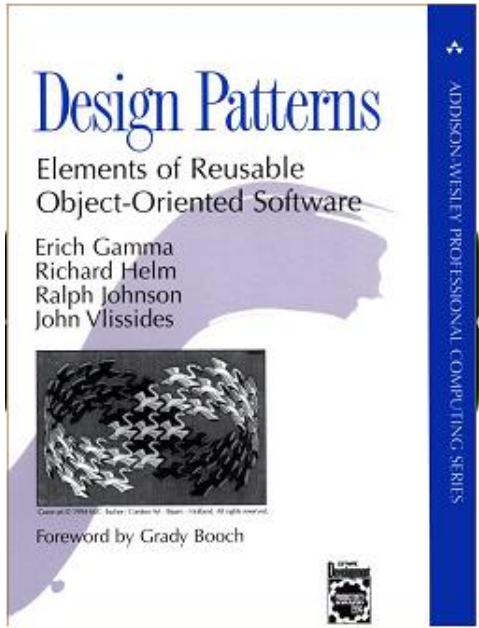
Classes devem encapsular informação que é **manipulada** em conjunto e que **evolui** em conjunto

Padrões de Projeto

- Escritores de livros, histórias em quadrinhos e roteiros raramente inventam novas histórias
- Idéias frequentemente são reusadas
  - "Herói Trágico" => Hamlet, Macbeth
  - "Anti-Herói" => Aquiles, Sawyer, Lobo
- Projetistas também reutilizam soluções, de preferência as boas
  - Experiência é o que torna uma pessoa um *expert*
- Problemas são tratados de modo a evitar a reinvenção de soluções

O que é um Padrão de Projeto?

- Abstração de uma solução de projeto recorrente
- Envolve dependências, estruturas, interações e convenções relativos a classes e objetos
- Documentam experiência no projeto de sistemas de software
- Tornam projetos OO mais flexíveis, elegantes e reusáveis



Definição de Alexander

"... describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

Alexander, Christopher, Sara Ishikawa and Murray Silverstein
A Pattern Language: Towns. Buildings, Construction.
Oxford University Press, USA, 1977.

Existem Muitos Tipos de Padrão!

- Padrões de gerenciamento
  - Ex. Métodos ágeis como SCRUM e XP
- Padrões organizacionais
- Padrões de análise
- Padrões arquiteturais => **Estilos Arquiteturais**
- **Padrões de implementação => *Idioms***

Propriedades dos Padrões de Projeto

- Padrões...
  - fornecem um vocabulário comum
  - fornecem uma abreviação para comunicar princípios complexos
  - auxiliam na documentação do projeto do sistema
  - capturam partes essenciais de um projeto de maneira compacta
  - mostram mais de uma solução
- Padrões não...
  - fornecem uma solução exata
  - resolvem todos os problemas de projeto
  - aplicam-se apenas ao projeto orientado a objetos
    - Em computação, surgiram nesse contexto, porém

Elementos de um Padrão de Projeto

1. Nome do padrão

- Apelido usado para descrever uma solução de projeto
- Facilita discussões de projeto
- Um padrão pode ter vários nomes distintos

Elementos de um Padrão de Projeto

2. Problema

- Descreve quando o padrão pode ser aplicado
- Pode incluir condições sobre a aplicabilidade do padrão (contexto)
- Sintomas de um projeto inflexível ou inadequado

Elementos de um Padrão de Projeto

3. Solução

- Descreve elementos do projeto
- Inclui relacionamentos, responsabilidades e colaborações
- Não descreve projetos concretos ou implementações
- Funciona como um modelo
  - Um **estilo arquitetural** nada mais é que um padrão aplicado no nível da arquitetura

Elementos de um Padrão de Projeto

4. Consequências

- Resultados e compromissos
- Normalmente classificadas em vantagens e desvantagens
- Fundamentais para se avaliar a aplicabilidade do padrão em determinado contexto
- Em geral, padrões de projeto implicam em maior flexibilidade
  - É necessário avaliar se essa flexibilidade é necessária

