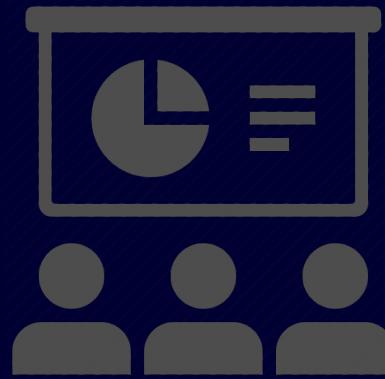


Design e Arquitetura de Software

Projeto de Software

AGENDA



1. Apresentação

2. Livros

3. Acordo de
Convivência

4. - História da
Arquitetura de Software

5. - Papel do Arquiteto

6. - Padrões
Arquiteturais

7. - Exercicio

Apresentação

FORMAÇÃO ACADÉMICA

- ◆ Graduado em Telemática/Telecomunicações - IFCE (2002 - 2008)
- ◆ Especialista em Engenharia de Software - FA7 (2011 - 2013)
- ◆ MSc em Engenharia de Software - UFPE (2011 - 2015)

CURRÍCULO PROFISSIONAL

- ◆ Atuei 4 anos na empresa privada
- ◆ 10 anos no ambiente Público
- ◆ Atualmente Líder Técnico de 45 Projetos de Tecnologia na SEPOG/PMF

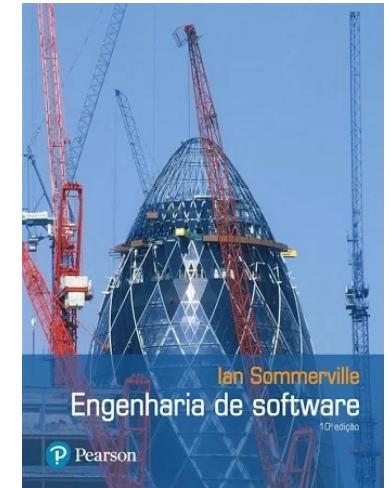
Apresentação

DOCÊNCIA

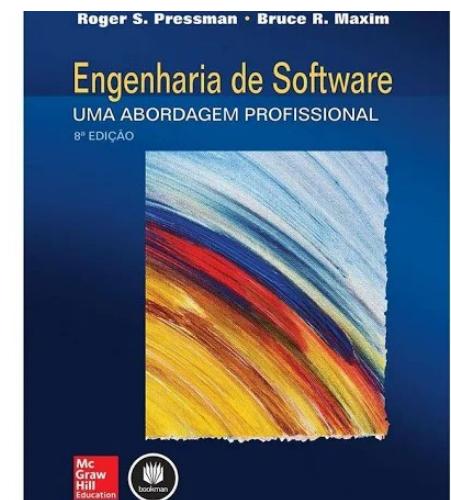
- ◆ Professor Substituto das Disciplinas de Sistemas de Informação – FA7
(2011 - 2012)
- ◆ Professor da Especialização em Sistemas WEB – FJN
(2011 - 2012)
- ◆ Professor de Bancas de graduação em Sistemas de Informações – FA7
(2012)
- ◆ Professor dos Cursos de Tecnologia da Unifanor (2015 – 2018)
- ◆ Professor do Curso de Tecnologia da Unichristus (2018 - Atual)

Livros

- **Engenharia de Software** - 10^a Ed – Ian Sommerville - Pearson



- **Engenharia de Software – Uma abordagem profissional** - 8^a Ed. AMGH



Dicas de Convivência

- ◆ Horários
- ◆ Conversas
- ◆ Dúvidas
- ◆ Celular
- ◆ Avaliações

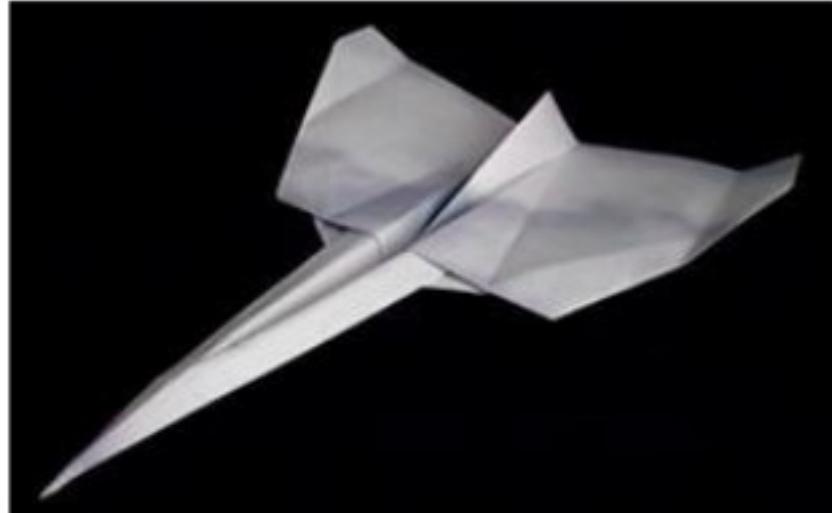




Questionamentos



Arquitetura de Software: Motivação



- Projetos simples podem ser realizados por uma única pessoa
 - Pouca modelagem
 - Ferramentas simples
 - Processo simples
 - Pouco projeto
 - Pouca especialização para construir

Arquitetura de Software: Motivação



- Projetos complexos/Maiores exigem arquitetura
 - Mais modelagem
 - Ferramentas mais poderosas
 - Processos mais bem definidos
 - Mais projeto
 - Alta especialização para construção

Arquitetura de Software

Conceito: O que é Arquitetura de SW?

- “É a organização fundamental de um sistema, expressa nos seus componentes, nos relacionamentos entre eles e com o ambiente, e nos princípios que governam seu projeto e sua evolução”.

Fonte: Norma 1471 do IEEE, publicada em <http://www.iso-architecture.org/ieee-1471/faq.html#wharch>

Arquitetura de Software

“PESSOAS DE TODO MUNDO ESTÃO
CONSTANTEMENTE CRIANDO APLICAÇÕES WEB
USANDO .NET, JAVA E PHP. NENHUMA DELAS
ESTÁ FALHANDO POR CAUSA DA TECNOLOGIA”

Joel Sposky

Arquitetura de Software

Conceituando Arquitetura de Software

O termo Arquitetura de Software é bastante antigo e foi sugerido pela primeira vez ao final da década de 60 por Edsger Dijkstra, porém, ganhou força e expressão na década de 90. Existem algumas definições de Arquitetura de Software com ideias em comum, mas que de modo geral se completam.

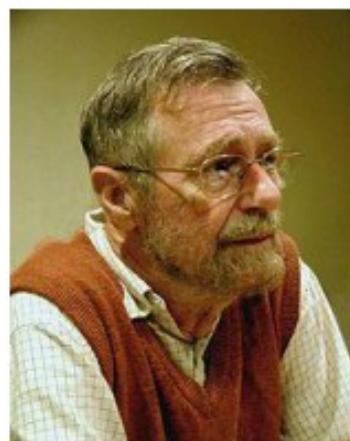


Figura 1 - Edsger Dijkstra (1930-2002)

Fonte: http://pt.wikipedia.org/wiki/Edsger_Dijkstra

Definição pela IBM

Uma arquitetura é o conjunto de decisões significativas sobre a organização de um sistema de software, a seleção de elementos estruturais e suas interfaces, juntamente com o comportamento especificado nas colaborações entre estes elementos, a composição destes elementos em subsistemas progressivamente maiores e o estilo arquitetural que guia esta organização. (*The Rational Unified Process: An Introduction*)

Design e Arquitetura de Software

Projetar Software é o processo de aplicar várias técnicas e princípios com o propósito de se definir um dispositivo, processo ou sistema, com detalhes suficientes para permitir sua realização física. (Taylor-59).

O Projeto de software é o núcleo técnico da Engenharia de Software. É a única maneira de se traduzir "com precisão", os requisitos do usuário para um produto ou sistema acabado.

Meta:

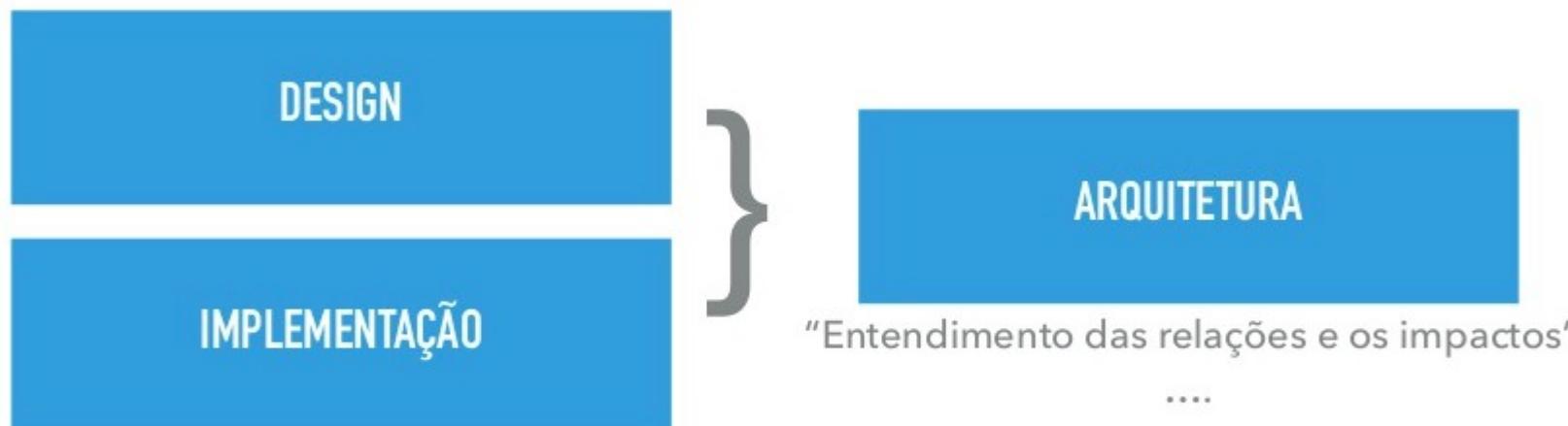
Traduzir requisitos numa representação de software.

Papel do Arquiteto de Software

- ▶ Identificar as interfaces entre os componentes.
- ▶ Direcionando a equipe de desenvolvimento
- ▶ “Guia ... experiente e capacitado que ensina aos outros se virar melhor - Martin Fowler”
- ▶ Ter a visão do Design e da implementação

Papel do Arquiteto de Software

Precisa tomar decisões



Habilidades do Arquiteto de Software



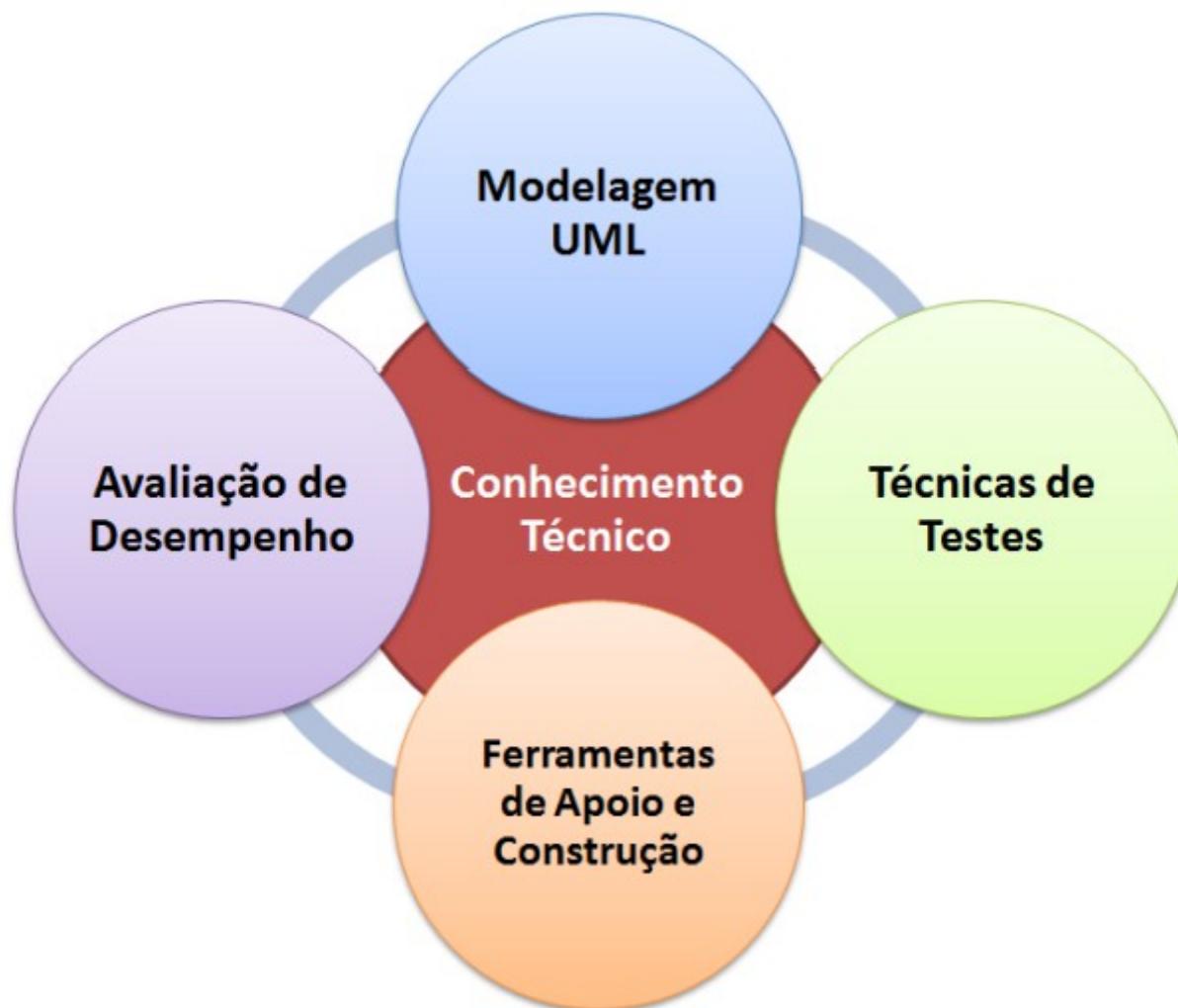
Influências sobre a Arquitetura de Software

Habilidades do Arquiteto de Software



Habilidades interpessoais

Habilidades do Arquiteto de Software



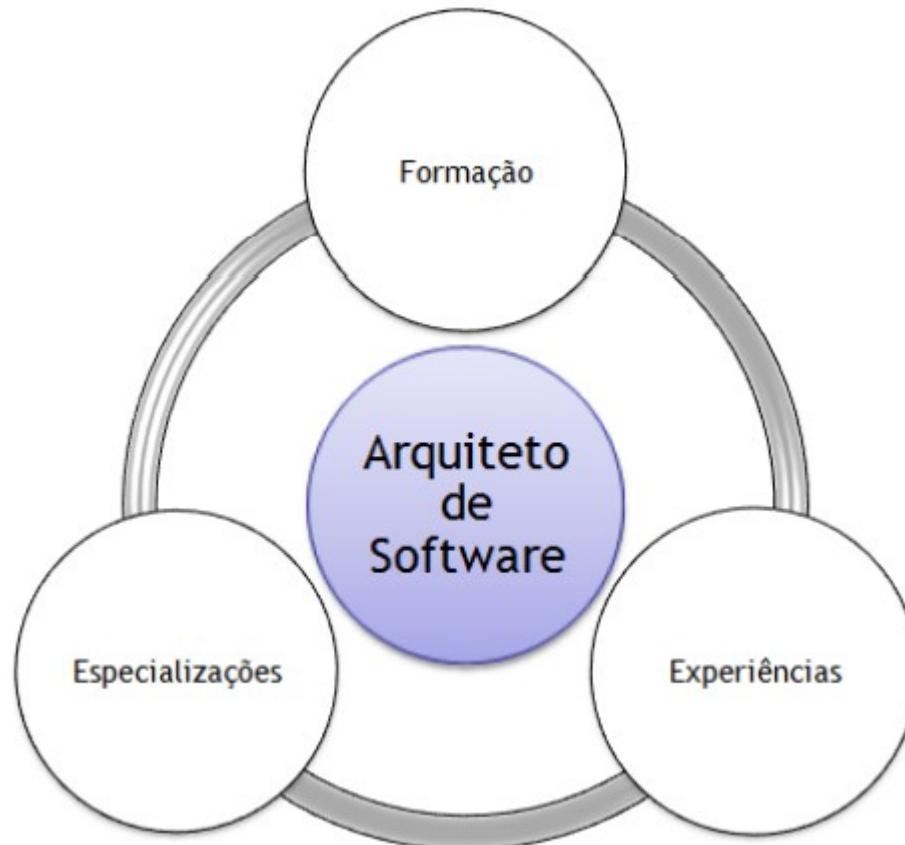
Conhecimento Técnico

Habilidades do Arquiteto de Software



Conhecimento de Negócio

Habilidades do Arquiteto de Software



Trajetória para Formação do Arquiteto de Software

Habilidades do Arquiteto de Software

Especializações

Tecnologias

- Linguagens
- Banco de dados
- Frameworks Front-end
- Frameworks Back-end

Ferramentas

- IDE's de desenvolvimento
- Gerenciadores de Versão
- Ferramentas de Modelagem
- Ferramentas de Apoio

Conhecimentos

- Arquitetura de Sistemas
- Orientação a Objetos
- Padrões de Projeto
- Integração Contínua
- Análise e otimização
- Gestão de Pessoa

Certificações

- The Open Group IT Architect Certification (ITAC)
- The Open Group TOGAF Certification
- IASA Certified IT Architect (CITA) Program
- The Enterprise Architecture Center of Excellence (EACOE)
- Sun Certified Enterprise Architect (SCEA)

Trajetória para Formação do Arquiteto de Software

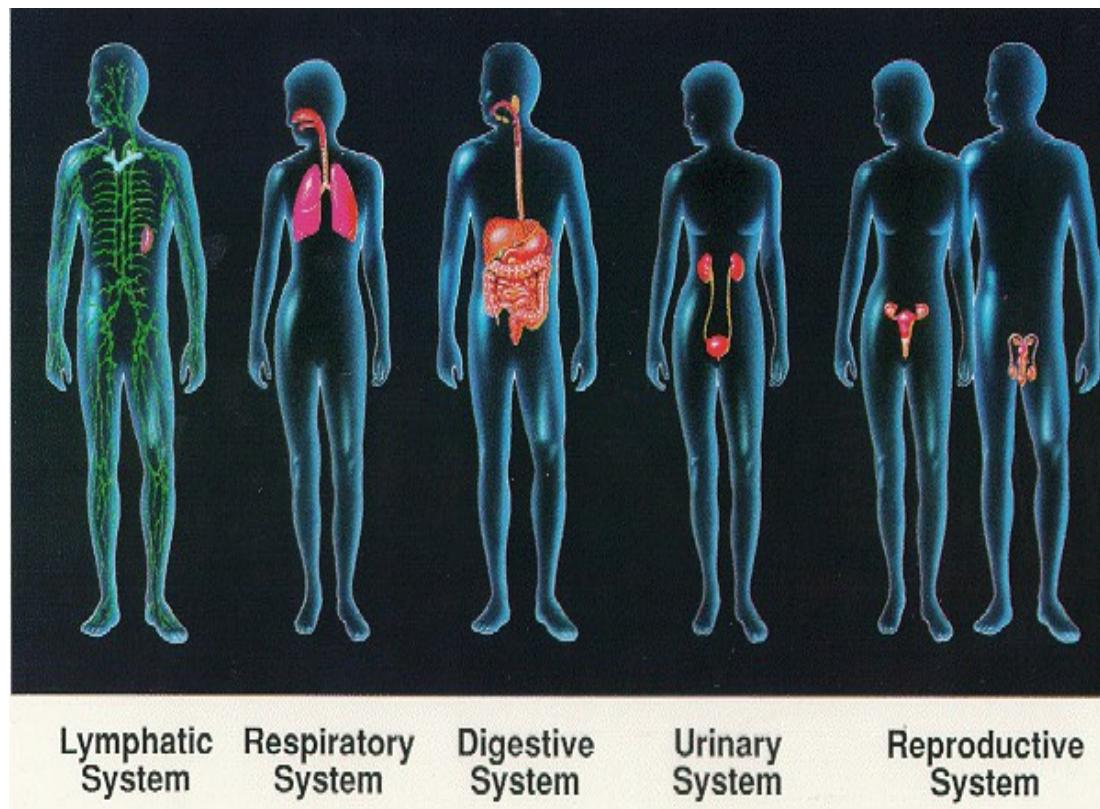
Conceito: O que é Arquitetura de SW? Componentes



- Pode-se pensar em um sistema de software constituído de diversos componentes.
- Estes elementos, podem ser substituídos por outros componentes equivalentes (mesma interface) ou mais sofisticados

Design e Arquitetura

- **Estrutura** é o conjunto de elementos e sua organização (real)
- **Visão** é a representação de tais elementos



Diferentes visões

- ✓ Projeto Procedimental: descrição da funcionalidade do software (algoritmos).
- ✓ Projeto de Dados: definição das estruturas de dados.
- ✓ Projeto das Interfaces: Layouts e mecanismos de interação homem-máquina (se necessário).
- ✓ Projeto Arquitetural: associação entre os principais elementos estruturais do software (árvore dos módulos, mensagens entre objetos, Nivelamento em Camadas).

