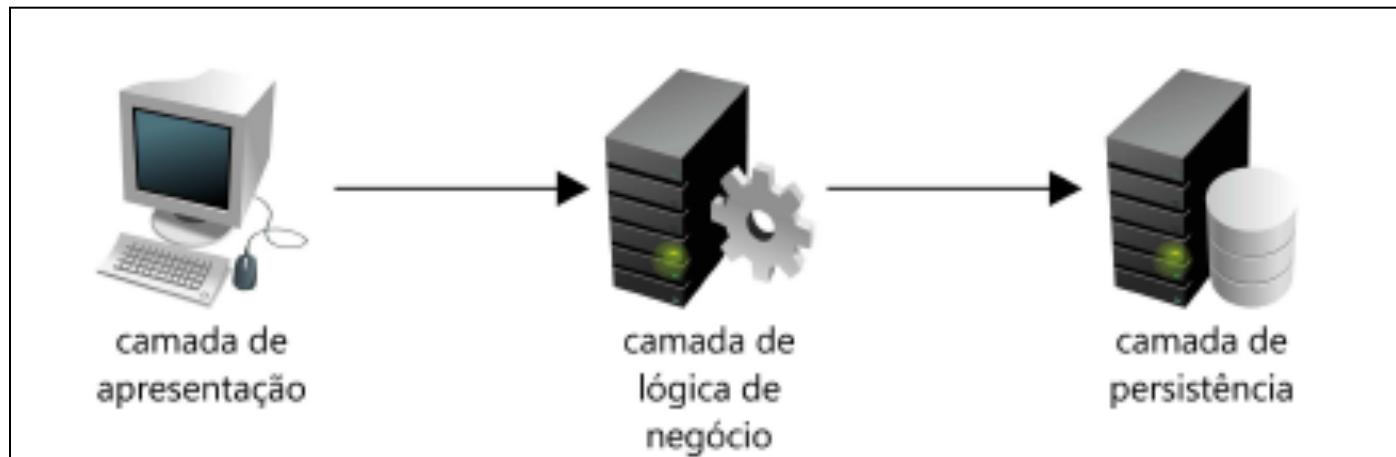


# Decompondo a definição de Arquitetura de Software



legenda:

→ comunicação entre camadas

# Decompondo a definição de Arquitetura de Software

- Exemplo
- A separação do sistema em três camadas pode também facilitar a **manutenção**.
- Se, além de adotar essa divisão, a camada de apresentação apenas se comunicar com a lógica de negócio, mas não com a de persistência, mudanças na camada de persistência afetarão apenas a camada de negócio.
- Portanto, caso seja necessário mudar o fornecedor da camada de persistência, a assinatura dos métodos disponíveis, ou mesmo o protocolo de comunicação, apenas a lógica de negócio será afetada por essas mudanças, uma vez que não existe acoplamento entre a apresentação e a persistência.

# Introdução

- Essa **estrutura e organização** é o que se chama de **arquitetura**.
- O atendimento aos atributos de qualidade do software se deve em grande parte à sua arquitetura.
- Através do estudo das características e técnicas de **projeto de arquitetura** que poderemos projetar e desenvolver melhores produtos de software.

# Arquitetura

- Atributos da Arquitetura de um edifício:
  - Forma geral da estrutura física ?



# Arquitetura

- Muito mais que isso:
  - É a maneira pela qual os vários componentes do edifício são integrados para formar um todo coeso
  - É o modo pelo qual o edifício se ajusta em seu ambiente e integra com outros edifícios da vizinhança
  - É o grau com que o edifício atende seu propósito expresso e satisfaz às necessidades de seu proprietário



# Arquitetura

- É o sentido estético da estrutura – o impacto visual do edifício: maneira pela qual as texturas, cores e materiais são combinados
- Engloba também os detalhes: projeto dos dispositivos de iluminação, o tipo de piso, o posicionamento dos painéis, etc...



- Arte



- Mais: milhares de decisões, tanto as grandes como as pequenas

# Definições

## Definição: Arquitetura de Software por Perry e Wolf

- **Perry e Wolf** introduziram sua definição para arquitetura de software em seu artigo seminal “*Foundations for the Study of Software Architecture*”.
- A definição que eles propõem consiste na fórmula abaixo e na explicação de seus termos:
- Fórmula:
  - **Arquitetura = {Elementos; Organização; Decisões}**
- De acordo com essa definição, a arquitetura de software é um conjunto de **elementos** arquiteturais que possuem alguma **organização**.
- Os elementos e sua organização são definidos por **decisões** tomadas para satisfazer objetivos e restrições.