Elementos de um Padrão de Projeto

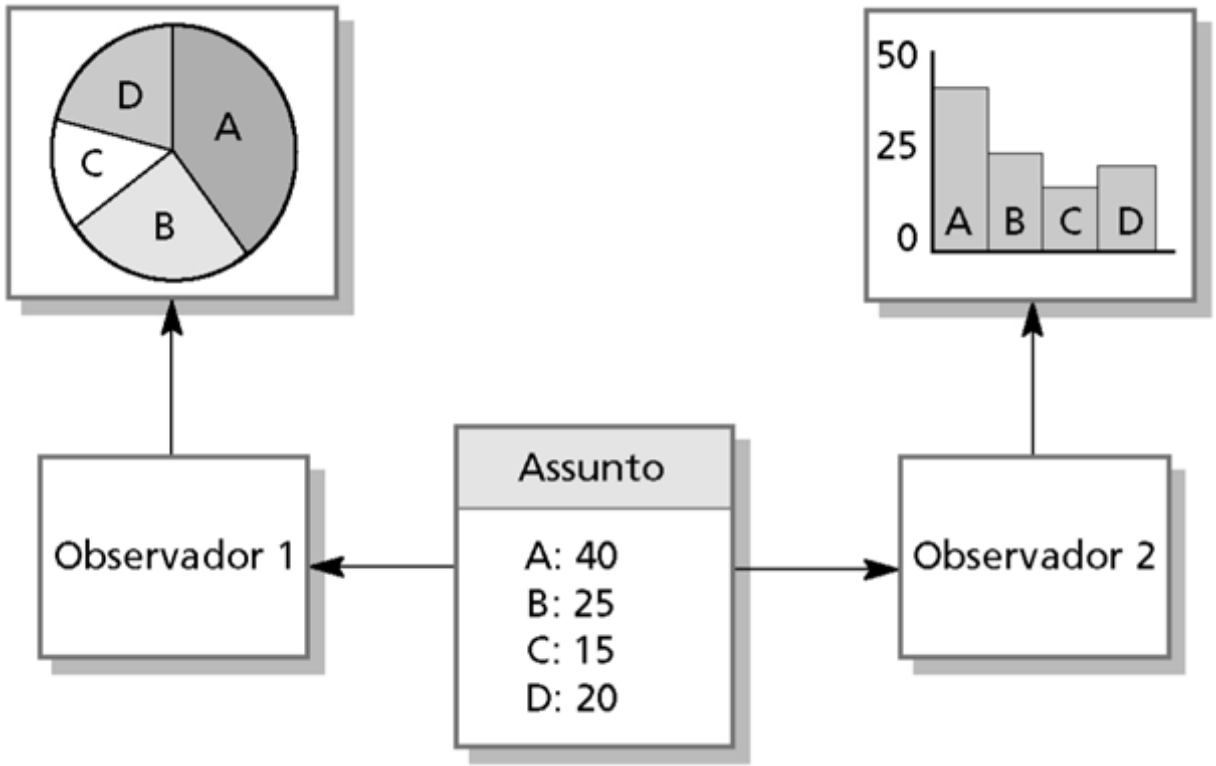
5. Usos conhecidos

- Padrões são soluções já testadas em vários contextos, potencialmente por várias pessoas
- Benefícios e desvantagens bem conhecidos!
- Regra dos três
  - Uma ocorrência é um evento isolado
  - Duas podem significar uma coincidência
  - Três fazem um padrão

Múltiplas formas de exibição

Figura 18.2

Vários displays.