Arquitetura nasce dos requisitos

Functionality (Funcionalidade):

- ✓ Requisitos mais tradicionais
- ✓ Associados as características principais do sistema

| Requisito | Descrição |
|-----------|--|
| Vendas | O sistema deve possuir um cadastro de orçamentos com opção para efetivação de venda |
| Impressão | O sistema deve oferecer opção para impressão de relatório de todas as opções de cadastros disponíveis |
| Segurança | O sistema deve permitir acesso somente após autenticação de usuários com login e senha |
| Auditoria | Todas as operações de leitura, inclusão, alteração e exclusão devem ser registradas para fins de auditoria |

Arquitetura nasce dos requisitos

Usability (Usabilidade)

- ✓ Aspectos de Interação entre Usuário e Sistema
- ✓ Cuidado com requisitos subjetivos. Tente obtê-los com descrições objetivas.

| Requisito | Descrição |
|-----------|--|
| Interface | O sistema deve possuir interface moderna (subjetivo) |
| Interface | O sistema deve fornecer opções de atalho para proporcionar agilidade e clareza nos trabalhos de rotina do pessoal operacional (objetivo) |

Arquitetura nasce dos requisitos

Reliability (Confiabilidade)

- ✓ Disponibilidade do sistema
- ✓ Precisão de resultados

| Requisito | Descrição |
|-----------------|---|
| Disponibilidade | O sistema deverá funcionar em 365x24x7 com no mínimo 99% de disponibilidade |
| Precisão | O sistema deve oferecer análise de crédito com no mínimo 85% de confiabilidade |
| Disponibilidade | A operação de simulação de empréstimos deve estar disponível em 99% das tentativas dos usuários |

Arquitetura nasce dos requisitos

Performance (Desempenho)

- ✓ Tempo de Resposta
- ✓ Throughput
- ✓ Carga

| Requisito | Descrição |
|---------------------------------|---|
| Carga e Tempo de Resposta | O sistema deverá suportar no mínimo 20000 usuários conectados simultaneamente no horário de pico e com tempo de resposta da operação de compra em no máximo 500 milissegundos |
| Throughput | O sistema deve ser capaz de processar no mínimo 2000 transações por segundo |

Arquitetura nasce dos requisitos

Supportability (Suportabilidade)

- ✓ Associados a requisitos não-funcionais:
 - Testabilidade, Adaptabilidade, Configurabilidade, Manutenibilidade

| Requisito | Descrição |
|-------------------|---|
| Testabilidade | As telas do sistema devem suportar automação de testes de interface |
| Adaptabilidade | O sistema deve ser capaz de adaptar sua interface de acordo com as diversas resoluções de vídeo |
| Configurabilidade | Todas as taxas de impostos aplicadas sobre transações do sistema devem permitir alteração e aplicação direta sem necessidade de reiniciar o sistema |
| Manutenibilidade | O processo de restauração de <i>backup</i> deve ser concluído em no máximo 40 minutos |

Arquitetura nasce dos requisitos

(+) Restrições adicionais importantes: Representam algumas restrições sobre o projeto e podem ser classificadas conforme exemplos abaixo:

✓ **Restrições ao desenho**

- “O sistema deve utilizar banco de dados relacional”

✓ **Restrições à implementação**

- “O banco de dados deve ser SQL-ERVER 2008 R2”

✓ **Restrições de interface**

- “Garantir interoperabilidade com outros sistemas por meio de Web Services”

✓ **Restrições físicas**

- “O dispositivo que irá suportar o sistema deve ter tamanho máximo de 100x200 cm e com peso máximo de 600 gramas”

Estilos e Padrões arquiteturais

- Estilos arquiteturais
 - Definem meios de selecionar e apresentar blocos de construção de arquitetura
- Padrões arquiteturais
 - Projetos de alto nível, testados e validados, de blocos de construção de arquitetura

Shaw, M., Garlan, D. Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1966

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad e M. Stahl. Pattern-Oriented Software Architecture - A System of Patterns, NY: John Wiley and Sons, Inc. 1996

Categorias de Estilos de Arquiteturas

- **Estrutura** (“**From mud to structure**”)- oferecem decomposição controlada das tarefas em sub-tarefas. Consideram requisitos estáveis e bem definidos.
- **Sistemas distribuídos** – aplicações distribuídas
- **Sistemas interativos** – interação HM.
- **Sistemas adaptáveis** – oferecem suporte para extensão e adaptação de aplicações devido a tecnologias e mudança de requisitos.

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad e M. Stahl. *Pattern-Oriented Software Architecture - A System of Patterns*, NY: John Wiley and Sons, Inc. 1996

D. Garlan and Mary Shaw. An introduction to software architecture. Technical Report- CMU-CS-94166, Carnegie Mellon University, January 1994.

Estilos Arquiteturais: Taxonomia

| <i>Sistemas de Fluxo de Dados</i> | <i>Sistemas de Chamada e Retorno</i> | <i>Componentes Independentes</i> | <i>Máquinas Virtuais</i> | <i>Sistemas Centrados em Dados</i> |
|-----------------------------------|---|--|------------------------------------|------------------------------------|
| Seqüenciais Batch | Programa Principal e Sub-rotinas | Processos Comunicantes | Interpretador | Bancos de Dados |
| Pipes & Filters | Sistemas OO | Invocação Implícita (ou Sistemas Baseados em Eventos) | Sistemas Baseados em Regras | Sistemas de Hipertexto |
| | Camadas | | | Blackboards (Quadro-Negro) |

Extraido de (SHAW e GARLAN, 1996)

Estilos arquiteturais

Fluxo de dados (Data Flow)

- 1. Sequenciais Batch**

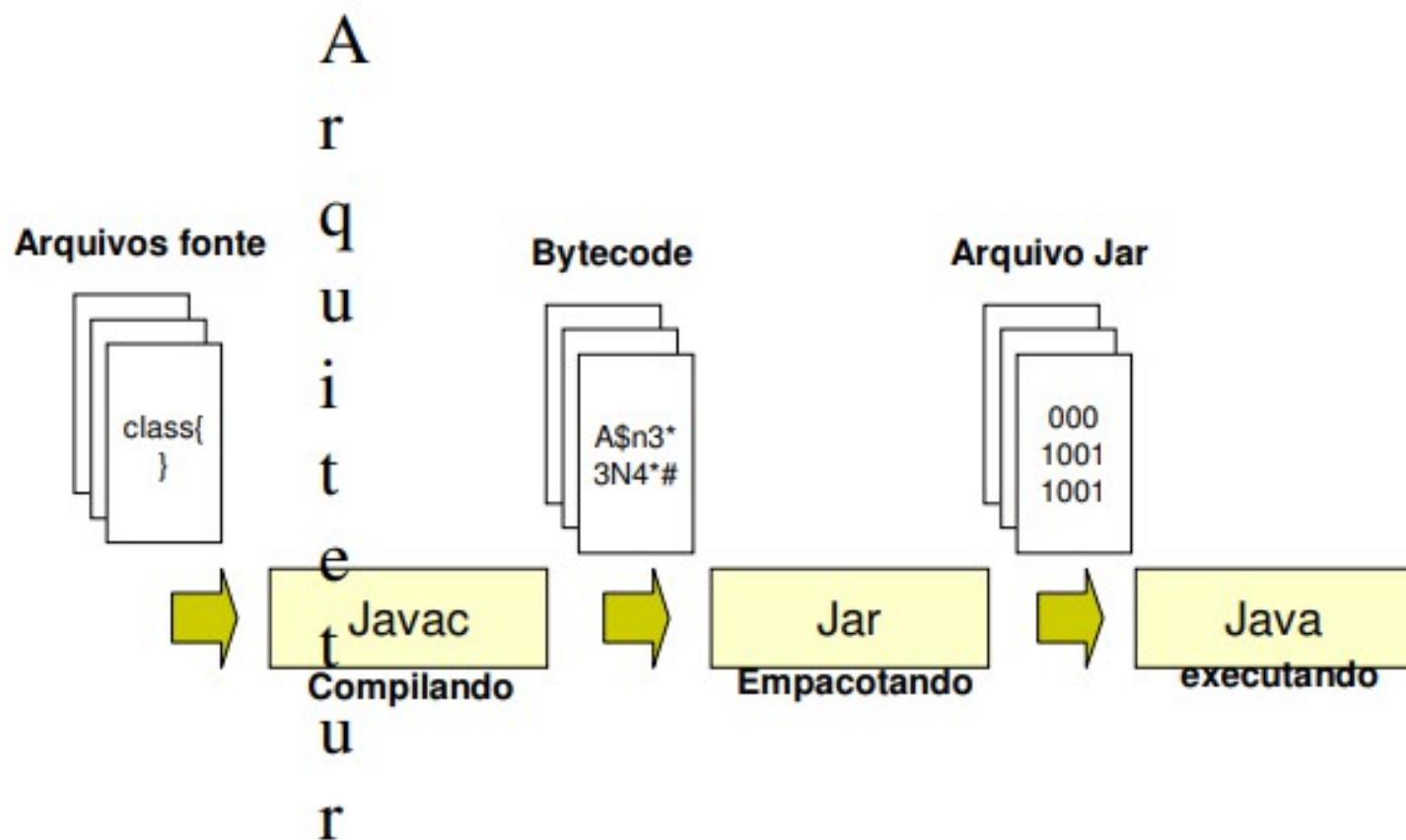
- 2. Pipes & Filters (Dutos e Filtros)**

1. Seqüências (*Batch Sequential*)

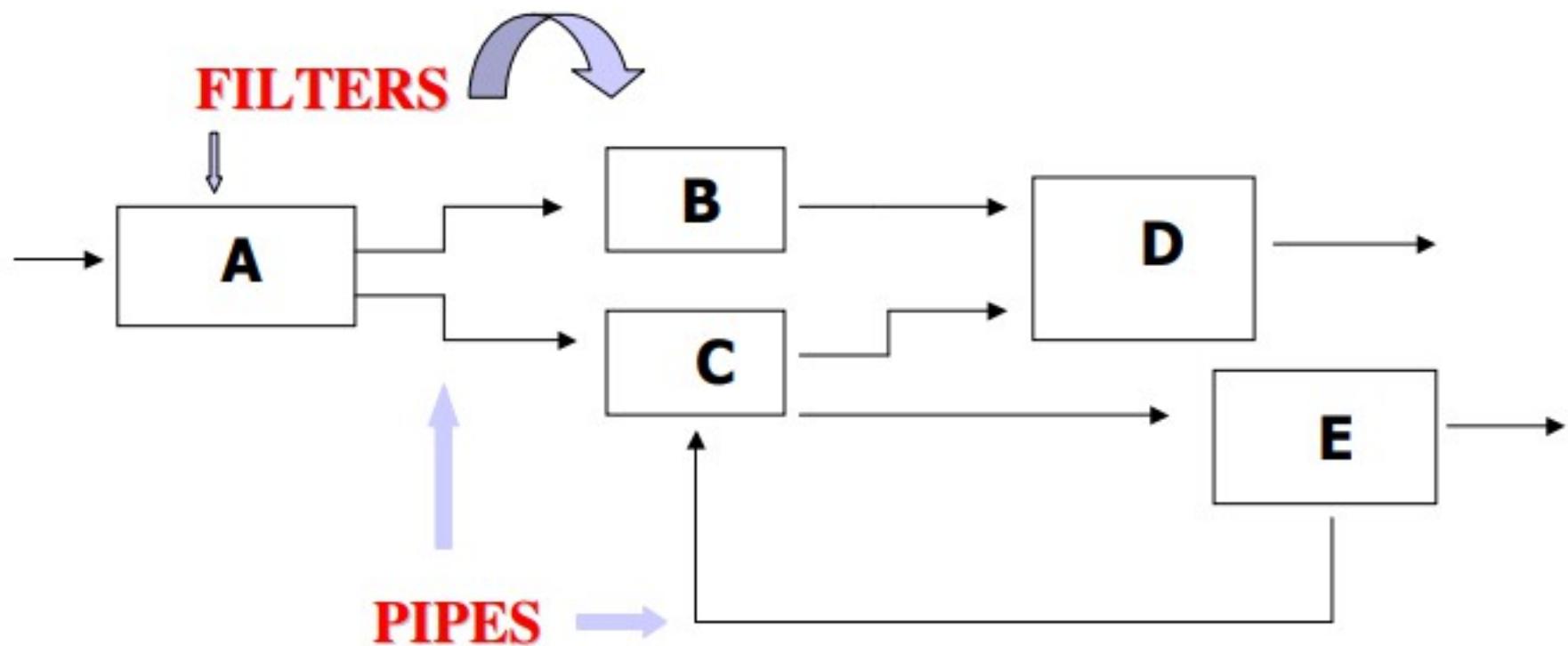
- Especialização de pipes & filtros
- Programas independentes executados em seqüência (pipelines: seqüências linear de filtros)
 - Um após o outro
 - Dado transmitido por completo entre um programa e outro

Ex: Arquitetura de Compiladores

-E



2. Pipes & Filters (Dutos e Filtros)



- Tipo mais geral – não precisa ser sequencial
- Pipes tipados – os tipos passados entre dois filtros tem um tipo bem definido.

Características: Dutos e Filtros

- Vantagens
 - Útil para aplicações de processamento de informação que interagem pouco com usuários
 - Rápida prototipação
 - Apóia reúso de transformações (filtros)
 - É fácil adicionar, recombinar, ou trocar, novas transformações (flexibilidade)
 - É relativamente simples implementar como sistema concorrente (vários filtros em paralelo) ou seqüencial
 - Eficiência em processamento

Características: Dutos e Filtros

- Desvantagens
 - Requer um formato comum para a transferência de dados ao longo do *pipeline*
 - Não é apropriado para aplicações interativas
 - Não existe compartilhamento de dados
 - Ausência de gerenciamento de erros.

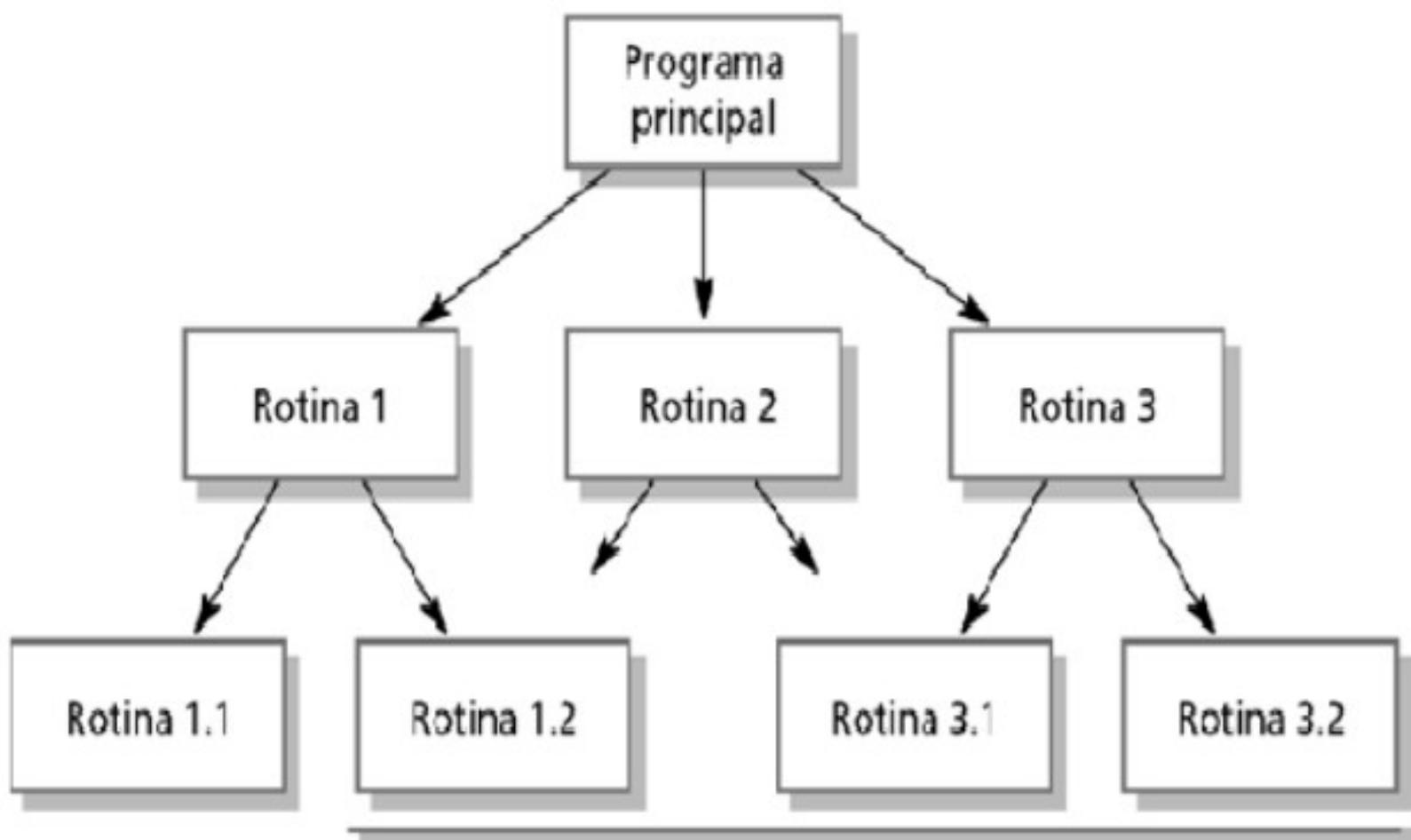
Estilos arquiteturais Chamada (ou invocações) e Retorno

1. Programa Principal e Sub-Rotina
2. Invocação Remota de Procedimento (RPC)
3. Sistema Orientados a Objetos
4. Camadas (Layered)

1. Programa principal e Sub-Rotinas *(Main Program/Subroutines)*

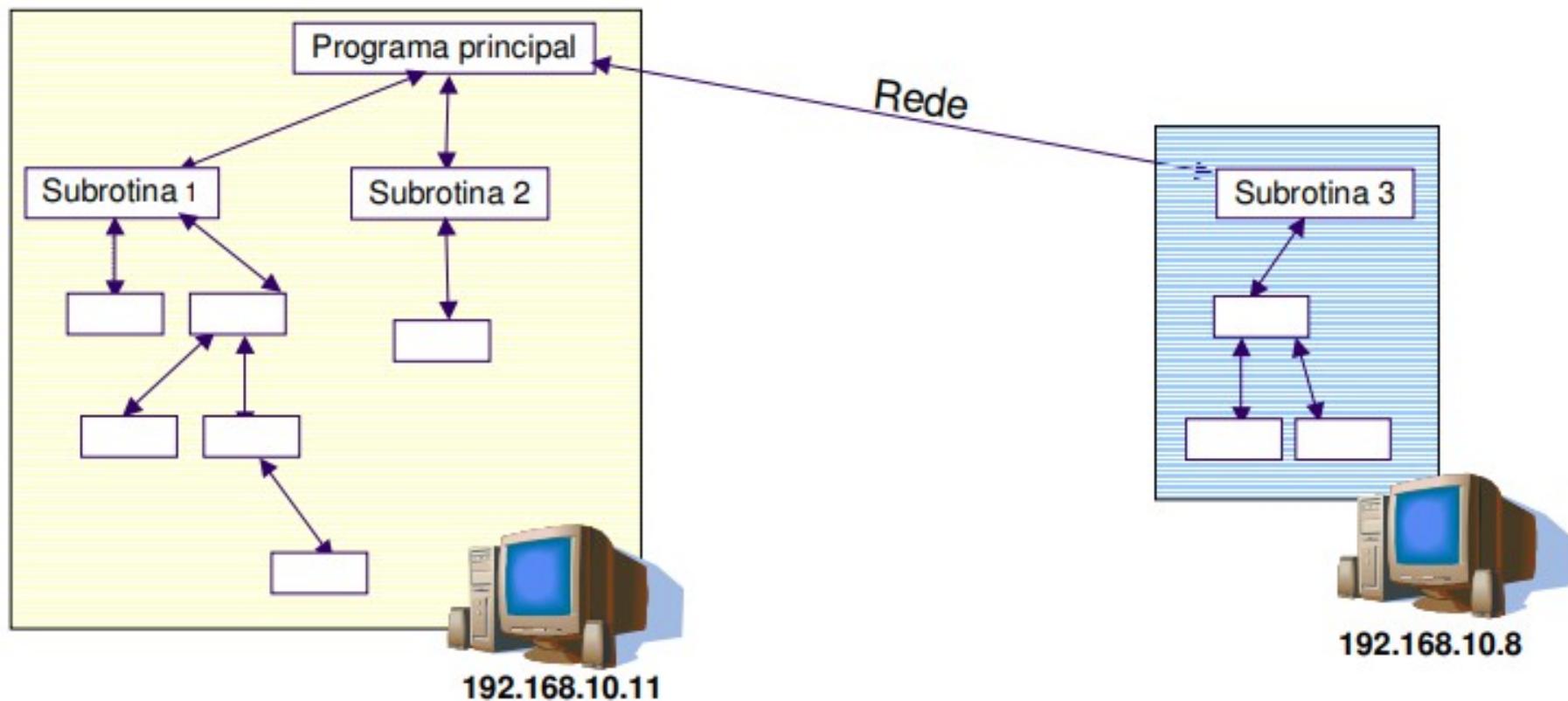
- Controle se inicia no topo de uma hierarquia de subrotinas e move-se para baixo na hierarquia.
 - Componentes – subrotinas
 - Conectores – chamadas de procedimento
- Vantagens: desenvolvimento pode ser independente; Gerenciamento de controle mais fácil de visualizar, na hierarquia de módulos.
- Desvantagens: o reúso, bem como alterações podem ser difíceis.