# Definições

- São destacados **três tipos** de elementos arquiteturais:
  - **Elementos de processamento:** são elementos que usam ou transformam informação;
  - **Elementos de dados:** são elementos que contêm a informação a ser usada e transformada;
  - **Elementos de conexão:** são elementos que ligam elementos de qualquer tipo entre si.

# Definições

**Definição : Arquitetura de Software por Bass, Clements, Kazman**

- “A arquitetura de um programa ou de sistemas computacionais é a estrutura ou estruturas do sistema, a qual é composta de elementos de software, as propriedades externamente visíveis desses elementos, e os relacionamentos entre eles.”

# Definições

**Definição:** Arquitetura de Software pelo Padrão ISO/IEEE 1471-2000

- “Arquitetura é a organização fundamental de um sistema incorporada em seus componentes, seus relacionamentos com o ambiente, e os princípios que conduzem seu design e evolução.”

# **Decompondo a definição de Arquitetura de Software**

- **Decompondo a definição de Arquitetura de Software**
- A arquitetura de software é mais bem entendida através de suas partes.
- Considerando as definições expostas acima, podemos ressaltar seus dois principais aspectos, que serão os meios para alcançar os atributos de qualidade:
  - **elementos** e
  - **decisões arquiteturais**.

# Decompondo a definição de Arquitetura de Software

- **1 - Elementos arquiteturais**
- A arquitetura de um sistema deve definir os elementos que formarão o software.
- Tais elementos definem como o software é particionado em pedaços menores e, assim, definem como o software é entendido.
- Elementos arquiteturais são divididos em dois tipos:
  - **elementos estáticos**
  - **elementos dinâmicos.**

# Decompondo a definição de Arquitetura de Software

- **2 - Decisões arquiteturais**
- Uma arquitetura não deve ter suas estruturas definidas aleatoriamente, uma vez que são elas que permitem o sucesso relativo aos objetivos do sistema.
- É trabalho do arquiteto definir essas estruturas em meio às alternativas de design arquitetural existentes.
- O arquiteto deve decidir entre as alternativas, particionando o sistema em elementos e relações que possibilitarão o atendimento aos atributos de qualidade.
- Essas decisões são chamadas **decisões arquiteturais**.

# Decompondo a definição de Arquitetura de Software

Decisões arquiteturais devem:

## 1. Descrever suas características:

- Descrição
  - descrição do que foi decidido para o sistema, seja a descrição um elemento, módulo, classe, ou serviço que existirá da arquitetura, a descrição da comunicação de um elemento da arquitetura com outro, a descrição da agregação de diversos elementos diferentes da arquitetura para formar um serviço, ou a descrição de um princípio ou mais princípios que conduzirão a evolução do sistema.
- Objetivo
  - explicitar qual o objetivo de dada decisão, normalmente, permitindo ou restringindo um conjunto de atributos de qualidade do sistema
- Fundamentação
  - explicitar por que tal decisão foi tomada, seja por ser um padrão conhecido na indústria, seja por conhecimento prévio de como satisfazer os objetivos em questão, ou pela atual decisão ter mostrado os melhores resultados em meio a uma avaliação prévia das alternativas

# Decompondo a definição de Arquitetura de Software

2. Proporcionar Rastreabilidade de Requisitos
  - Facilita o entendimento e a manutenção do sistema representado pela arquitetura.
3. Preocupar com Evolução
  - Isso significa que uma decisão não necessariamente descreverá módulos, classes, ou serviços, mas também poderá descrever regras que deverão ser seguidas ao longo do desenvolvimento do sistema.

# Por que a arquitetura é importante?

- Há 3 razões-chave:
  - As representações da arquitetura de software são um facilitador para a comunicação entre todas as partes interessadas no desenvolvimento de um sistema computacional.
  - A arquitetura evidencia decisões de projeto iniciais que terão profundo impacto em todo o trabalho de engenharia de software que vem a seguir e, tão importante quanto, no sucesso final do sistema como uma entidade operacional.
  - A arquitetura “constitui um modelo relativamente pequeno e intelectualmente compreensível de como o sistema é estruturado e como seus componentes trabalham em conjunto”.
- O modelo de projeto de arquitetura e os padrões de arquitetura nele contidos são transferíveis.
  - Gêneros, estilos e padrões de arquitetura podem ser aplicados ao projeto de outros sistemas e representam um conjunto de abstrações que permitem aos engenheiros de software descrever arquitetura de modo previsível.