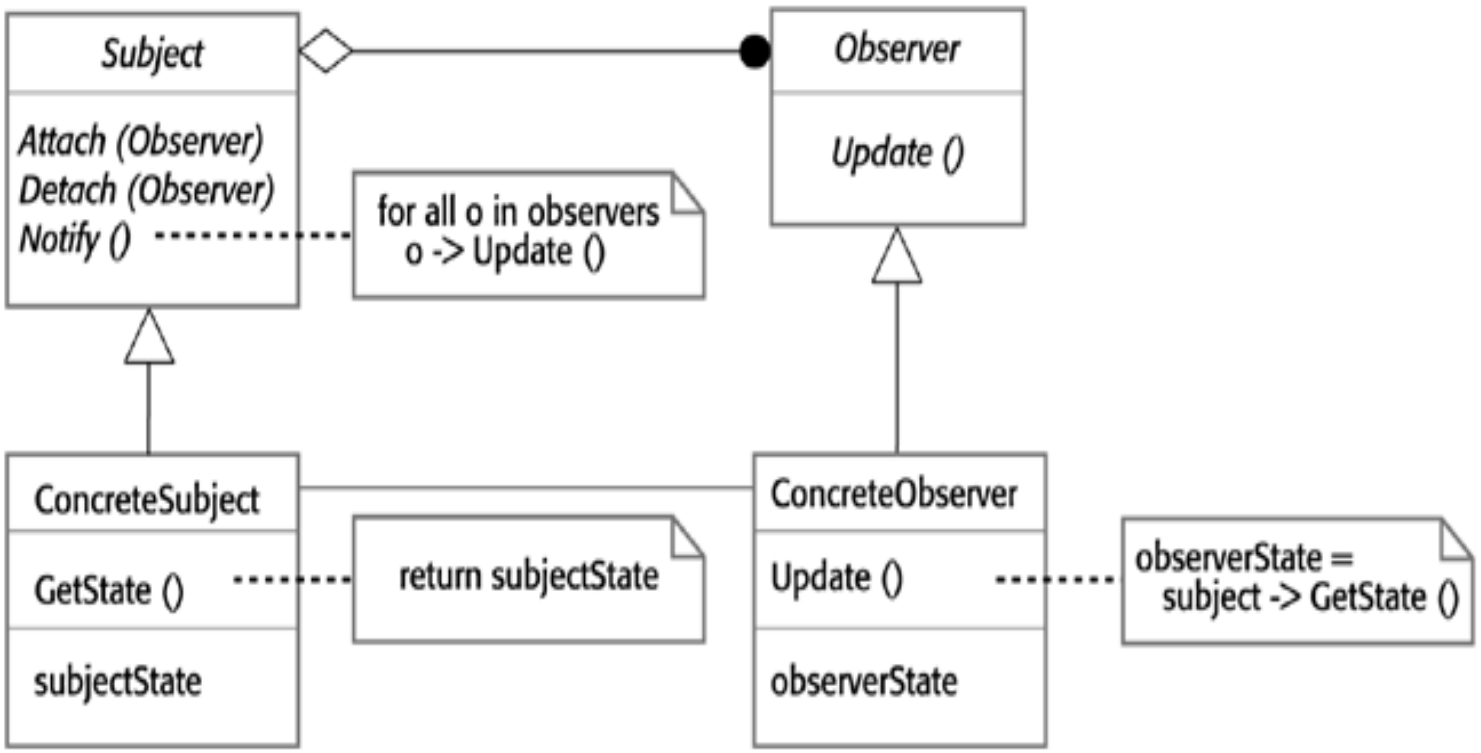
O padrão *Observer*

- **Nome: *Observer*.**
- **Descrição do problema: É necessário representar um mesmo conjunto de dados de diversas maneiras, de modo que mudanças nesses dados se reflitam em todos os modos de exibição**
- **Descrição da solução: Mecanismo para que o *display seja notificado de modificações no estado sem que, para isso, o estado precise conhecê-lo***
- **Conseqüências:**
  - Estado e *display desacoplados*
  - *É fácil atualizar tanto um display quanto vários displays*
  - Invocações implícitas podem resultar em erros de programação difíceis de rastrear
  - Exige código extra

O padrão *Observer*

Figura 18.3

Padrão Observer.