Exemplo:

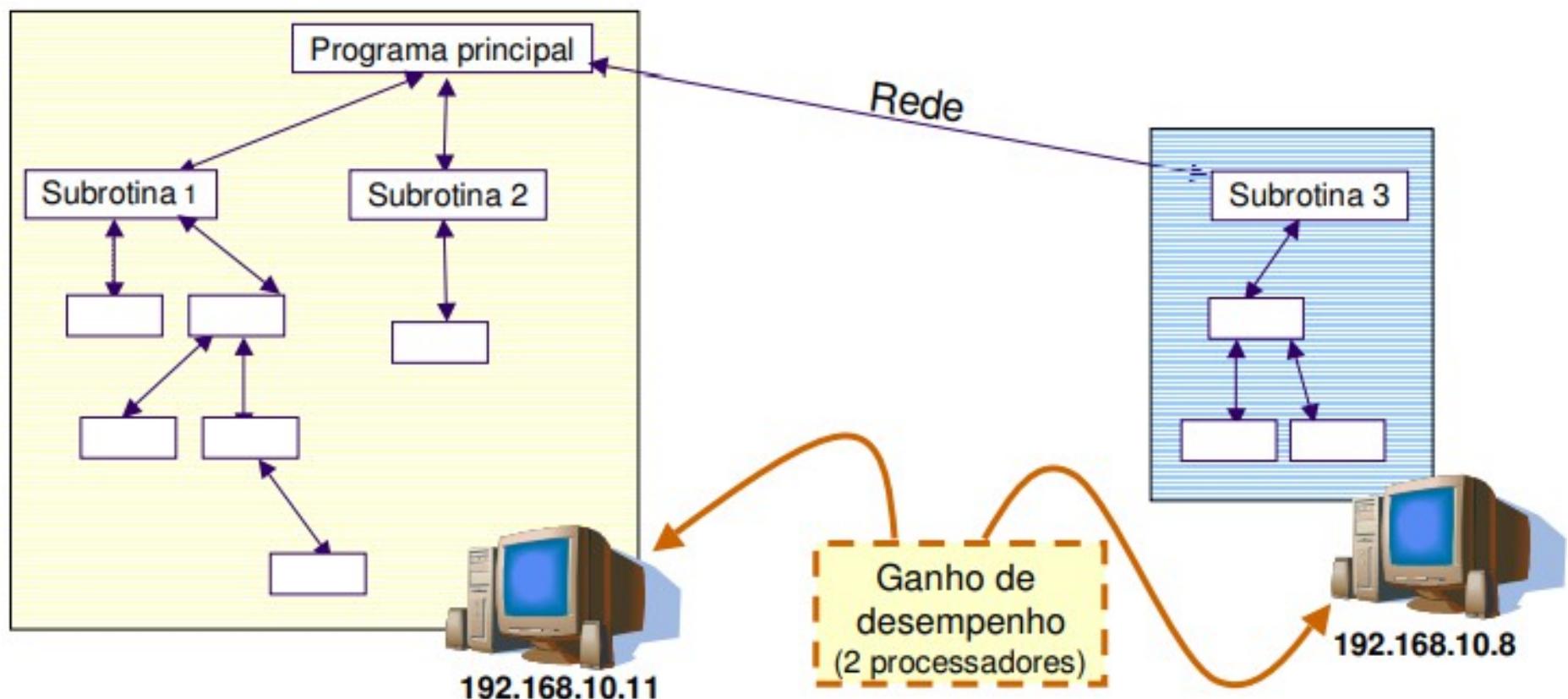


2. Invocação remota de procedimento (*RPC – remote procedure call*)

- especialização do programa principal e sub-rotinas



Vantagens



3. Sistema Orientados a Objetos

- Sistema como um conjunto de objetos fracamente acoplados e com interfaces bem definidas
 - Cada objeto oferece um conjunto de serviços
- Sistemas Orientados a Objetos (OO) podem ser vistos como uma rede ou grafo de objetos comunicantes.
 - Componentes: Objetos.
 - Conectores: envio de mensagem
 - Regras: objetos encapsulam seus dados; um objeto é responsável pela manutenção da integridade de sua representação.

Características: Sistemas Orientados a 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

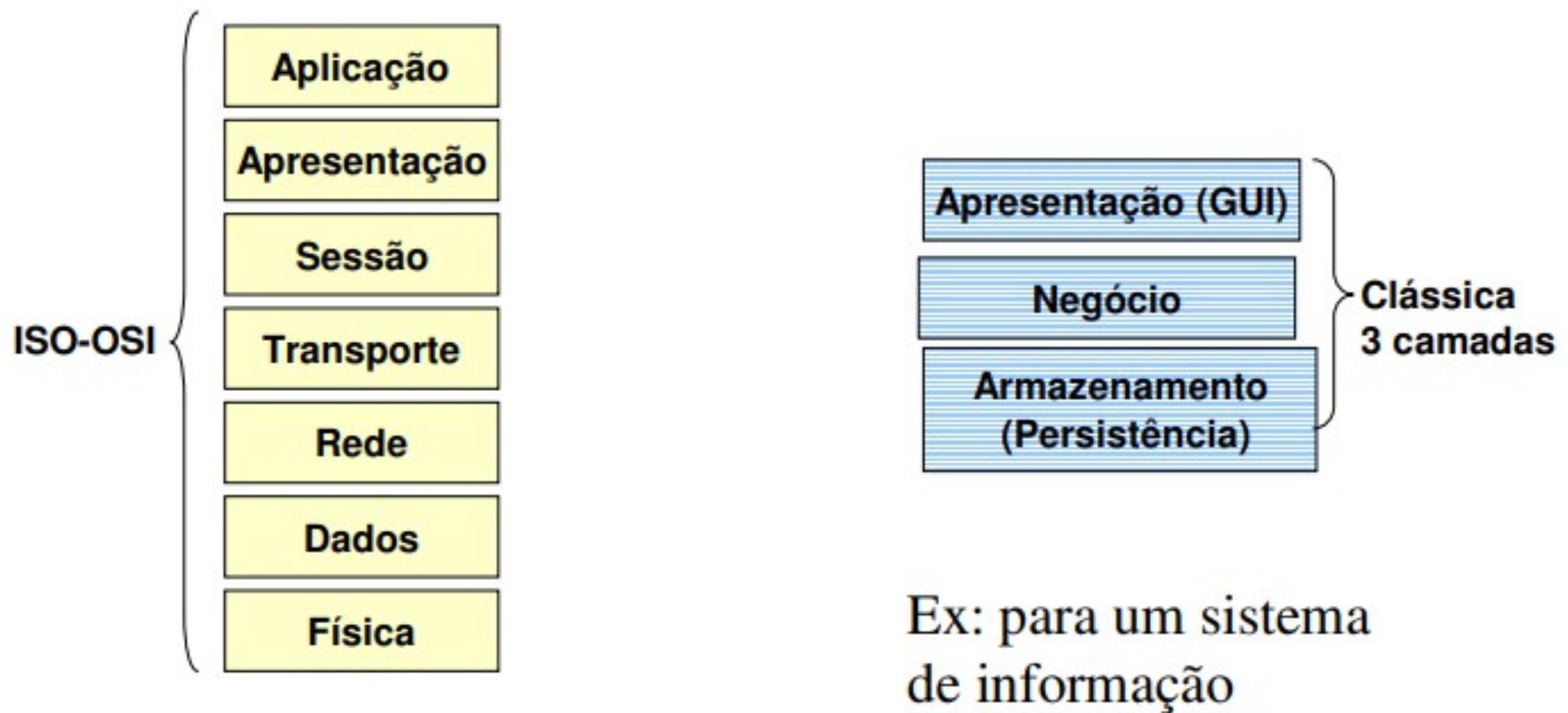
4. Camadas (Layered)

- Sistema organizado hierarquicamente.
- Cada camada oferece o serviço para a camada superior (externa) e utiliza um serviço da camada inferior (interna).
 - Componentes: são camadas, grupo de tarefas em um mesmo nível de abstração
 - Conectores: protocolos que indicam como as camadas irão interagir e limitam as comunicações a camadas adjacentes; permitem comunicação em máquinas diferentes
- Cada camada está associada a um conjunto de entidades (entidade complexa), constituindo-se de diferentes componentes (por ex.: conjunto de objetos, funções, etc).

4. Camadas (Layered)

- Sistema organizado hierarquicamente.
- Cada camada oferece o serviço para a camada superior (externa) e utiliza um serviço da camada inferior (interna).
 - Componentes: são camadas, grupo de tarefas em um mesmo nível de abstração
 - Conectores: protocolos que indicam como as camadas irão interagir e limitam as comunicações a camadas adjacentes; permitem comunicação em máquinas diferentes
- Cada camada está associada a um conjunto de entidades (entidade complexa), constituindo-se de diferentes componentes (por ex.: conjunto de objetos, funções, etc).

Exemplos



Características

- Camadas se comunicam apenas com outras adjacentes



Características: Estilo em Camadas

Vantagens:

- Facilidade de compreensão: utiliza níveis crescentes de abstração, particionando problemas complexos em sequência de passos incrementais.
- Suporte a padronização
- Desenvolvimento independente
- Facilidade de manutenção, reúso das camadas e suporte à evolução dos sistemas, oferecendo flexibilidade e boa manutenibilidade
- As dependências tendem a ser locais (dentro da camada)- restrição de comunicação entre camadas adjacentes, mudanças afetam no máximo duas camadas
- Se interface bem definida, permite uso de diferentes implementações da mesma camada

Características: Estilo em Camadas

Desvantagens

- Às vezes é difícil estruturar um sistema através de camadas. Conseqüências:
 - É comum que a estruturação seja violada
 - Camadas **relaxadas** são necessárias – todas as camadas podem se comunicar entre si.
 - Mudanças em serviços de uma cada inferior, podem requerer propagação de mudanças até as superiores
 - Pode haver necessidade de duplicação de funcionalidade
- *Overhead* (problemas) de implementação, comunicação, e desempenho
- Complexidade na Implementação e Testes do Sistema

Estilos arquiteturais Componentes Independentes

1. Processos comunicantes
Cliente-servidor, Peer to peer

2. Baseado em eventos
Invocação implícita

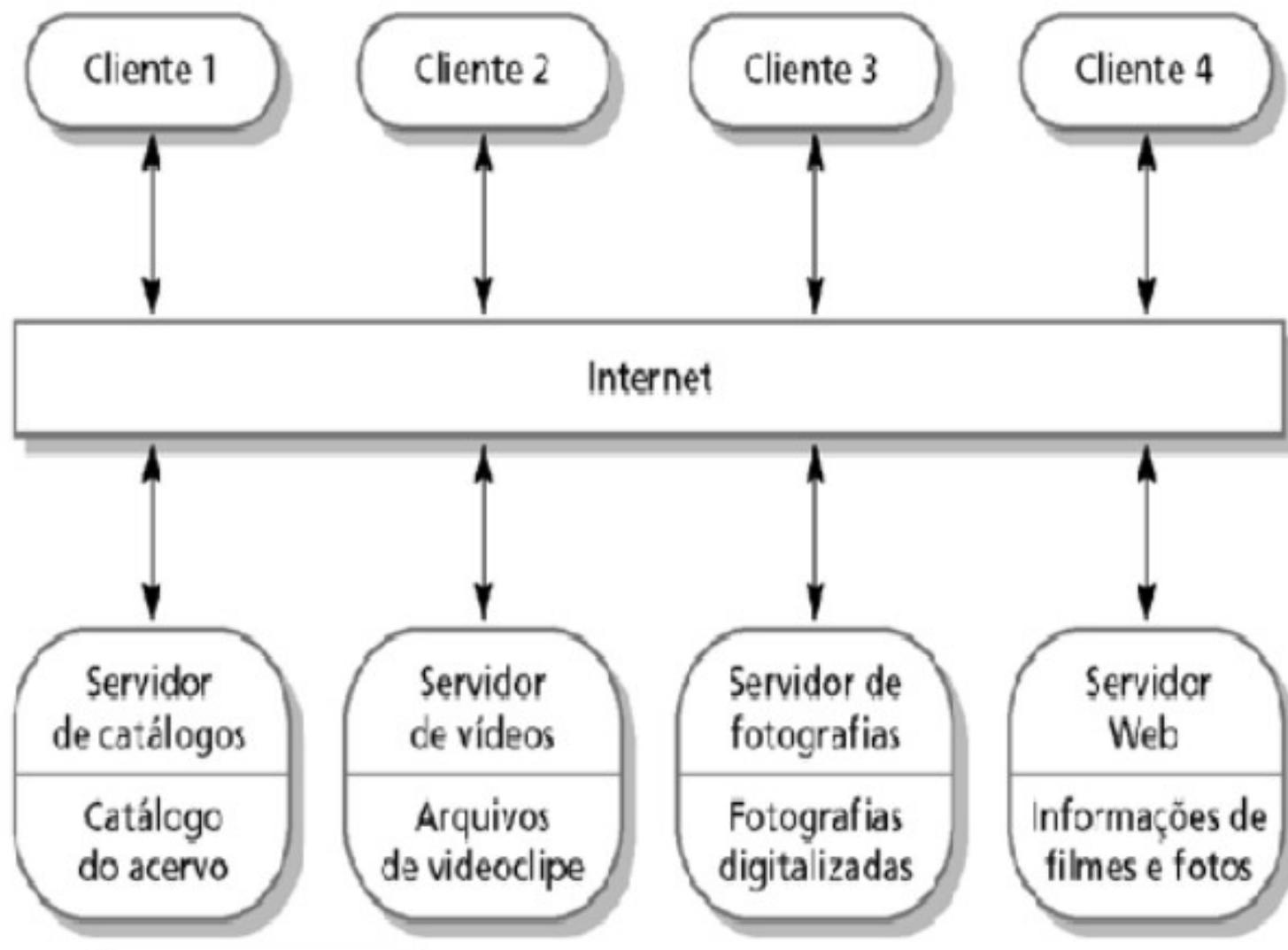
1. Processos comunicantes

- Baseado na comunicação via troca de mensagens entre processos
 - Em geral, via rede
- Cliente - Servidor
- Ponto a ponto (*Peer to Peer – P2P*)

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

Ex: Biblioteca de filmes e fotografias



Características: Cliente-Servidor

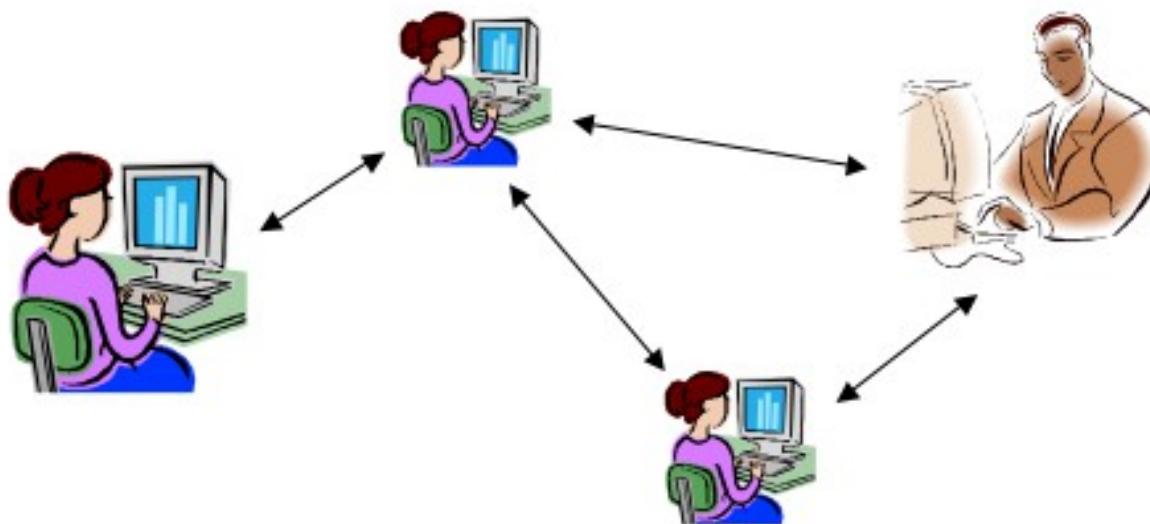
- **Vantagens**
 - Separação de interesses
 - Inerentemente distribuído: pode haver balanceamento de carga, tolerância a falhas
 - É fácil adicionar novos servidores ou atualizar servidores existentes.
 - Utilização dos recursos do servidor
 - Escalabilidade: aumentando a capacidade computacional do servidor

Características: Cliente-Servidor

- 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
 - Introduz complexidade
 - Custos de comunicação
 - Falhas no servidor

Ponto a Ponto (P2P)

- Não há distinção entre nós
- Cada nó mantém seus próprios dados e endereços conhecidos
- Cada nó é “cliente e servidor ao mesmo tempo”
- Vantagem: reduz problemas de falhas
- Desvantagem: aumenta o tempo de consulta

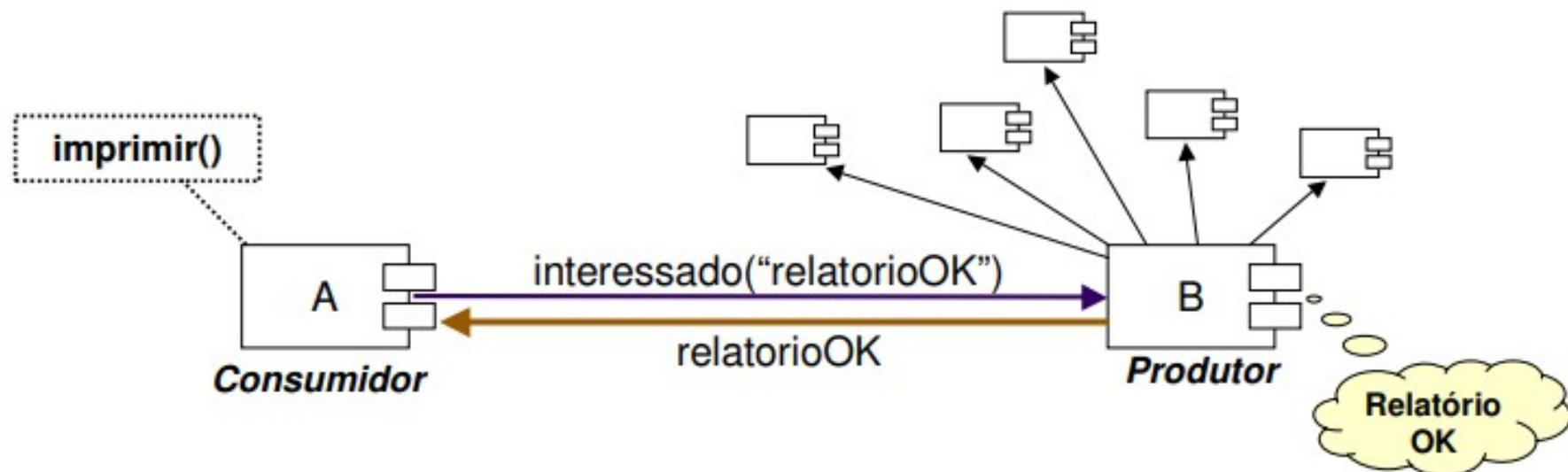


2. Baseado em eventos

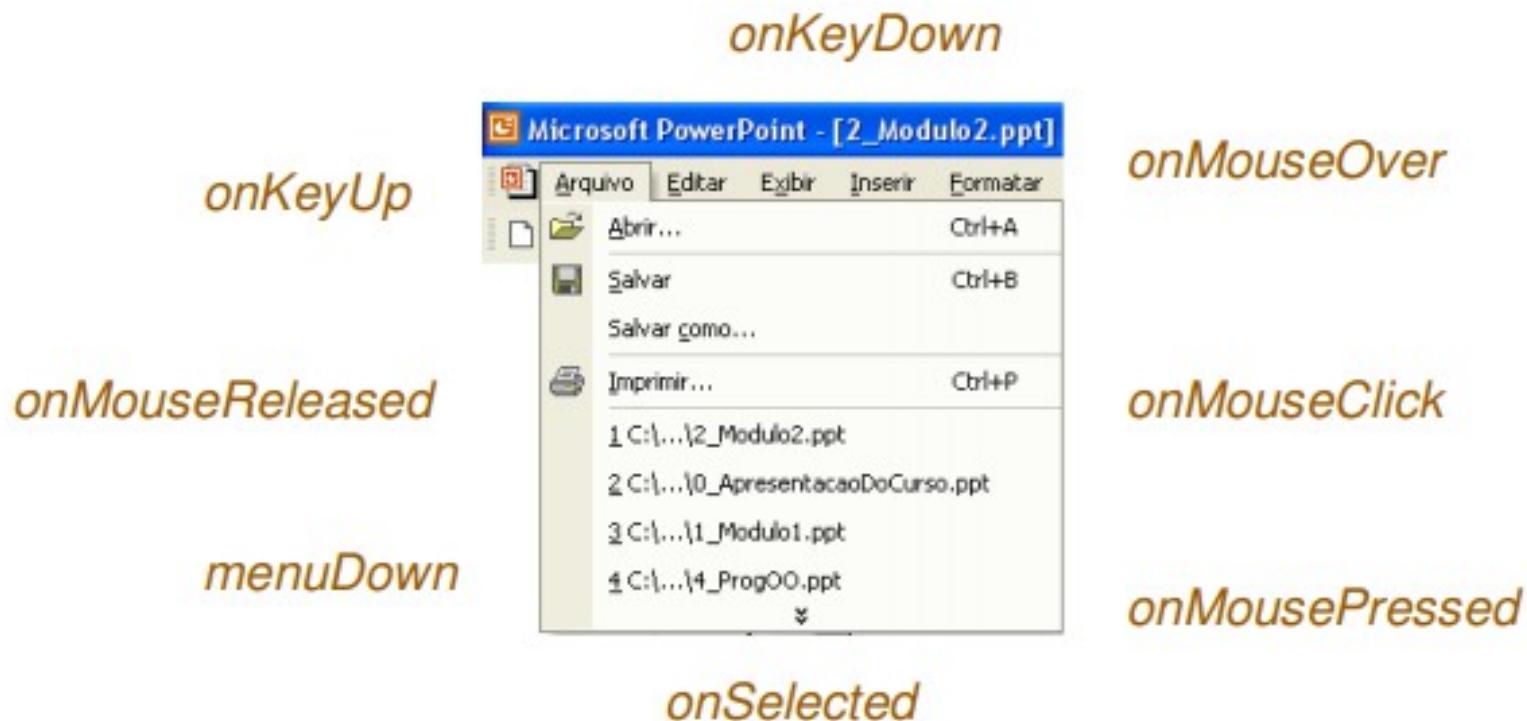
- Desacoplamento entre consumidores e produtores de eventos
- Escalabilidade: adição de novos observadores para eventos que já são produzidos
- Invocação implícita: O produtor de eventos não controla quem será notificado ou quando ele será notificado