# Descrições de arquitetura

- Diferentes interessados (steakholders) verão uma arquitetura sob diversos pontos de vista, orientados por conjuntos de interesses distintos.
- Um descrição de arquitetura é um conjunto de artefatos que refletem diferentes visões do sistema.
- **Exemplo:** um arquiteto de um *conjunto comercial* tem solicitações do:
  - Steakholder proprietário: esteticamente agradável e que forneça espaço e infraestrutura suficientes para garantir lucratividade.
    - Desenhos tridimensionais do edifício (para ilustrar a parte estética) e um conjunto de plantas bidimensionais para visão de espaço e infraestrutura do conjunto comercial.
  - Steakholder fabricante de aço para a estrutura do prédio: informações detalhadas sobre o aço de construção que irá sustentar o edifício, incluindo tipos de vigas duplo T, suas dimensões, conectividade, materiais e outros detalhes.
    - Desenhos especializados da estrutura de aço de construção

# Descrições de arquitetura

- A descrição de arquitetura de um sistema baseado em software deve apresentar características análogas àquelas citadas para o conjunto comercial.
  - Os desenvolvedores desejam **orientação clara e determinada** sobre como prosseguir com um projeto.
  - Os clientes querem um entendimento claro sobre as **mudanças** que devem ocorrer no ambiente e garantias de que a arquitetura atenderá suas necessidades de negócio.

Cada um desses “desejos” se reflete em uma visão diferente representada sob um ponto de vista diferente.

# Descrições de arquitetura

- A IEEE Computer Society propôs o **IEEE-Std-1471-2000**, (Recomended Practice for Architectural Description os Software-Intensive Systems, com os seguintes objetivos:
  1. Estabelecer uma definição conceitual e um vocabulário para uso durante o projeto da arquitetura de software
  2. Fornecer diretrizes detalhadas para representar uma descrição de arquitetura
  3. Encorajar práticas de projeto de arquitetura consistentes
- Definição de ‘**descrição da arquitetura**’ (architectural description, AD):
  - “**Conjunto de produtos para documentar uma arquitetura**”
- Uma descrição é representada usando-se várias visões: cada visão é “a representação de um sistema como um todo segundo a perspectiva de um conjunto de necessidades (dos interessados) relacionado”.

# Decisões de arquitetura

- A visão é criada de acordo com regras e convenções definidas em um ponto de vista.
- Cada visão desenvolvida como parte da descrição arquitetural trata de uma necessidade específica do interessado.
- Para desenvolver cada visão, o arquiteto considera uma variedade de alternativas e decide sobre as características de uma arquitetura específica que melhor atendam à necessidade.
- Pode ser usado um template (sugerido) para documentar cada decisão importante de arquitetura, estabelecendo um registro histórico que pode ser útil quando mudanças forem necessárias.

# Gêneros de arquitetura

- Os **princípios fundamentais** do projeto da arquitetura se aplicam a todos os tipos de arquitetura.
- O **gênero** normalmente ditará a abordagem de arquitetura específica para a estrutura que deve ser construída.
- Implica uma **categoria específica** no domínio de software geral.
- Em cada categoria, pode-se encontrar uma série de subcategorias.
- Exemplo: estilos gerais dentro do **gênero edifícios**:
  - casas
  - condomínios
  - prédios de apartamentos,
  - conjuntos comerciais,
  - prédios industriais,
  - armazéns
- Em cada estilo geral, poderiam ser aplicados estilos mais específicos.
- Cada estilo teria uma estrutura que poderia ser descrita usando-se um conjunto de padrões previsíveis.

# Gêneros de arquitetura

- Gêneros de arquitetura para sistemas baseados em software, segundo Grady Booch:
  - **Inteligência artificial** — Sistemas que simulam ou ampliam a cognição, locomoção ou outros processos orgânicos humanos.
  - **Comercial e sem fins lucrativos** — Sistemas que são fundamentais para a operação de um empreendimento comercial.
  - **Comunicações** — Sistemas que fornecem a infraestrutura para a transferência e o gerenciamento de dados, para conectar usuários desses dados ou para apresentar dados na periferia de uma infraestrutura.
  - **Autoria de conteúdo** — Sistemas que são utilizados para criar ou manipular artefatos de texto ou multimídia.
  - **Dispositivos** — Sistemas que interagem com o mundo físico para fornecer algum serviço para um indivíduo.
  - **Esportes e entretenimento** — Sistemas que gerenciam eventos públicos ou que geram uma experiência de entretenimento a um grande grupo.
  - **Financeiros** — Sistemas que fornecem a infraestrutura para transferir e administrar dinheiro e outros valores.

# Gêneros de arquitetura

- Cont.