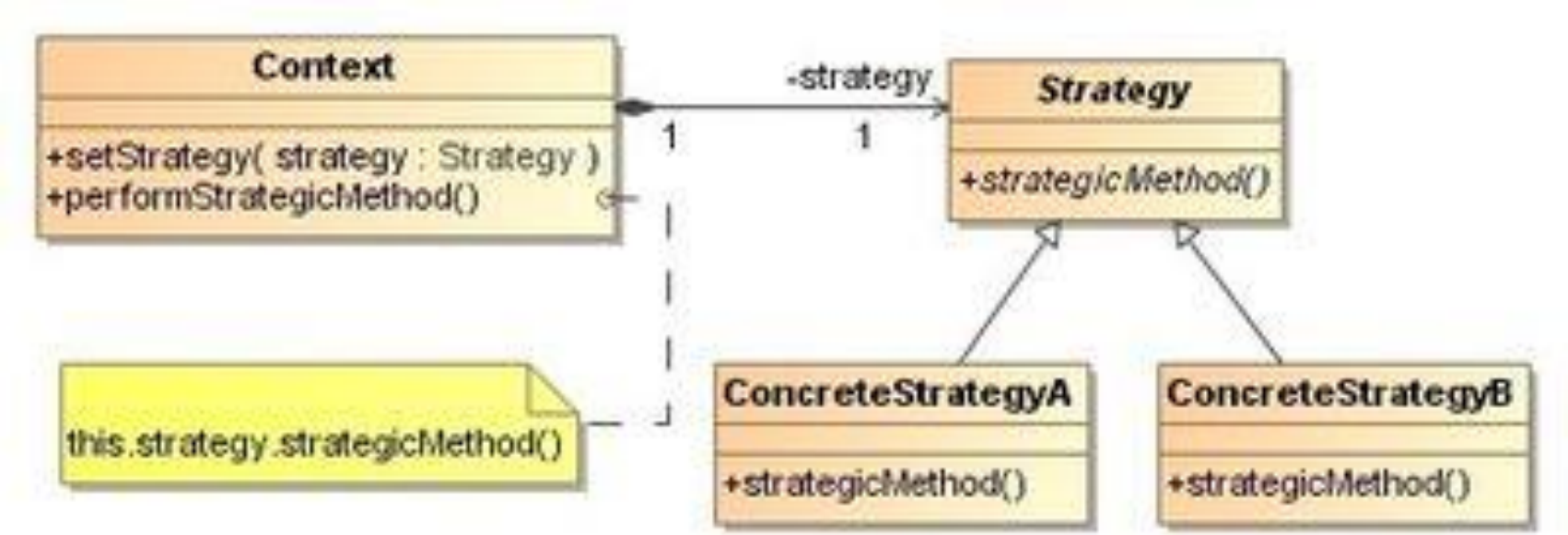
Usos conhecidos

- ***AWT do Java Development Kit***
- ***Diversas partes da implementação da plataforma Eclipse***
- ***Um grande número de arcabouços modernos para a construção de interfaces com o usuário***

O padrão *Strategy*

- **Nome**: *Strategy*
- **Descrição do problema**: Dependendo do contexto, é necessário usar algoritmos diferentes para resolver um problema, embora seja desejável não ter que modificar esse contexto por causa do algoritmo empregado
- **Descrição da solução**: Definir uma interface para todos os algoritmos e fazer com que o contexto conheça apenas essa interface, não suas implementações
- **Conseqüências**:
  - Separa contexto e cliente das implementações do algoritmo
  - Contexto não está ciente da estratégia usada; o cliente configura o contexto
  - *Strategies* podem ser substituídas em tempo de execução
  - Normalmente mais efetivo se usado junto com uma *Factory*