Exemplo: relatório de impressão

- Produtores e consumidores são independentes
- Execução via procedimentos disparados via mudança de estados
- Escalabilidade no número de interessados
- Consumidores se registram nos Produtores
- Produtores notificam consumidores registrados



Ex: Interface Gráfica



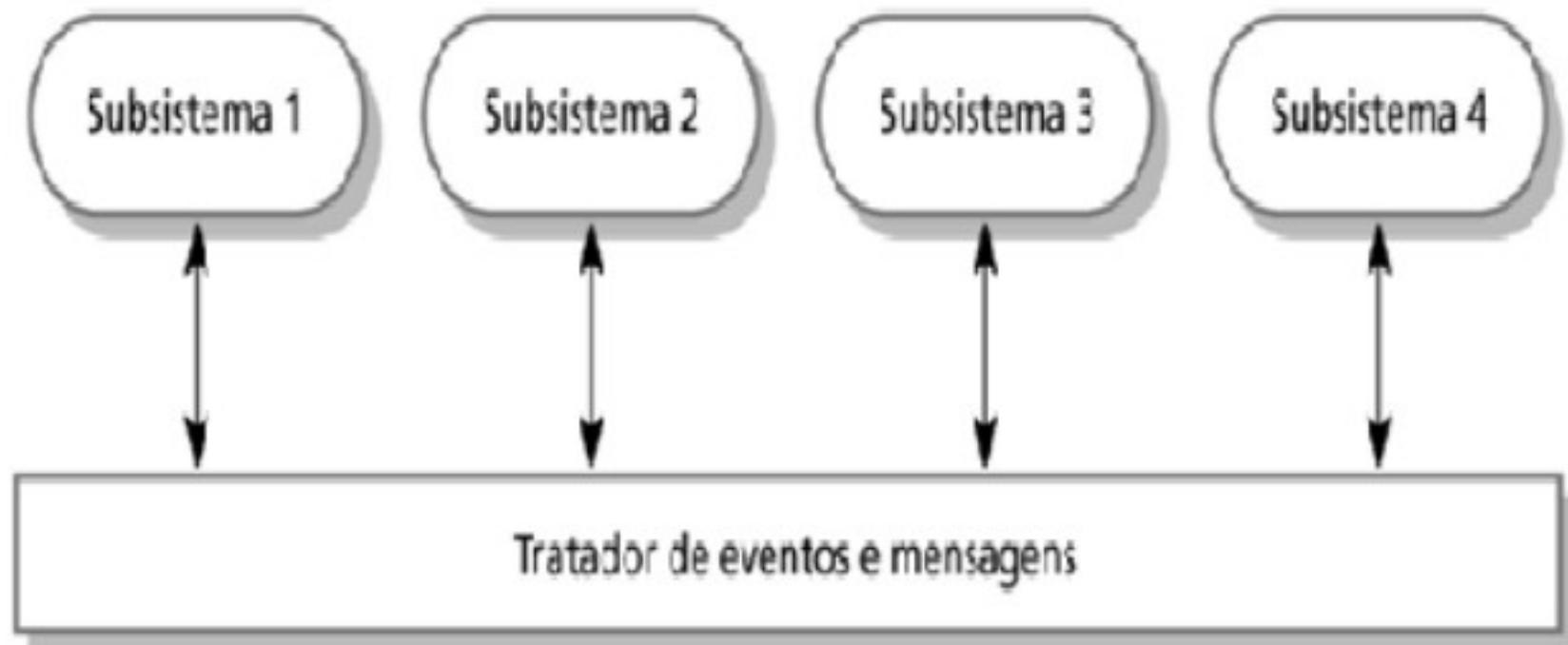
Sistemas orientados a eventos

- 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

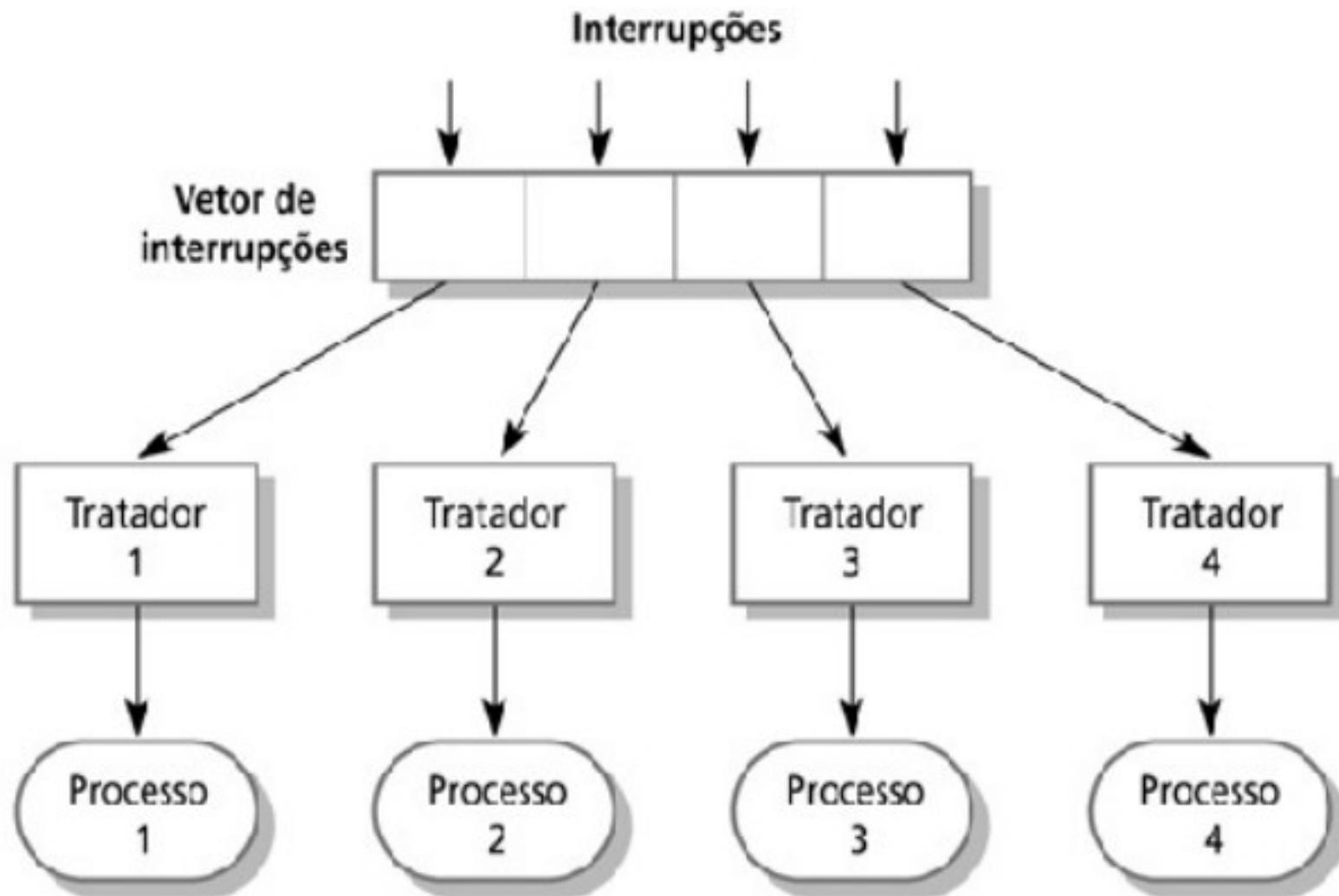
Publisher/Subscriber



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



Estilos arquiteturais

Máquina Virtual (Virtual Machine)

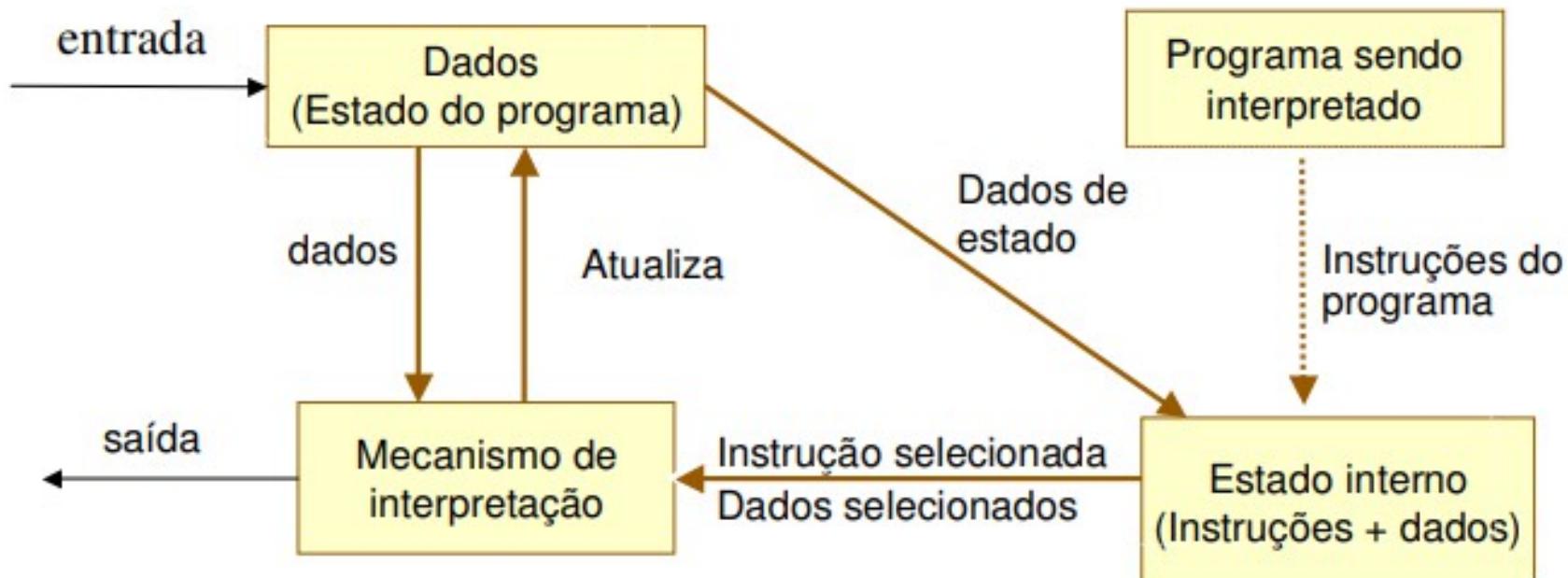
1. Interpretador
2. Baseado em Regras

1. Interpretador (Interpreter)

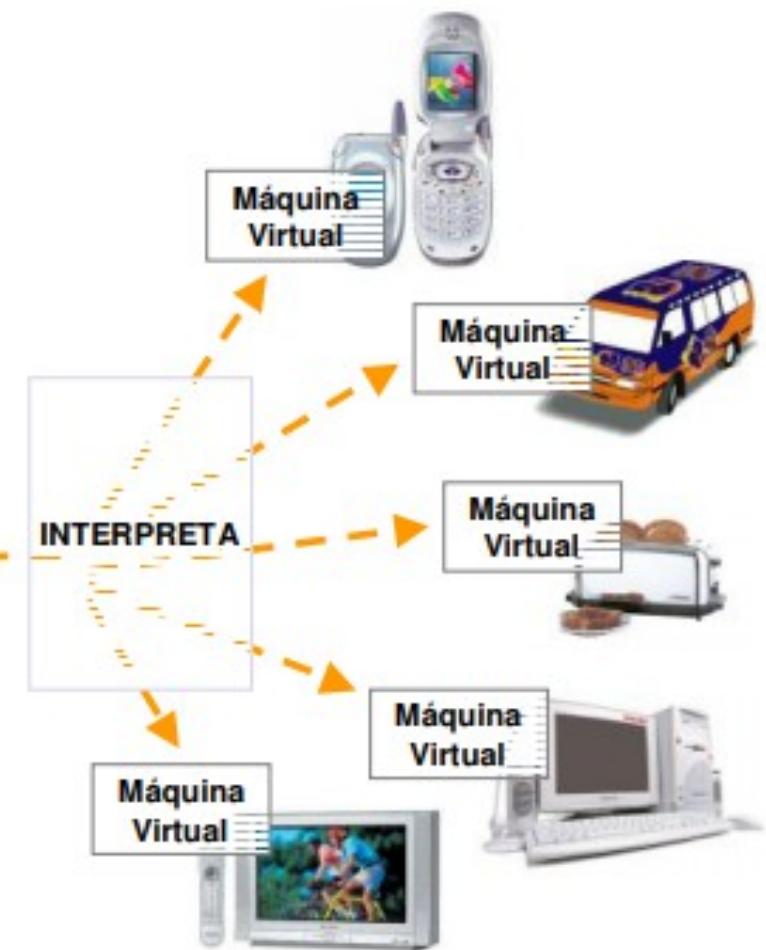
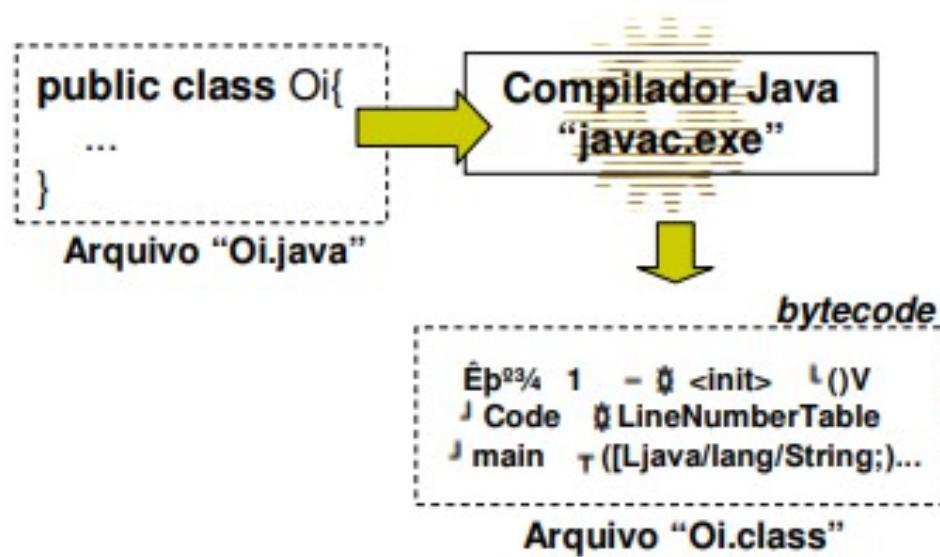
- Esta arquitetura inclui geralmente 4 componentes:
 - mecanismo de interpretação
 - uma memória que contém o pseudo-código a ser interpretado
 - representação do estado do mecanismo de interpretação
 - representação do estado atual do programa sendo simulado

A máquina tenta preencher a lacuna que existe entre o que o programa precisa e o que o hardware disponibiliza

Componentes de um Interpretador



Ex: Java VM



Características - Interpretador

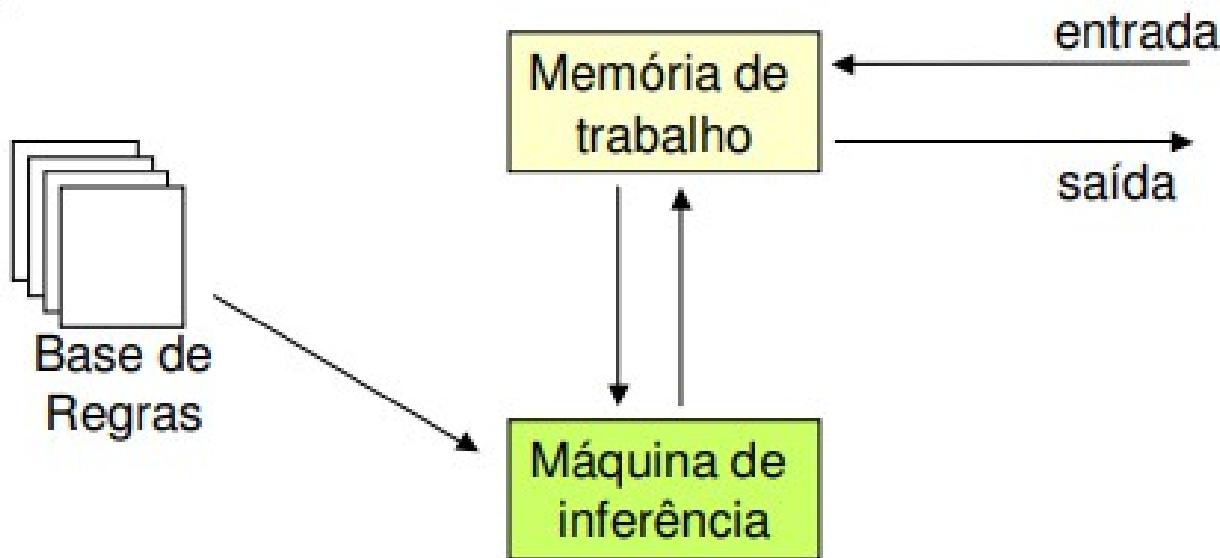
- Desvantagens: Desempenho
- Vantagens: Portabilidade

Algumas pesquisas apontam que algumas das linguagens interpretadas já conseguem ser mais rápidas que C

- Java, por exemplo

2. Baseado em Regras (Rule-Based)

- Conjunto de regras sobre um estado
- Definição do estado atual com base em dados de entrada
- Regras alteram o estado



Ex: Prolog, Sistemas Especialistas

**SE "HORA=21:00"
ENTÃO "AÇÃO=LANCHE"**

**SE "HORA=22:00"
ENTÃO "AÇÃO=LIBERAR"**

**SE "HORA<19:00"
ENTÃO "AÇÃO=ESPERAR"**

**SE "HORA=19:00"
ENTÃO "AÇÃO=COMEÇAR"**

Base de Regras

Memória de trabalho

HORA = ?
AÇÃO = ?

HORA=18:00

Máquina de
inferência

Estilos arquiteturais Centrado em Dados

1. Repositório

2. Blackboard

Centrado em Dados (*Data-centered*)

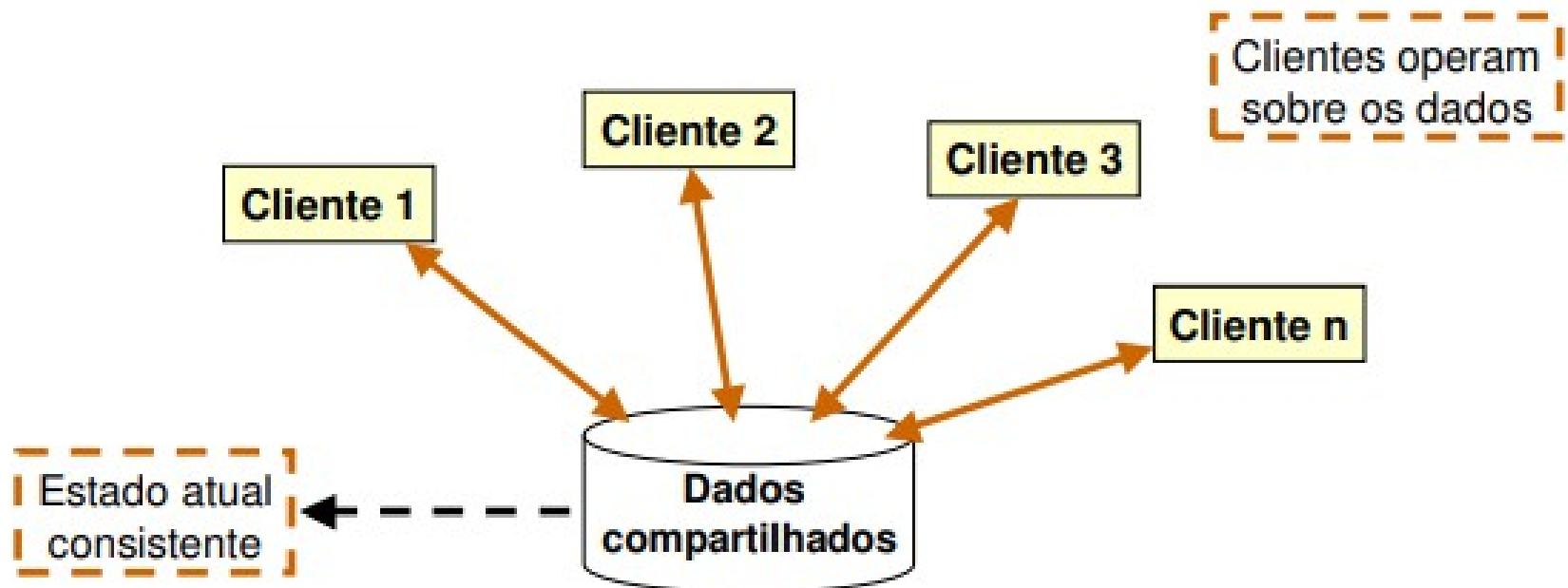
- A meta é a integração de dados
- Sistemas cujas partes precisam trocar dados com frequência
- Descreve o acesso e atualização de repositórios de dados amplamente acessíveis
- Existe um grande depósito de dados centralizado, manipulado por computações independentes

1. Repositório (*Repository*)

- 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

Repositório (*Repository*)

- Integridade, escalabilidade (novos clientes, novos dados)



Ex: Arquitetura de uma Ferramenta Case



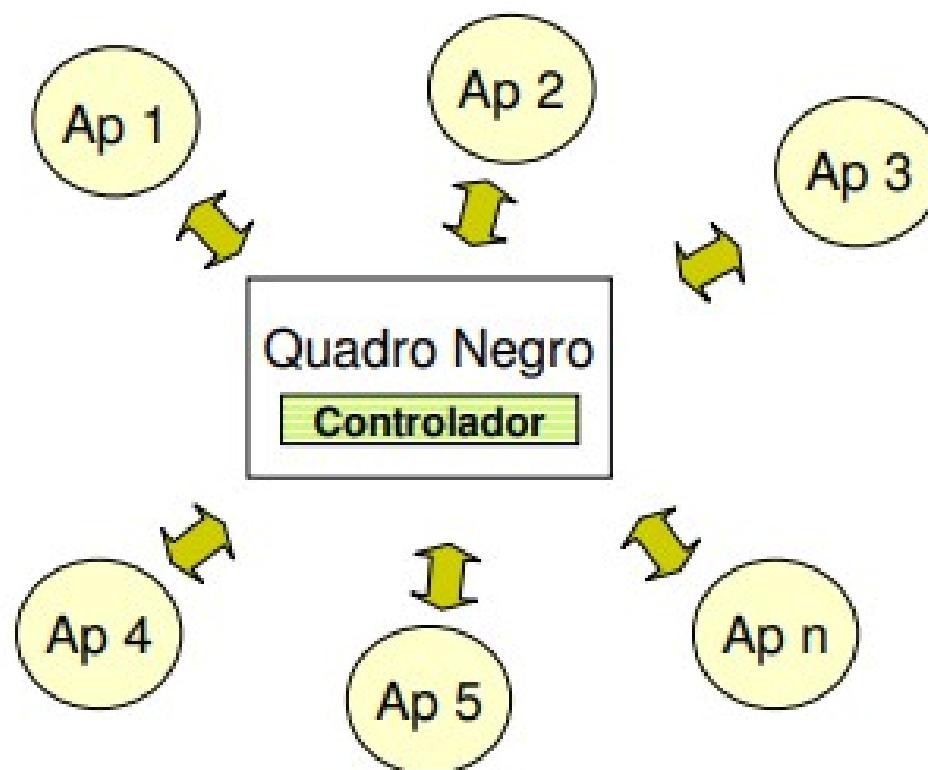
Características: Repositório

- 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 informações 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

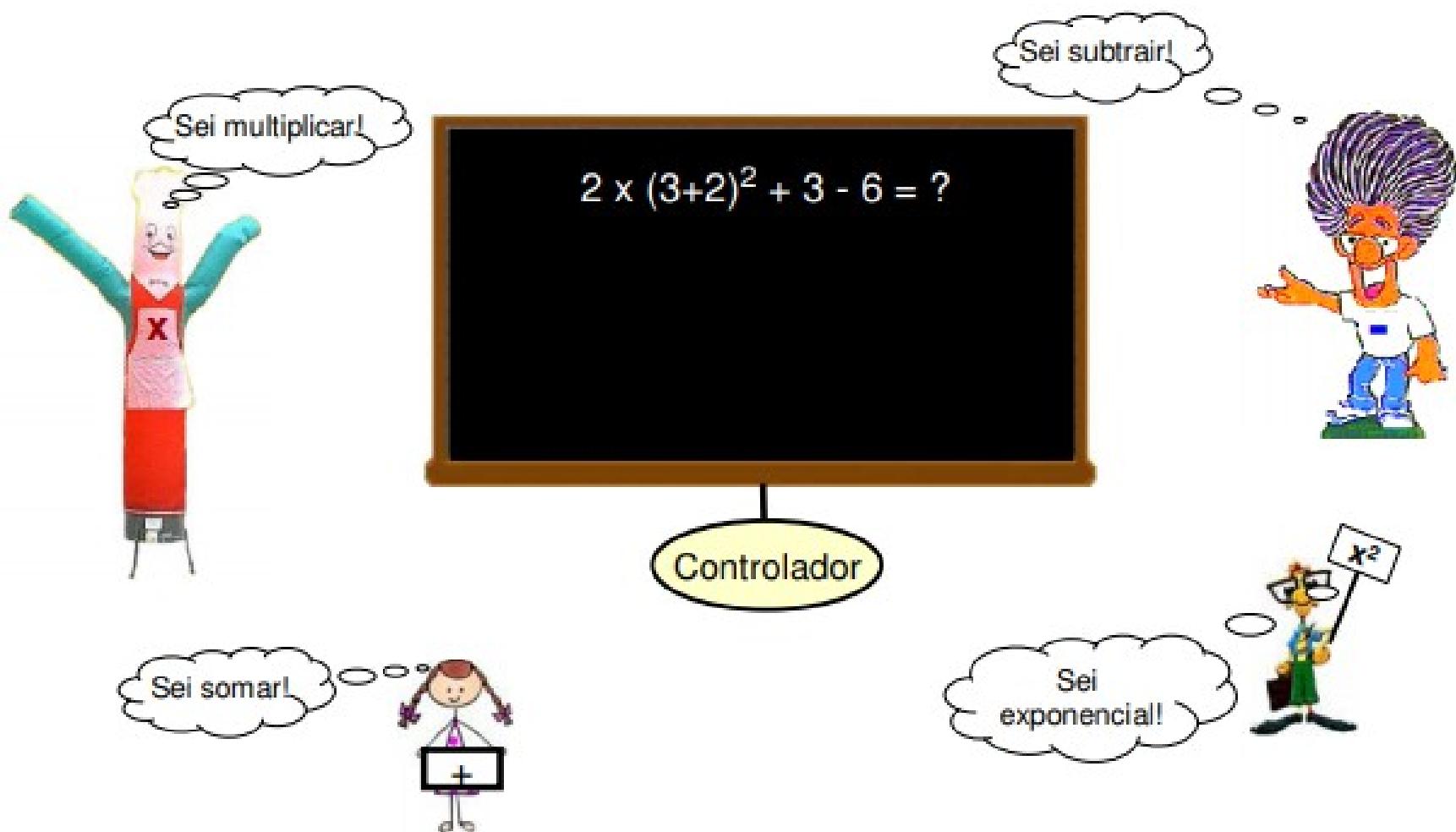
2. Quadro negro (*Blackboard*)

- O sistema é dividido em
 - blackboard: armazena dados – o vocabulário
 - base de conhecimento: subsistemas independentes, cada qual resolvendo aspectos específicos do problema
 - componente de controle: monitora mudanças no blackboard e decide as ações

Quadro negro (*Blackboard*)



Quadro negro (*Blackboard*)



Características: *Blackboard*

Usado em:

- Sistemas que não possuem estratégias de soluções determinísticas conhecidas e são baseados em soluções aproximadas ou parciais.
- Problemas que podem ser decompostos em sub-problemas e abrangem muitos domínios de conhecimento
 - Sistemas complexos - Resolução Distribuída de Problemas - RDP
 - Paradigma de agentes

Características: *Blackboard*

- Vantagens:
 - Ajuda a resolver problemas de experimentação
 - Suporte a mudanças e manutenção
 - Escalabilidade – aplicações independentes
 - Reúso de conhecimentos
 - Suporte: tolerância a falhas e robustez

Características: *Blackboard*

- Desvantagens
 - Dificuldades para testar por não usar algoritmos determinísticos
 - Nenhuma boa solução é garantida
 - Dificuldade em estabelecer uma boa estratégia de controle.
 - Baixa eficiência e alto esforço de desenvolvimento
 - Não suporta paralelismo

Seleção de Estilos

O que considerar?
Passos a seguir

Fluxo de Dados

- As interfaces entre os componentes são simples
- O sistema produz resultados simples e bem identificáveis que derivam diretamente da transformação seqüencial de uma entrada facilmente identificável
- A relação entre entrada e saídas é temporalmente independente

Fluxo de Dados

- Sequencial: transformações são seqüenciais
 - Existe uma única saída, resultante de uma única entrada de dados
- Filtros e Pipes:
 - A computação envolve transformações sobre uma cadeia de dados contínua
 - As transformações são incrementais. Uma transformação pode executar antes do término do passo anterior

Chamada/Retorno

- A ordem da computação é fixa
- Componentes não podem fazer progresso enquanto aguardando o resultado das chamadas

Chamada/Retorno

- Orientação a objetos:
 - Ocultamento da Informação: produz módulos similares, que no decorrer do desenvolvimento e teste se beneficiam do uso de herança

Chamada/Retorno

- Camadas:
 - As tarefas do sistema podem ser particionadas entre: específicas da aplicação, e genéricas a muitas aplicações, mas específicas à plataforma subjacente
 - Portabilidade é importante
 - Pode-se usar uma infra-estrutura de computação pré-existente

Componentes Independentes

- O sistema executa em uma plataforma multi-processada (ou que pode vir a ser no futuro)
- O sistema pode ser estruturado como um conjunto de componentes fracamente acoplados, de modo que um componente pode fazer progressos de forma independente dos outros.
- Ajuste de desempenho é importante, seja através da re-alocação de tarefas a processos, seja através da re-alocação de processos a processadores

Componentes Independentes

- Processos Comunicantes:
 - Objetos Distribuídos: OO + Componentes Independentes
 - Redes de filtros: Data-Flow + Componentes Independentes
 - Cliente-Servidor: As tarefas podem ser divididas entre geradores de pedidos (ou consumidores de dados) executores de pedidos (ou produtores de dados)

Componentes Independentes

- Baseada em Evento
 - Quando é necessário desacoplar consumidores e produtores de eventos
 - Quando é necessário escalabilidade, e permitir a adição de novos processos ao sistema, os quais serão integrados a eventos já sinalizados no sistema

Centrada em Dados

- As questões importantes são o armazenamento, representação, gerenciamento e recuperação de uma grande quantidade de dados persistentes

Seleção de estilos –Passos a seguir

1. Identificar os principais elementos da arquitetura
2. Identificar o estilo arquitetural dominante
3. Considerar responsabilidades adicionais associadas com a escolha do estilo
4. Modificar o estilo para atingir objetivos adicionais

1. Identificar os principais elementos da arquitetura

- Cada elemento arquitetural tem um estilo arquitetural dominante que reflete as qualidades importantes que devem ser alcançadas no contexto daquele elemento
- A escolha do estilo arquitetural dominante é baseada nos principais elementos arquiteturais
- Os atributos de qualidade sobre cada elemento arquitetural podem acarretar a utilização ou não de um estilo

2. Identificar o estilo arquitetural dominante

- O estilo dominante pode ser modificado para alcançar objetivos particulares
- Se nenhum estilo conhecido parece ser apropriado, o arquiteto deve projetar e documentar um novo estilo
- As decisões sobre escolhas baseadas em atributos de qualidade dentro de um estilo devem ser documentadas

3. Considerar responsabilidades adicionais associadas com a escolha do estilo

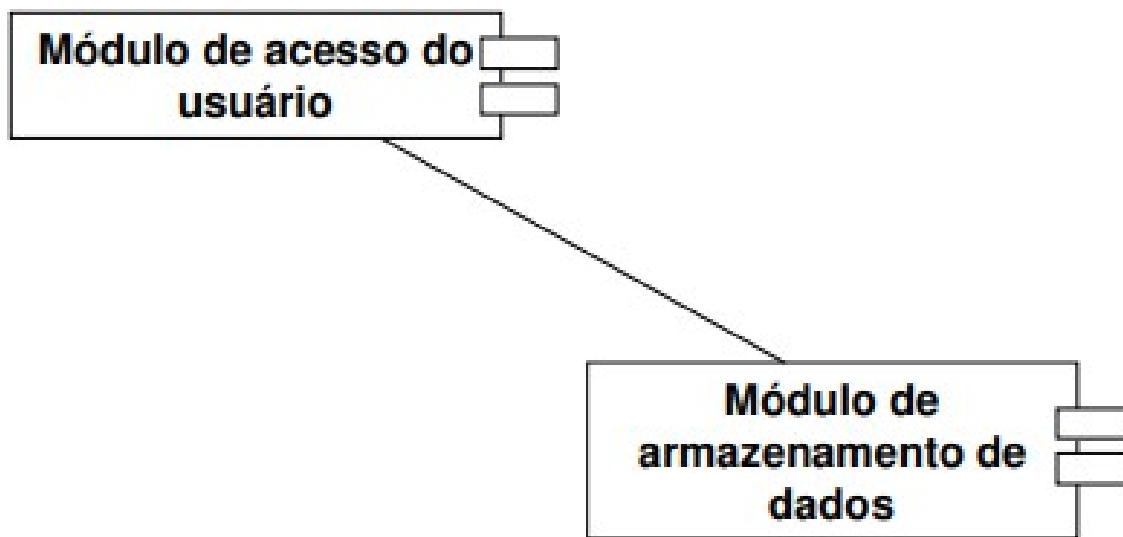
- A escolha de um estilo arquitetural introduzirá responsabilidades adicionais
- Por exemplo:
 - Se o estilo é “Quadro negro”, então devem-se gerenciar os mecanismos para o controle do quadro negro
 - Se o estilo é “cliente-servidor”, devem-se gerenciar os protocolos de interação
- Responsabilidades adicionais devem ser atribuídas a elementos arquiteturais existentes ou a novos elementos criados para este fim.

4. Modificar o estilo para atingir objetivos adicionais

- Pode-se alterar o estilo arquitetural caso este necessite ser adaptado devido a atributos de qualidade ou até mesmo funcionalidade