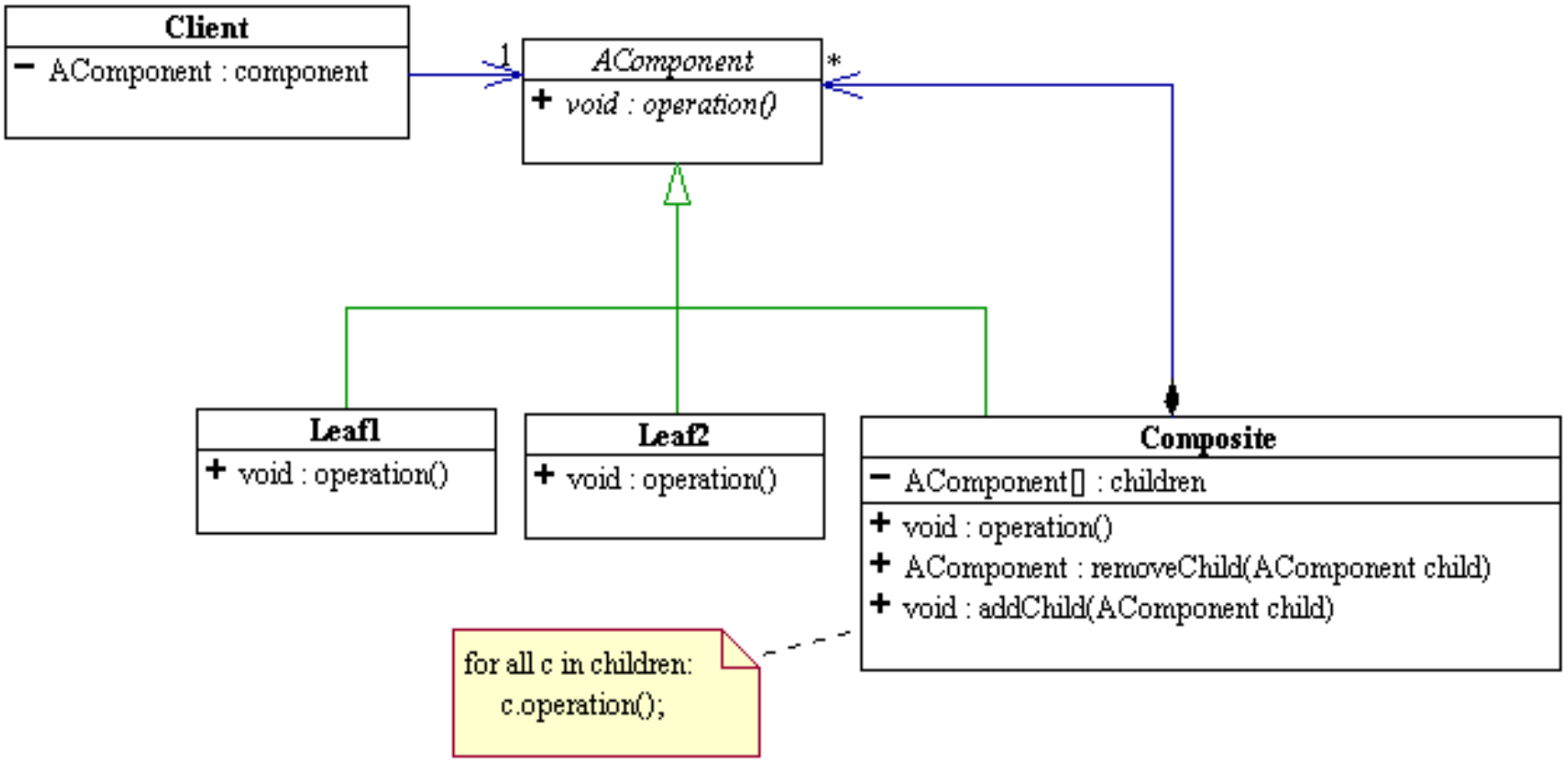
O padrão *Strategy*



O padrão *Composite*

- **Nome**: *Composite*
- **Descrição do problema**: Dados que o programa precisa manipular podem ser tanto elementos atômicos quanto grupos de elementos (podendo conter, inclusive, outros grupos). O programa deveria tratar esses itens de maneira uniforme
- **Descrição da solução**: Definir uma interface compartilhada tanto por itens atômicos quanto por grupos de elementos e fazer com que o programa lide com ela, sem saber se o objeto subjacente é atômico ou um grupo
- **Conseqüências**:
  - Trata todos os componentes do mesmo jeito, independentemente da complexidade
  - Novos tipos de componentes podem ser incluídos com facilidade
  - Pode exigir um número grande de objetos