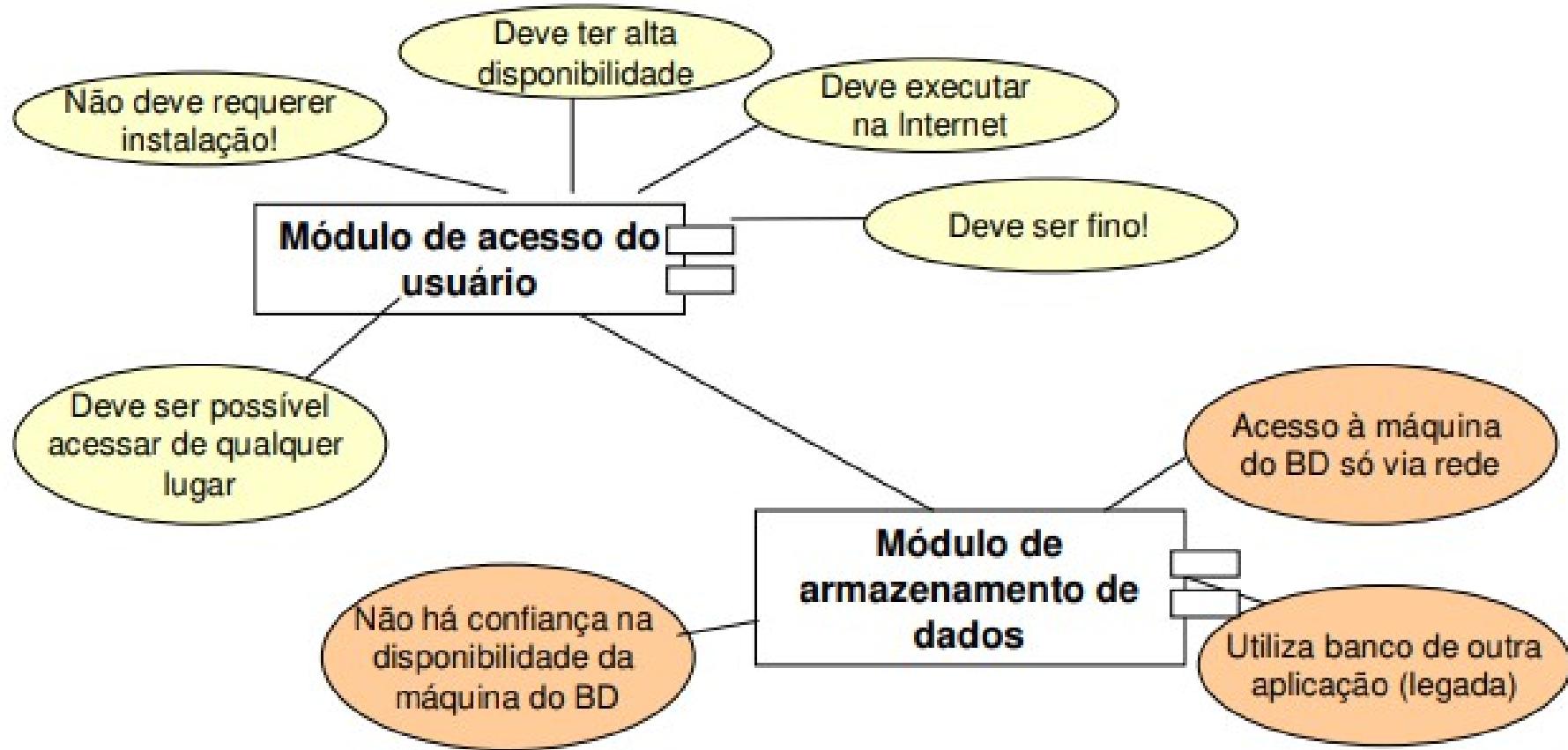
Exemplo: Sistema de matrícula

- 1. Identificar os principais elementos da arquitetura



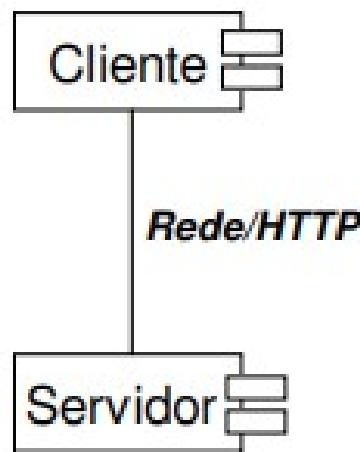
Exemplo: Sistema de matrícula

- 2. Identificar o estilo arquitetural dominante



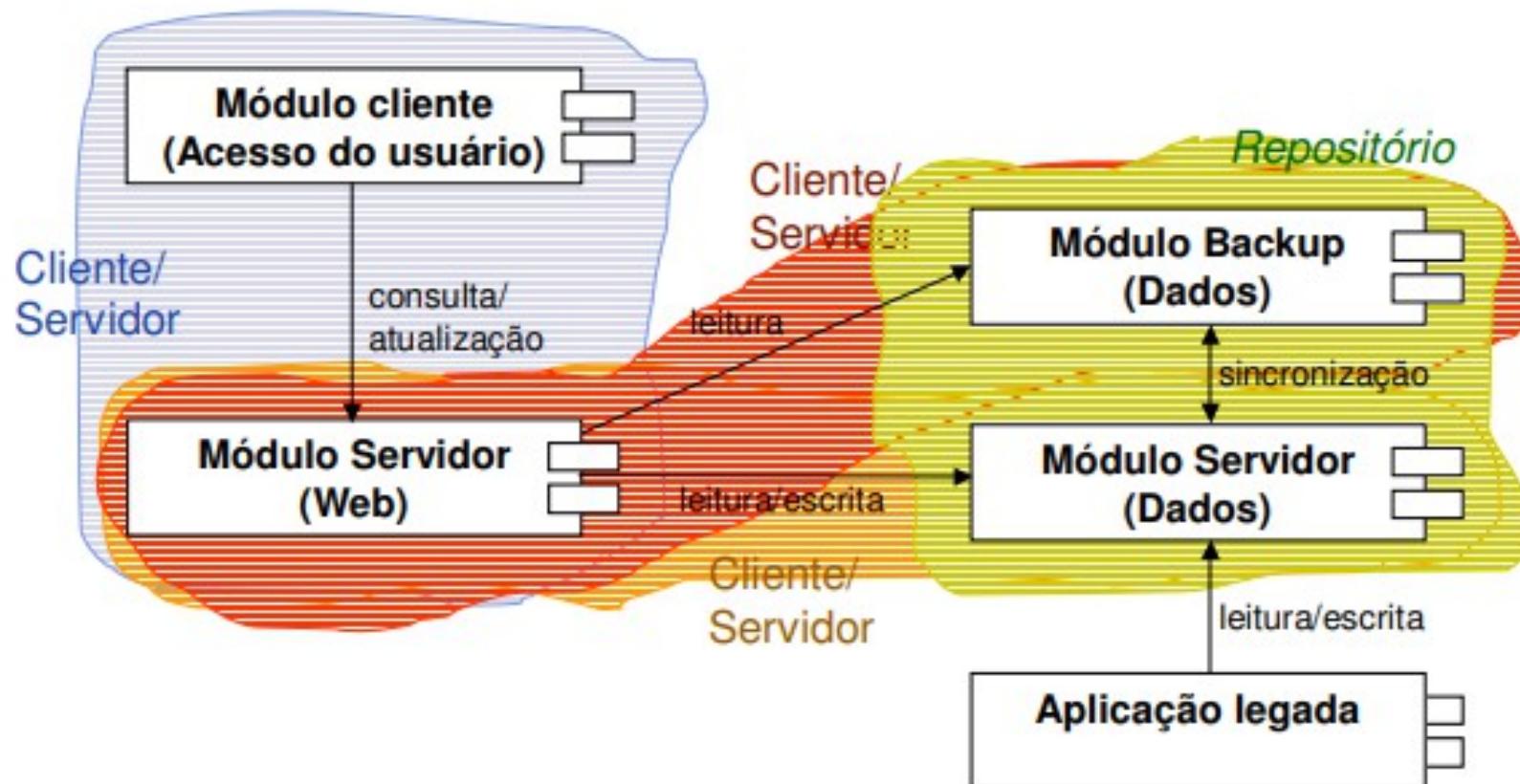
Exemplo: Sistema de matrícula

- 3. Considerar responsabilidades adicionais associadas com a escolha do estilo
 - Estilo dominante escolhido: Cliente-servidor
 - Estilo secundário: Repositório
 - Responsabilidades: considerar protocolo de comunicação



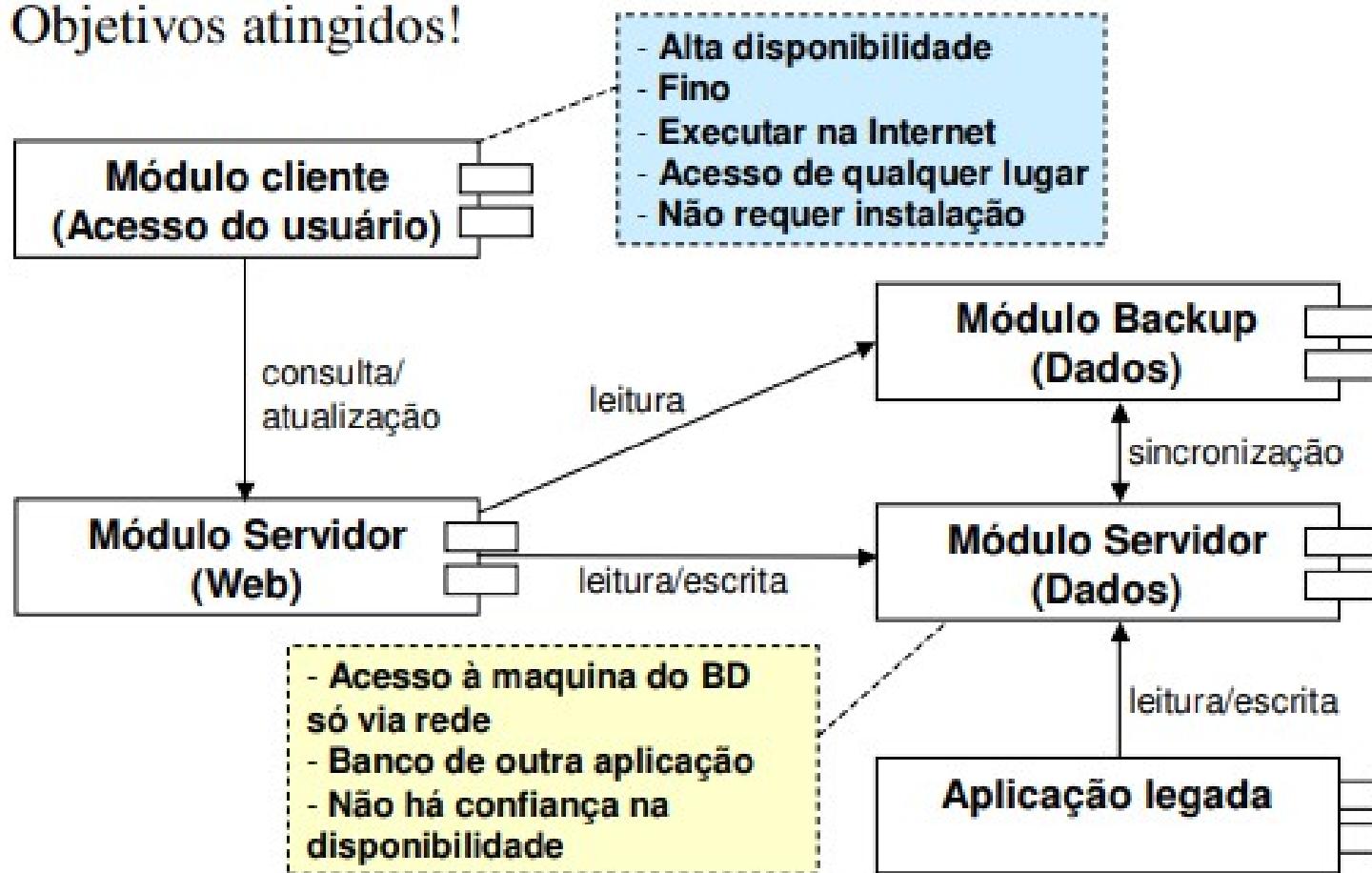
Exemplo: Sistema de matrícula

- 4. Modificar o estilo para atingir objetivos adicionais
 - Cliente servidor de 2 camadas e repositório com *backup*



Exemplo: Sistema de matrícula

- 4. Modificar o estilo para atingir objetivos adicionais
 - Objetivos atingidos!



Design e Arquitetura: Projeto de Dados

Representação do relacionamento lógico entre elementos de dados individuais. Determina a organização, métodos de acesso, associações e alternativas de processamento.

- ✓ Parte dos dados levantados e representados na fase de análise:
 - Dicionário de Dados
 - Fluxo, Conteúdo e Estrutura

Design e Arquitetura: Projeto de Dados

Atividades do Projeto de Dados:

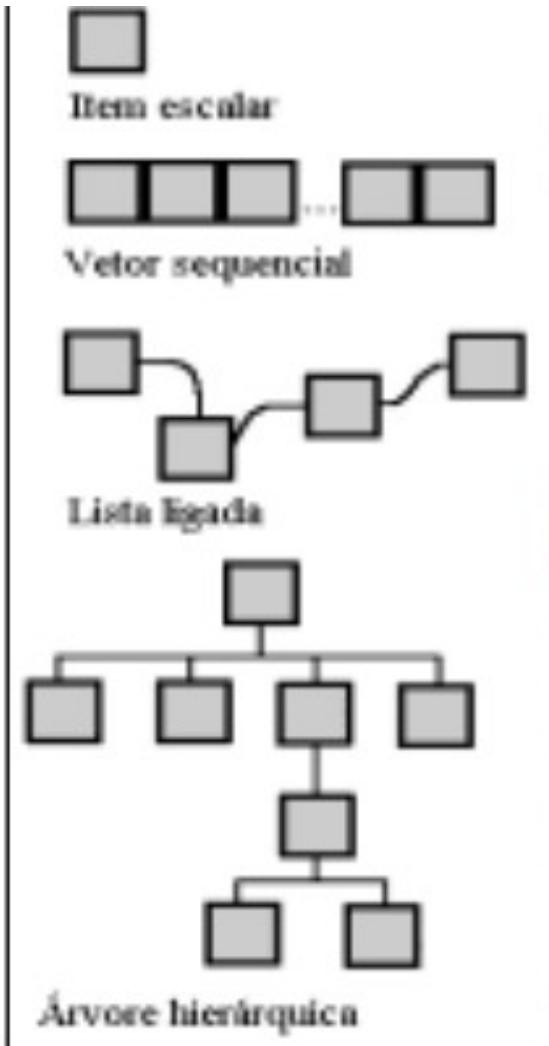
- ✓ Selecionar representações de dados;
- ✓ Estudar e escolher estruturas de dados que permitam a implementação mais adequada;
- ✓ Caracterizar a Abrangência (escopo) dos Dados
 - Local (componente), Partes do software ou Global
 - Caracteriza a Persistência dos Dados
 - Persistentes (B.Dados)
 - Não Persistentes (Dados em memória, estruturas de manipulação/intermediária)..

Design e Arquitetura: Projeto de Dados

Estruturas/Elementos de Dados

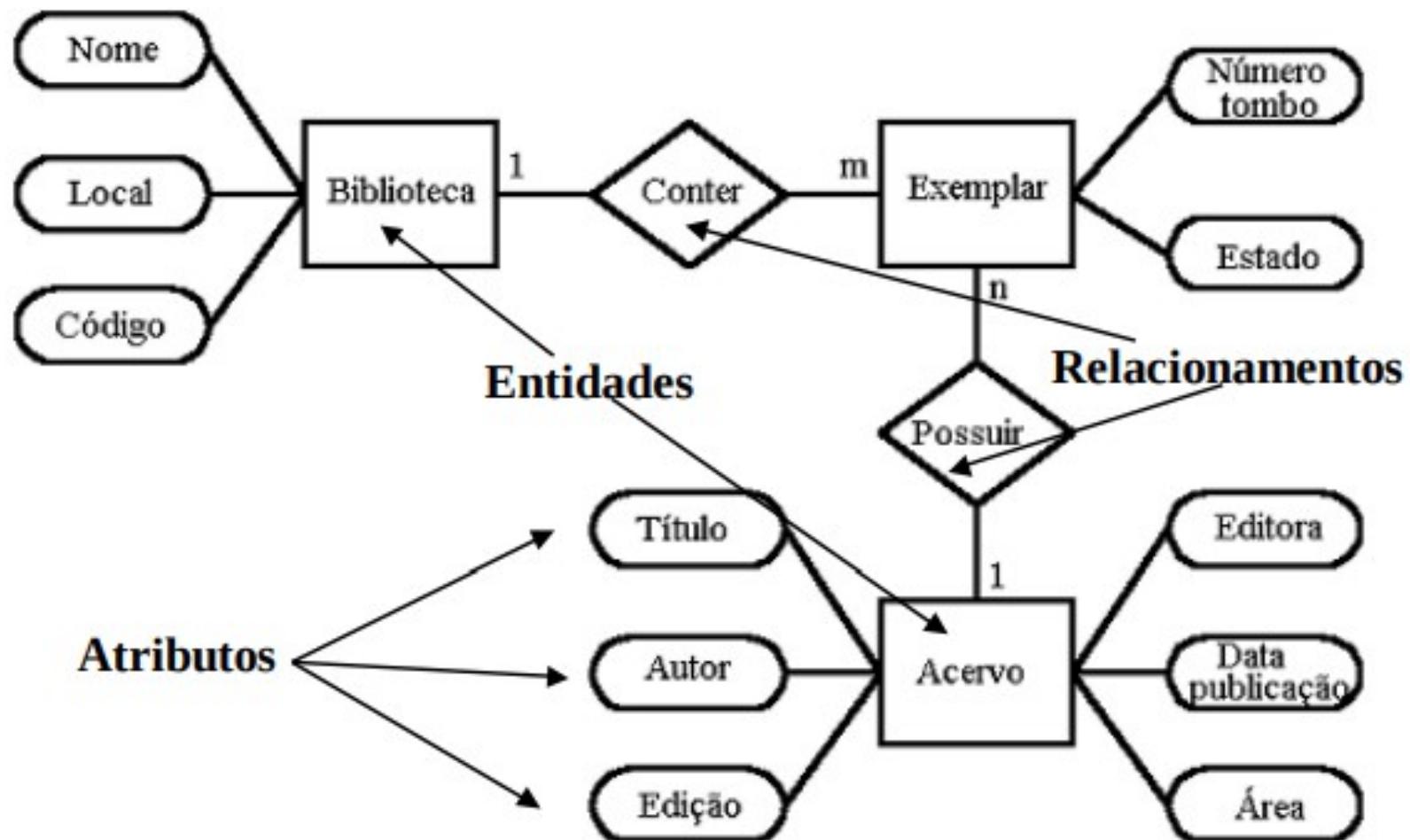
Comuns:

- ✓ Itens Elementar (Tipos Primitivos)
- ✓ Listas Lineares
 - Gerais
 - Pilhas
 - Filas
- ✓ Lista não lineares
 - Árvores
 - Grafo



Design e Arquitetura: Projeto de Dados

Modelo de Entidade-Relacionamento (MER)



Design e Arquitetura: Projeto Arquitetural

Princípios / Padões / Estilos

| | | Creational | Structural | Behavioral |
|----------|--------|--|--|---|
| By Scope | Class | <ul style="list-style-type: none">• Factory Method | <ul style="list-style-type: none">• Adapter (class) | <ul style="list-style-type: none">• Interpreter• Template Method |
| | Object | <ul style="list-style-type: none">• Abstract Factory• Builder• Prototype• Singleton | <ul style="list-style-type: none">• Adapter (object)• Bridge• Composite• Decorator• Façade• Flyweight• Proxy | <ul style="list-style-type: none">• Chain of Responsibility• Command• Iterator• Mediator• Memento• Observer• State• Strategy• Visitor |

Design e Arquitetura: Projeto Arquitetural

Princípios / Padões / Estilos

- **Exemplos:**

- Arquitetura em camadas (tier – camadas físicas)
 - Cliente – Servidor
 - 3 Camadas – Apresentação, Negócio e Acesso a Dados
- Arquitetura modelo-visão-controlador – MVC (layer – camadas lógicas)
- Arquitetura Orientada a Serviço (SOA) – “estilo arquitetural”

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

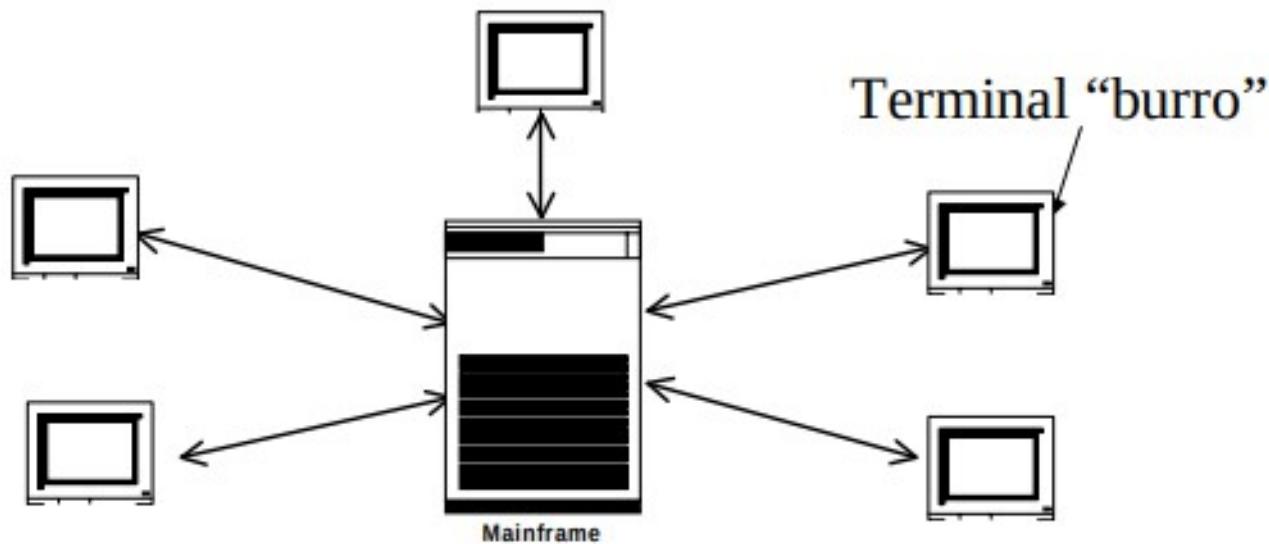
Visa modelar a estrutura completa do software e as maneiras fornecidas para manter a integridade conceitual de um sistema:

Arquiteturas Tradicionais:

- Centralizada
- Parcialmente Distribuída
- Cliente-Servidor (2 Camadas)
- Cliente-Servidor (3 Camadas)
- Distribuída (Multi-Camadas)

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais: Arquitetura Centralizada

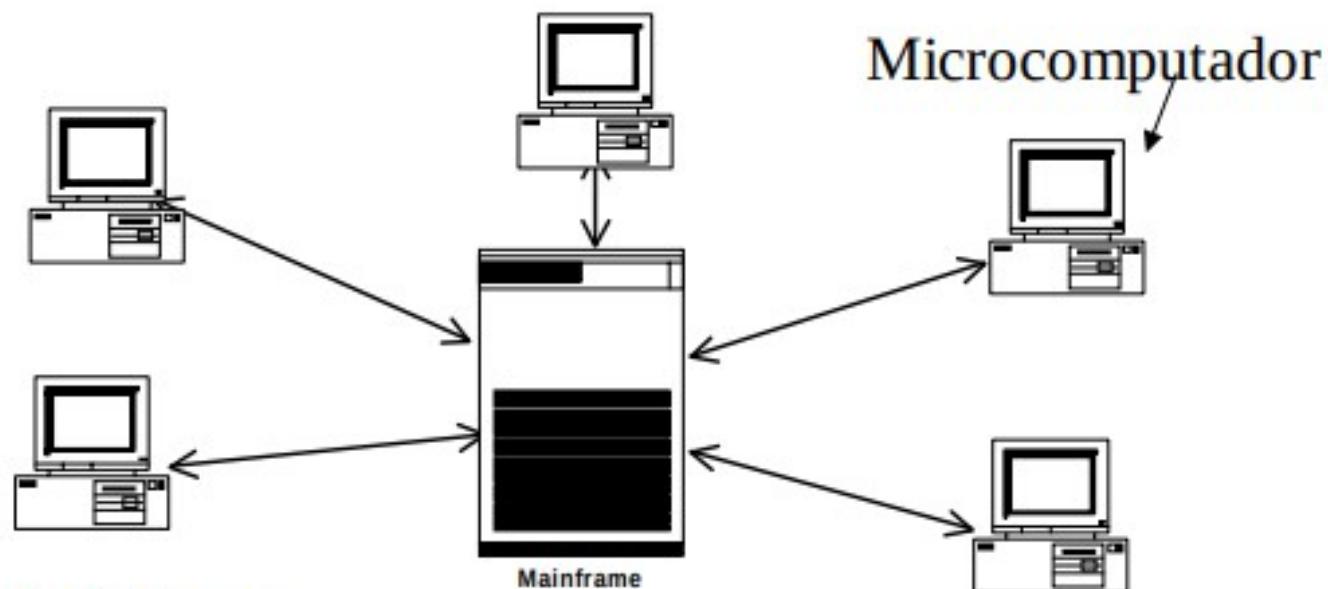


✓ Características:

- Processamento centralizado no Mainframe;
- Terminais Burros (sem processamento);
- Redes Corporativas;
- Software uso Departamental (Baixa Integração).

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais: Arquitetura Parcialmente Distribuída



Características:

- Um pouco de processamento departamental;
- Micros com algumas aplicações locais;
- Rede Corporativa conectando micro-mainframe;
- Software Departamental com maior Integração;
- Início de Integração entre parceiros (EDI).

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Big Ball Mud

Falta de estrutura arquitetural aparente, crescimento descontrolado, código mal escrito. Impossível garantir qualidade!

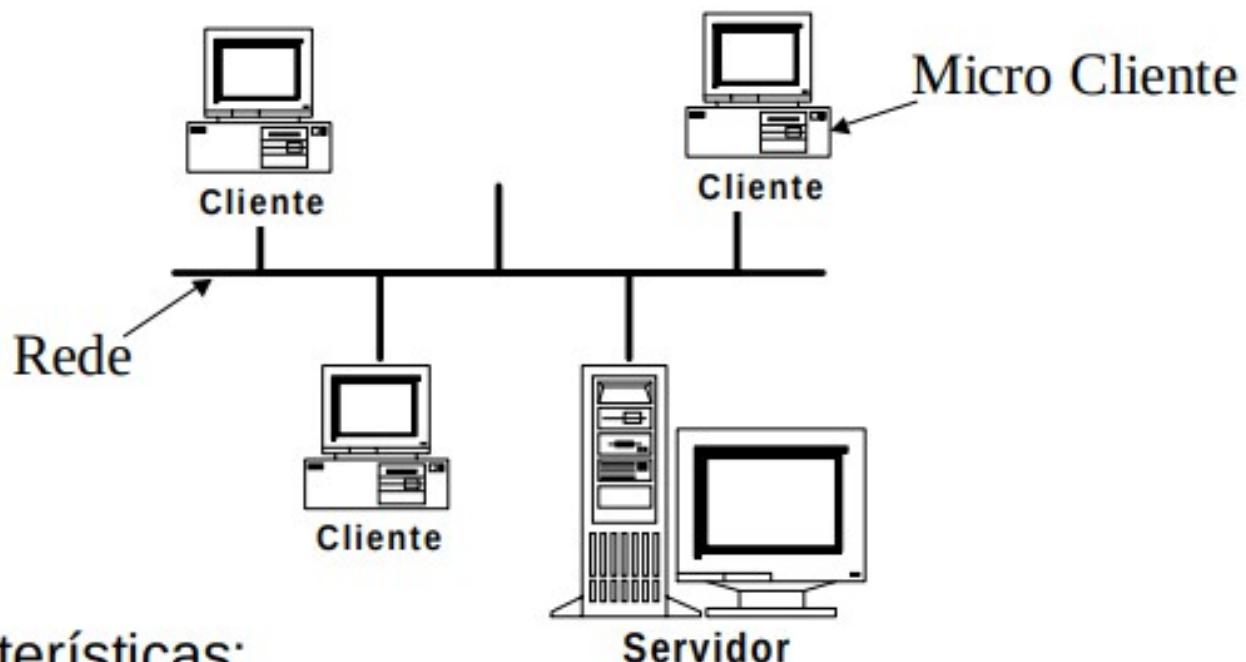


Figura 40 - Estilo Arquitetural - Grande bola de lama (*Big Ball Mud*)

Fonte: <http://blogdozanei.wordpress.com/2012/05/04/futebol-jornalismo-e-estomago/>

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais: Cliente-Servidor 2 Camadas

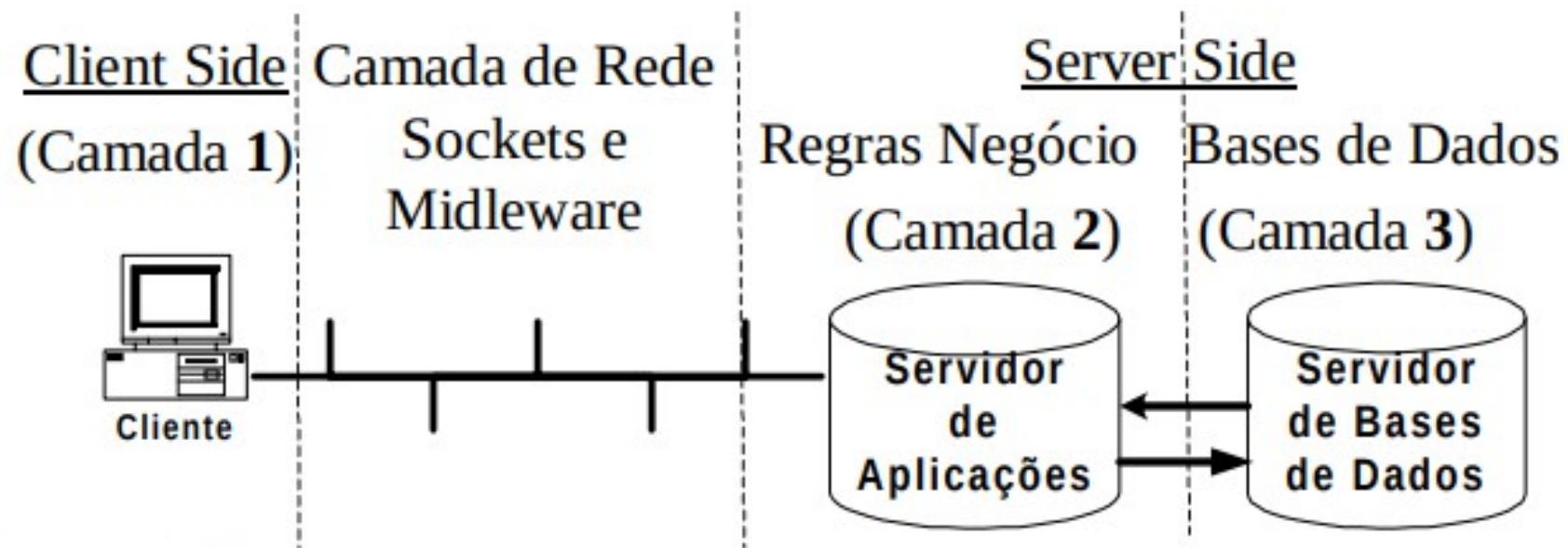


Características:

- Boa parte do processamento local
- Micros com algumas aplicações locais
- Redes LAN's ou WAN's
- Integração entre Parceiros (cliente-servidor)

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais: Cliente-Servidor 3 Camadas

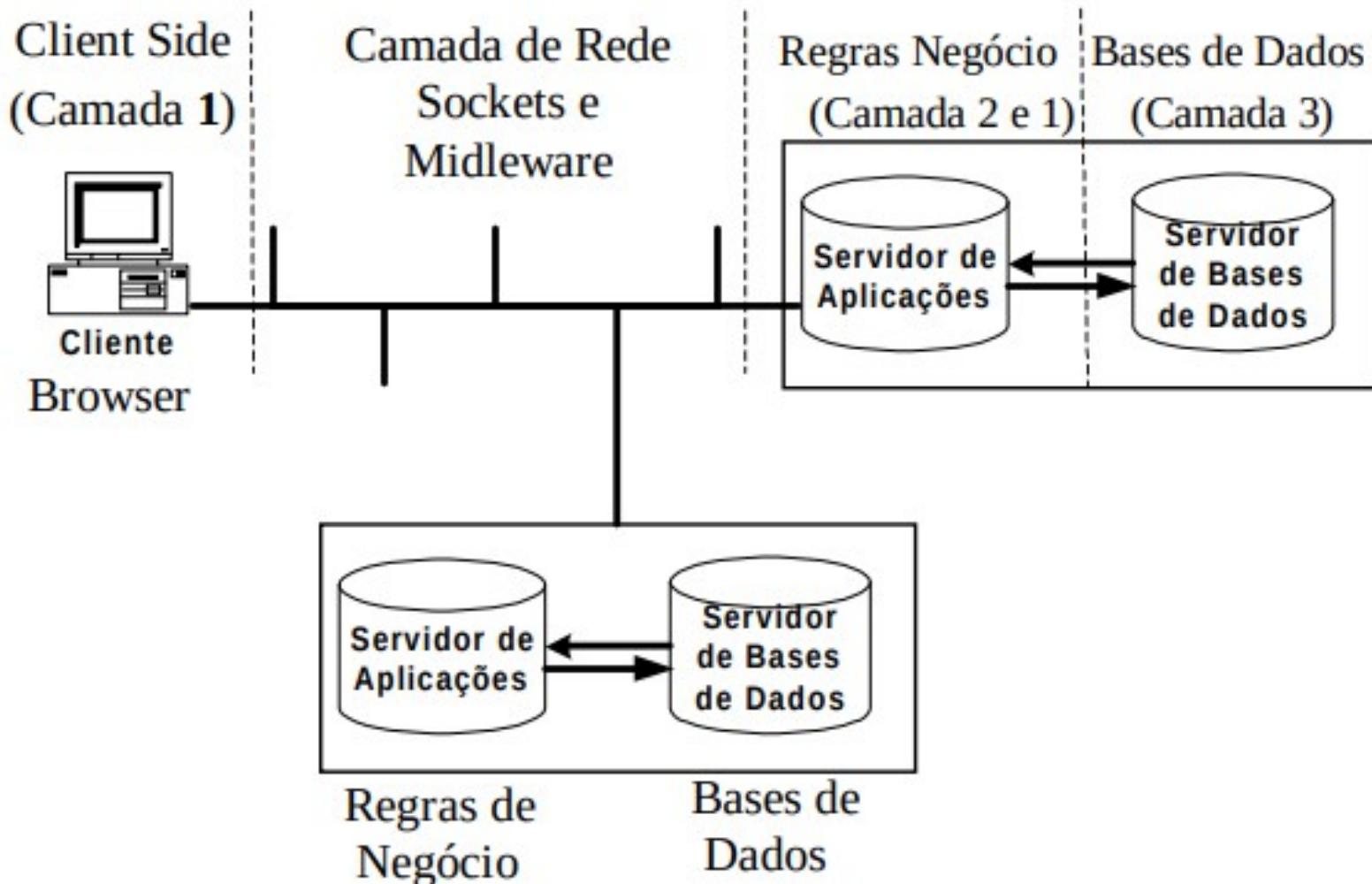


Características:

- Mais capacidade de Processamento Distribuído
- Internet/Intranet como infra-estrutura de rede
- 2 Níveis de Servidor
- Internet Applications / Software Integrados(ERPs)

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais: Arquitetura Distribuída Multi-Camadas



Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais: A System of Patterns

- **From Mud to Structure**
 - Layers
 - Pipes and Filters
 - Blackboard
- **Distributed Systems**
 - Broker
- **Interactive Systems**
 - Model-View-Controller
 - Presentation-Abstraction-Control
- **Adaptable Systems**
 - Microkernel
 - Reflection

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais: Patterns for Concurrent, Networked Objects

- **Service Access & Configuration**
 - Wrapper Facade, Component Configurator, Interceptor, Extension Interface
- **Event Handling**
 - Reactor, Proactor, Asynchronous Completion Token, Acceptor-Connector
- **Synchronization**
 - Scoped Locking, Strategized Locking, Thread-Safe Interface, Double-Checked Locking Optimization
- **Concurrency**
 - Active Object, Monitor Object, Half-Sync/Half-Async, Leader/Followers, Thread-Specific Storage

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais: Patterns of Enterprise Application Architecture

Domain Logic Patterns

- Transaction Script
- Domain Model
- Table Module
- Service Layer

Data Source Architectural Patterns

- Table Data Gateway
- Row Data Gateway
- Active Record
- Data Mapper

Object-Relational Behavioral Patterns

- Unit of Work
- Identity Map
- Lazy Load

Object-Relational Structural Patterns

- Identity Field
- Foreign Key Mapping
- Association Table Mapping
- Dependent Mapping
- Embedded Value
- Serialized LOB
- Single Table Inheritance
- Class Table Inheritance
- Concrete Table Inheritance
- Inheritance Mappers

Object-Relational Metadata Mapping Patterns

- Metadata Mapping
- Query Object
- Repository

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais: Patterns of Enterprise Application Architecture

Web Presentation Patterns

- Model View Controller
- Page Controller
- Front Controller
- Template View
- Transform View
- Two Step View
- Application Controller

Distribution Patterns

- Remote Facade
- Data Transfer Object

Offline Concurrency Patterns

- Optimistic Offline Lock
- Pessimistic Offline Lock
- Coarse-Grained Lock
- Implicit Lock

Session State Patterns

- Client Session State
- Server Session State
- Database Session State

Base Patterns

- Gateway
- Mapper
- Layer Supertype
- Separated Interface
- Registry
- Value Object
- Money
- Special Case
- Plugin
- Service Stub
- Record Set

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Layered Style

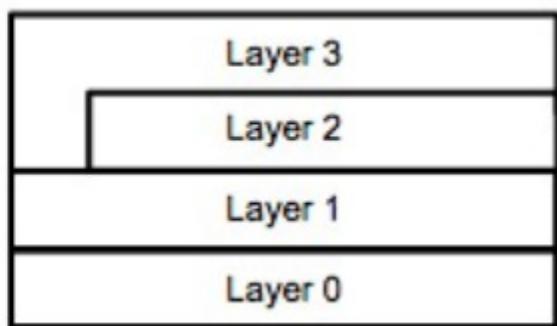
Pilha ordenada de camadas que usam apenas camadas adjacentes. Há casos especiais onde o exemplo mostra a camada 3 acessando a camada 1.

✓ **Vantagens**

- Facilidade de compreensão
- Facilidade de manutenção
- Desenvolvimento independente
- Facilidade de Reutilização

✓ **Desvantagens**

- Duplicação de funcionalidade
- Overhead de implementação e desempenho
- Em alguns casos, é difícil estruturar um sistema através de camadas
 - Daí ocorre sua violação (implementação fora do padrão em camadas)



Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Piper-and-Filter

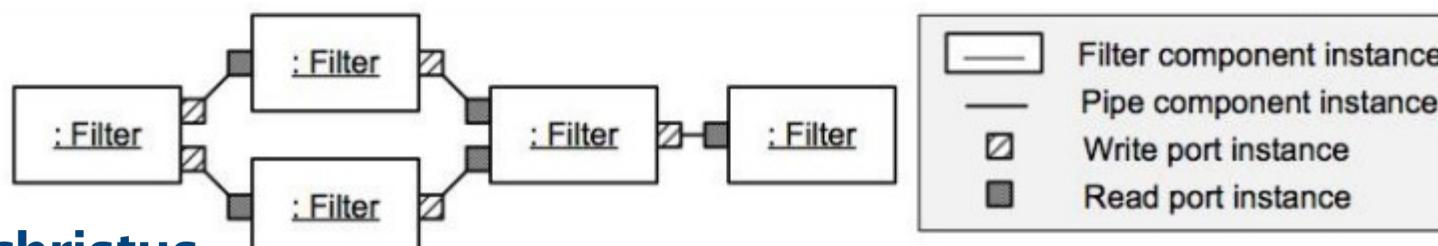
Cada filtro processa incrementalmente sua entrada e escreve sua saída. Os filtros são independentes e somente interagem com outros através de suas vias e não podem compartilhar estados

✓ **Vantagens**

- Encapsulamento
- Alta Coesão
- Reuso
- Interagem com filtros de forma limitada, logo, baixo acoplamento

✓ **Desvantagens**

- Processamento em lotes
- Pode ter baixa performance
- Não indicado para aplicações interativas
- Tratamento de erros é difícil e a segurança fica em jogo



Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Batch-Sequential

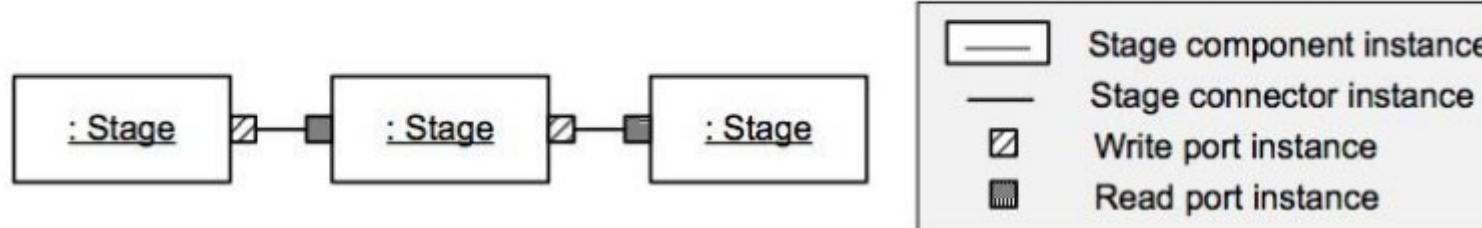
Cada estágio lê completamente sua entrada e escreve sua saída de uma vez

✓ **Vantagens**

- Util para problemas que podem ser resolvidos em vários passos de forma sequencial

✓ **Desvantagens**

- Não permite processamento em paralelo
- Pode ter problemas com performance



Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Model-centered

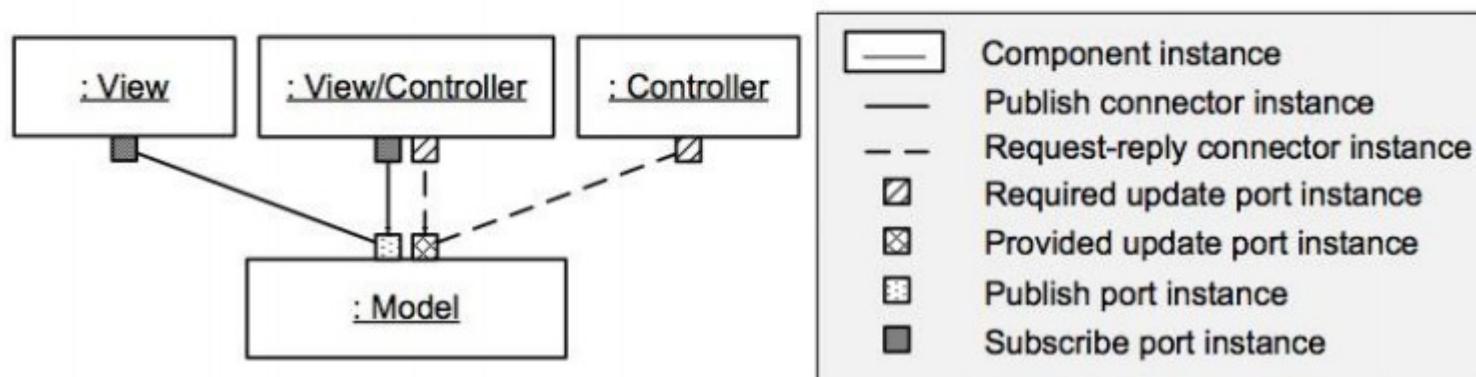
Os componentes dependem apenas do componente modelo e interagem através dele. Baixo acoplamento.

✓ **Vantagens**

- Manutenção facilitada
- A camada de Interface pode ser alterada facilmente
- Facilidade na Integração com outros sistemas

✓ **Desvantagens**

- Visões e controles interagem somente através do modelo



Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Publish-subscribe

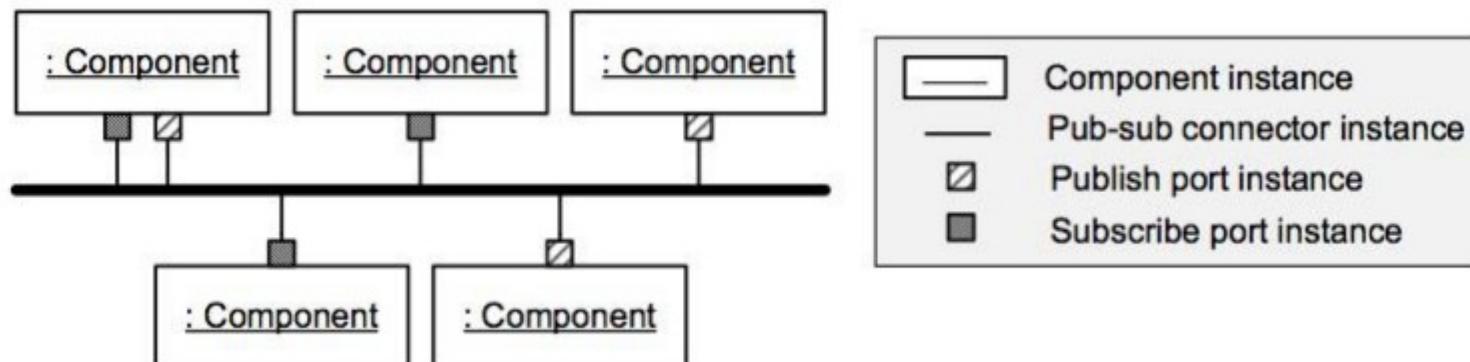
Também conhecido como baseado em eventos (*event-based*). Os componentes independentes publicam eventos e fazem “assinaturas” para acompanhar outros eventos.