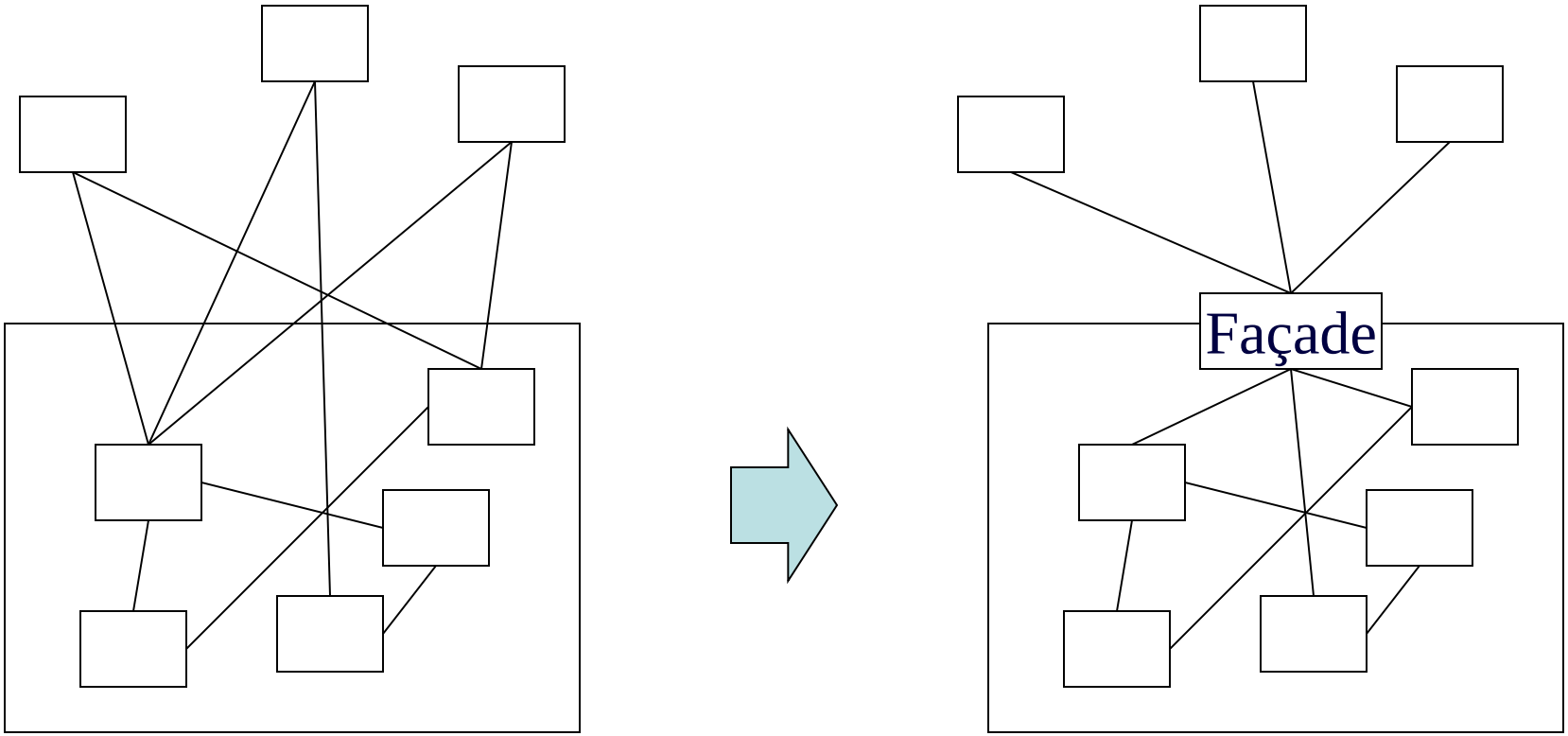
O padrão *Composite*



O padrão *Iterator*

- **Nome**: *Iterator*
- **Descrição do problema**: É necessário separar as operações que serão realizadas sobre uma ou mais estruturas de dados da maneira como essas estruturas de dados serão percorridas
- **Descrição da solução**: Definir uma interface padrão para percorrer estruturas de dados em geral e fazer com que seus usuários vejam apenas a interface e não a maneira como a estrutura de dados é percorrida
- **Conseqüências**:
  - Independência entre a estrutura de dados e a maneira de percorrê-la
  - Múltiplos iteradores => múltiplas maneiras de percorrer uma estrutura de dados
  - Comunicação extra entre o iterador e a estrutura
  - Modificações à estrutura podem criar inconsistências

O padrão *Façade*



Mais sobre padrões de projeto

Para um entendimento rapido, dêem uma olhada em:

(Importante para estudantes de graduação!)

http://en.wikipedia.org/wiki/Design_Patterns

O texto sobre padrões está legal e tem *links* para as descrições de todos os padrões do livro (com código!)

ou

http://www.oodesign.com/