✓ **Vantagens**

- Relativamente fácil adicionar/remover e alterar componentes
- A independência dos componentes proporciona reusabilidade

✓ **Desvantagens**

- Não há garantia que algum componente irá responder há um evento
- Não há garantia que a ordem de resposta é a ideal
- O tráfego de eventos pode ser variável (desempenho pode ser afetado)



Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Client-server

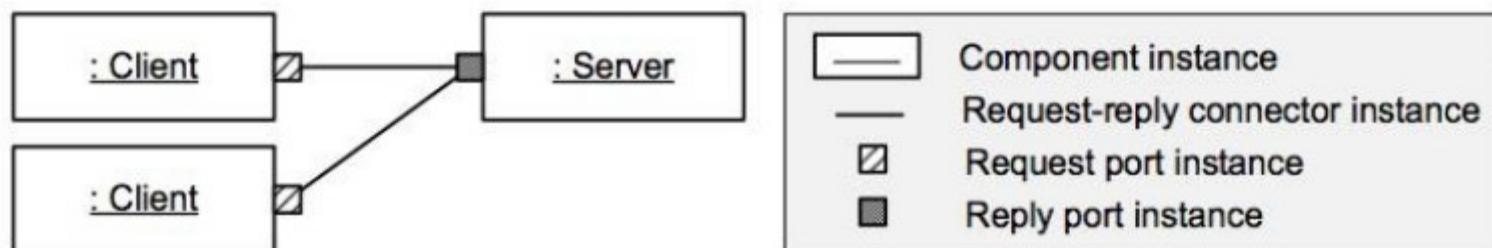
Cientes requisitam serviços aos servidores. Geralmente de modo síncrono e por meio de um conector de requisição e resposta

✓ **Vantagens**

- É a base de muitas aplicações web atuais
- Separação de interesses
- Fácil expandir a infraestrutura
- Permite distribuir o processamento

✓ **Desvantagens**

- Disponibilidade é um fator crítico
- O gerenciamento de redundância é complicado
- Servidores podem fazer pedidos a outros servidores, mas não a outros clientes
- O cliente precisa conhecer os serviços oferecidos pelo servidor
- O cliente precisa saber como contactar os servidores



Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Peer-to-Peer

Cada nó pode requisitar ou fornecer serviços para outro nó

✓ **Vantagens**

- Reusabilidade
- Reconfigurabilidade (modificabilidade)

✓ **Desvantagens**

- Filtros independentes
- Processamento incremental

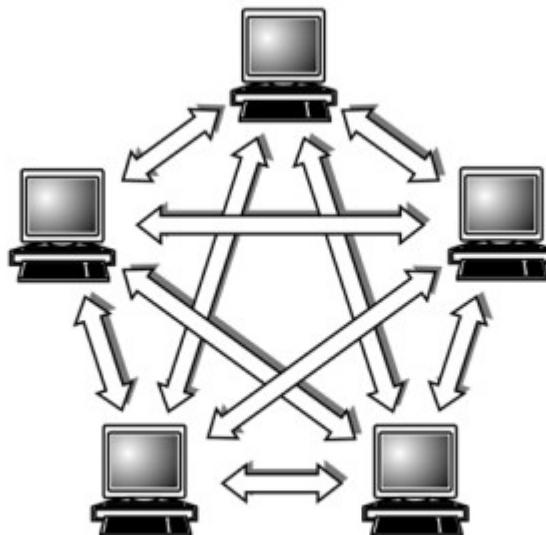


Figura 47 - Estilo Arquitetural - Peer-to-Peer

Fonte: <http://tekarts.com/wp-content/uploads/2012/02/peer-to-peer-network.gif>

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Map-reduce

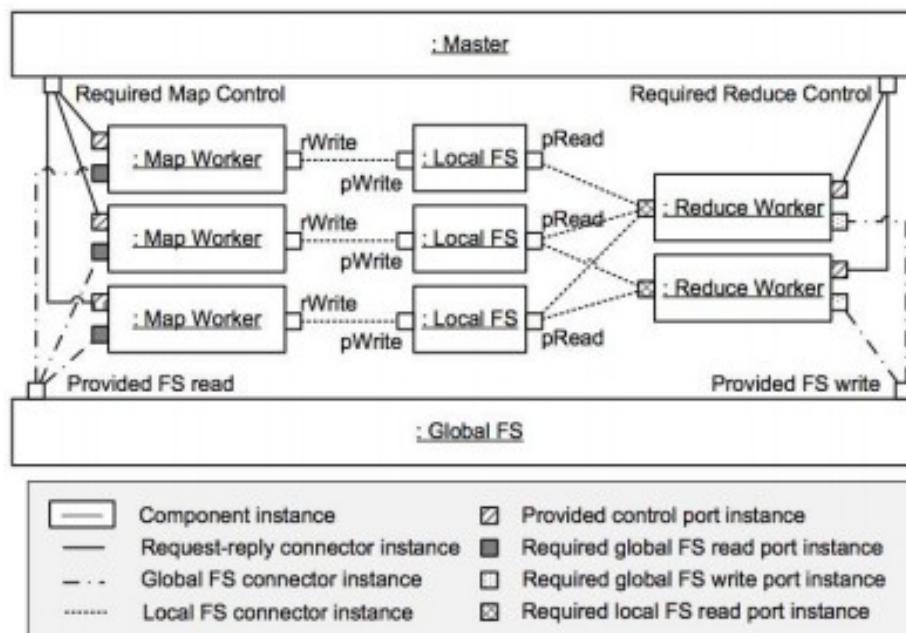
Grandes conjuntos de dados são processados de forma distribuída através de múltiplos computadores. Os resultados do processamento são combinados e agrupados formando a resposta

✓ **Vantagens**

- Escalabilidade
- Performance
- Disponibilidade
-

✓ **Desvantagens**

- Complexidade
- O custo pode ser alto

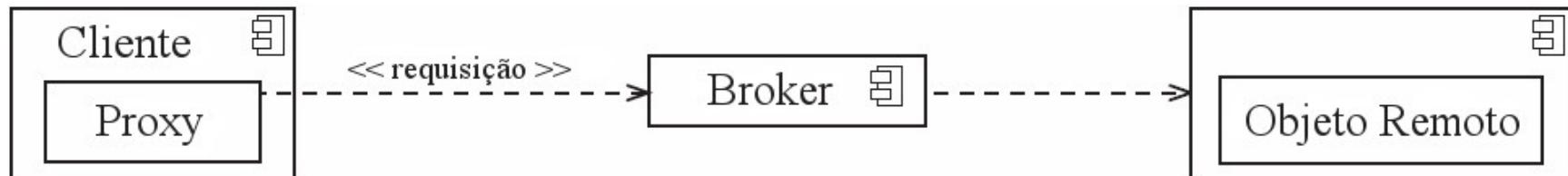


Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Padrão arquitetural *Broker*

- Distribuição transparente de aspectos da aplicação por diferentes nós
 - Um objeto pode chamar métodos de outros objetos sem saber que estes objetos estão remotamente distribuídos.
 - CORBA é um padrão aberto bem conhecido que permite que este tipo de arquitetura seja construída.

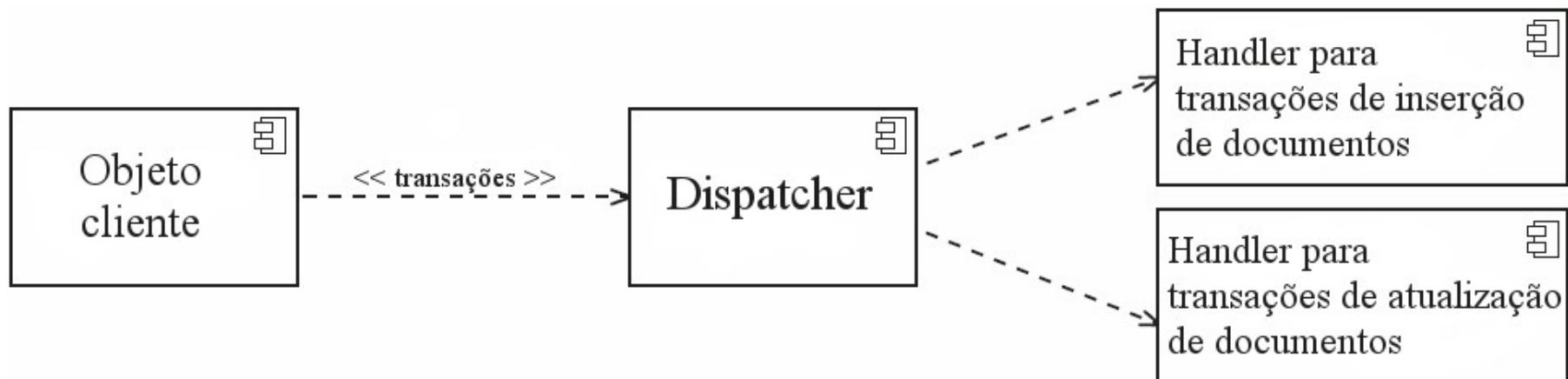


Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Padrão arquitetural Transaction-Processing

- Um processo lê uma série de entradas individualmente (uma a uma).
 - Cada entrada descreve uma transação - um comando que tipicamente realiza um conjunto de mudanças nos dados armazenados pela aplicação
 - Existe um componente, chamado *dispatcher*, responsável por definir o que deve ser feito com cada transação
 - Ele então dispara uma chamada a procedimento ou mensagem para o componente apropriado (*handler*), dentre os componentes (*handlers*) responsáveis por tratar as transações.



Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

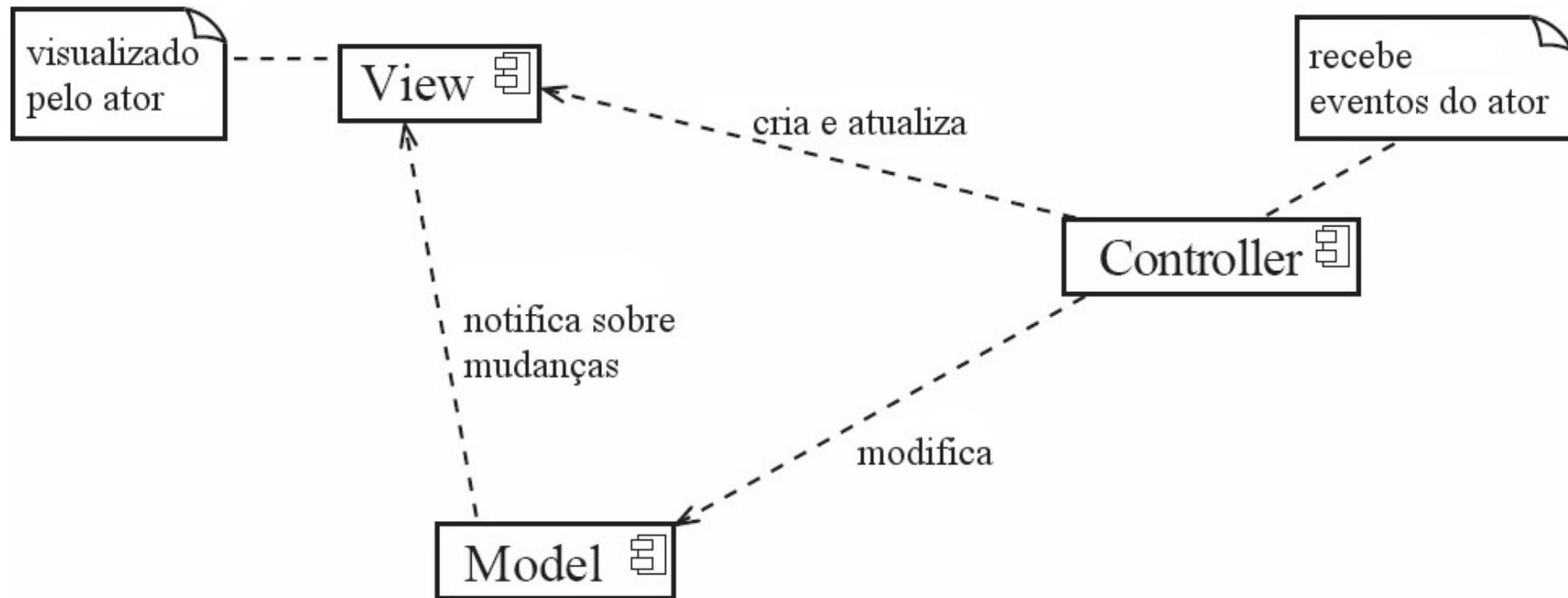
Padrão arquitetural Model-View-Controller (MVC)

- Padrão arquitetural utilizado para ajudar na separação da camada de interface com o usuário das demais partes da aplicação
 - O *model* contém as classes cujas instâncias devem ser visualizadas e manipuladas
 - O *view* contém os objetos utilizados para renderizar a aparências dos dados do modelo na interface do usuário
 - O *controller* contém os objetos que controlam e tratam as interações do usuário com o *view* e o *model*.
 - O padrão de projeto Observable é normalmente utilizado para separar o *model* do *view*.

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Padrão arquitetural Model-View-Controller (MVC)



Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

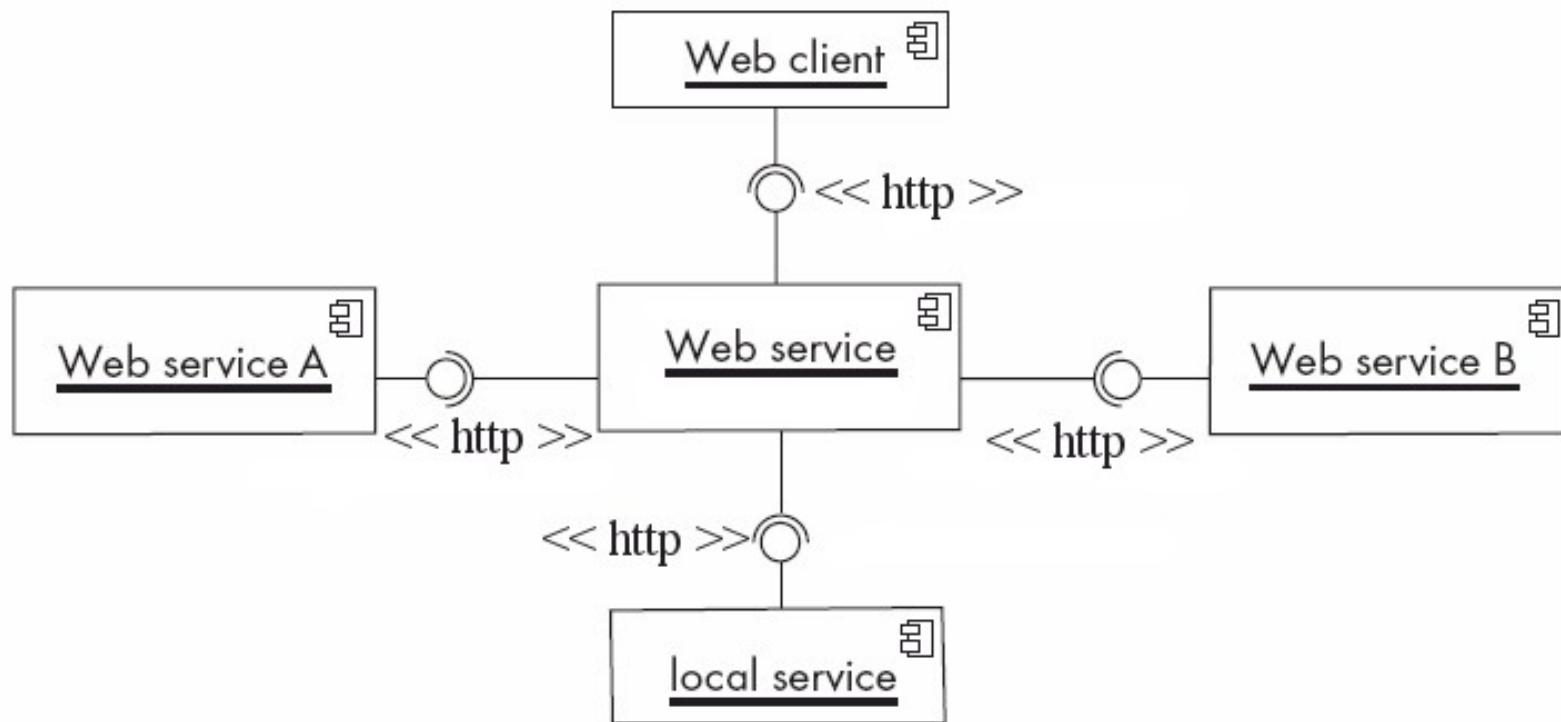
Padrão arquitetural orientado a serviço

- Este padrão organiza a aplicação como coleções de serviços que se comunicam utilizando interfaces bem definidas
 - No contexto de aplicações web, os serviços são chamados de Web services
 - Um web service é uma aplicação acessível através da internet e que pode ser integrada com outros serviços para formar uma aplicação maior
 - Os diferentes componentes geralmente se comunicam entre si utilizando padrões abertos, como o XML.

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Padrão arquitetural orientado a serviço



Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

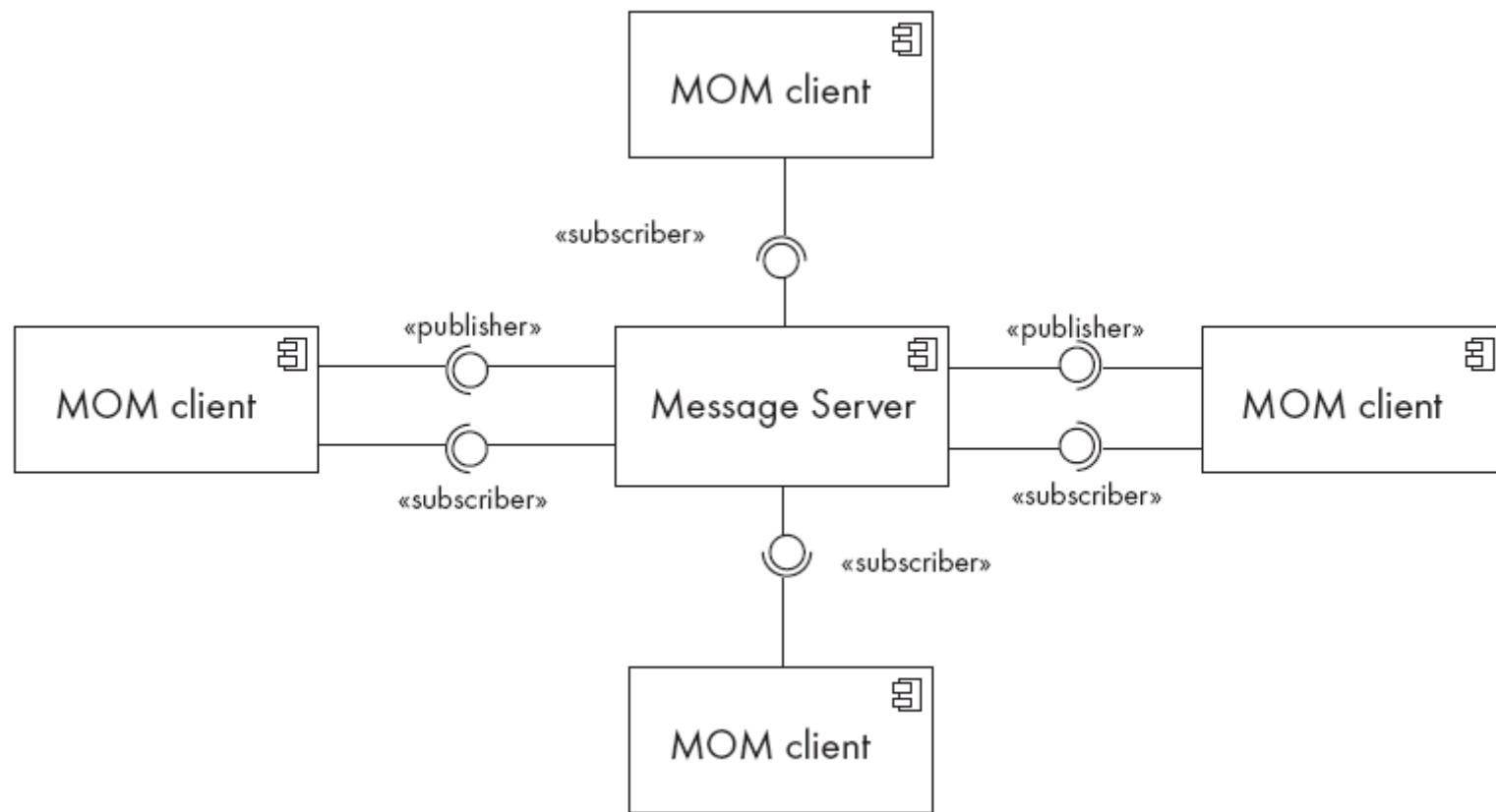
Padrão arquitetural orientado a mensagens

- Neste padrão diferentes subsistemas se comunicam e colaboram para realizar determinada tarefa apenas através de mensagens.
 - Também conhecido como *Message-oriented Middleware* (MOM)
 - O núcleo deste padrão é um sistema de mensagens entre aplicações (*application-to-application messaging system*)
 - Participantes da troca de mensagens precisam conhecer apenas os formatos das mensagens
 - Adicionalmente, as aplicações que estão se comunicando não precisam estar disponíveis ao mesmo tempo, pois as mensagens podem ser persistidas
 - As mensagens auto-contidas são enviadas de um componente (*publisher*) através de canais virtuais (*topics*) para os quais outros componentes de software podem assinar (*subscribers*)

Design e Arquitetura: Projeto Arquitetural

Estilos Arquiteturais

Padrão arquitetural orientado a mensagens





Escreva o documento
de design e arquitetura
do seu Projeto.

Apresentar na próxima aula...

Valendo !!!



(5^a -Insígnia da Invisibilidade)

http://www.cesarkallas.net/arquivos/faculdade/engenharia_de_software/13-Projeto%20de%20Software/Projeto%20de%20Software.pdf

<http://www.inf.ufpr.br/andrey/ci163/EstilosAI.pdf>