- **Jogos** — Sistemas que geram uma experiência de entretenimento para indivíduos ou grupos.
- **Governo** — Sistemas que dão apoio à conduta e operações de uma entidade municipal, estadual, federal, internacional ou outras entidades políticas.
- **Industriais** — Sistemas que simulam ou controlam processos físicos.
- **Legais** — Sistemas que dão apoio ao setor judiciário.
- **Médicos** — Sistemas que diagnosticam ou curam ou contribuem para a pesquisa médica.
- **Militar** — Sistemas para consultas, comunicações, comando, controle e inteligência (C4I), bem como armamento de ataque e defesa.
- **Sistemas operacionais** — Sistemas que se situam logo acima do hardware para fornecer serviços básicos de software.
- **Plataformas** — Sistemas que se posicionam logo acima dos sistemas operacionais para fornecer serviços avançados.
- **Científicos** — Sistemas que são utilizados para pesquisa e aplicações científicas.
- **Ferramentas** — Sistemas que são utilizados para desenvolver outros sistemas.
- **Transportes** — Sistemas que controlam veículos de navegação, terrestres, aéreos ou espaciais.
- **Serviços públicos** — Sistemas que interagem com outros softwares para fornecer algum serviço.

# Gêneros de arquitetura

- Sob o ponto de vista do projeto da arquitetura, cada gênero representa um desafio único.
- **Exemplo:** arquitetura de software para um sistema de jogos eletrônicos:
  - Cálculo de algoritmos intensivos
  - Computação gráfica sofisticada
  - Fontes de dados multimídia streaming
  - Interatividade em tempo real via entradas convencionais e não convencionais
  - Outras necessidades especializadas

# Estilos de arquitetura

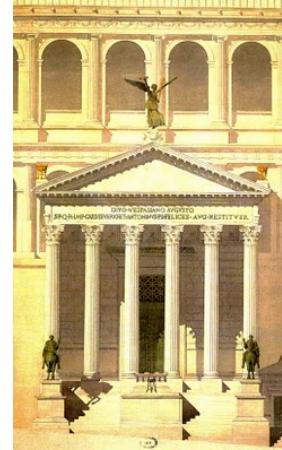
- Descrição de uma casa:

Casa “Colonial americano com hall central”

Pessoas dos EUA conseguem evocar uma imagem geral de como se parecerá a casa



- Uso de um estilo arquitetônico como mecanismo descritivo para diferenciar a casa de outros estilos (minimalista, brutalista, cape cod, etc)



# Estilos de arquitetura

- O estilo é também um **template** para construção: devem ser definidos mais detalhes:
  - suas dimensões finais devem ser especificadas,
  - características personalizadas acrescentadas,
  - materiais de construção definidos,
  - mas o estilo orienta o construtor em seu trabalho
- O **software** que é criado para sistemas computacionais também apresenta um de muitos estilos de arquitetura.

# Estilos de arquitetura

- O software que é criado para sistemas computacionais também apresenta **um de muitos estilos de arquitetura**.
- Cada estilo descreve uma categoria que engloba:
  1. Um conjunto de componentes (ex: banco de dados, módulos computacionais) que realiza uma função específica por um sistema
  2. Um conjunto de conectores que habilitam a “comunicação, coordenação e cooperação” entre os componentes
  3. Restrições que definem como os componentes podem ser integrados para formar o sistema
  4. Modelos semânticos que permitem a um projetista compreender as propriedades gerais de um sistema por meio da análise das propriedades conhecidas de suas partes constituintes

# Estilos de arquitetura

- A arquitetura de software representa uma estrutura com entidades (componentes) interligados por relacionamento definidos (conectores)
- Tipos de estruturas (componentes, conectores e propriedades) podem ser usados para descrever uma arquitetura.
- Há 5 estruturas de arquitetura fundamentais (ou canônicas):

**Estrutura funcional.** Os componentes representam entidades funcionais ou de processamento. Os conectores representam interfaces que fornecem a capacidade de "usar" ou "passar dados para" um componente. As propriedades descrevem a natureza dos componentes e a organização das interfaces.

**Estrutura física.** Essa estrutura é similar ao modelo de implantação desenvolvido como parte do projeto. Os componentes são o hardware físico onde reside o software. Os conectores são as interfaces entre os componentes de hardware e as propriedades tratam a capacidade, largura de banda, desempenho e outros atributos.

# Estilos de arquitetura

- cont.

**Estrutura de concorrência.** Os componentes representam “unidades de concorrência” organizadas como tarefas paralelas ou *threads*. “As relações [conectores] incluem sincronizações-com, é-de maior prioridade-que, envio-dados-a, não é possível-executar-sem e não é possível-executar-com. Entre as propriedades relevantes a essa estrutura temos prioridade, preempção e tempo de execução” [Bas03].

**Estrutura de desenvolvimento.** Essa estrutura define os componentes, artefatos e outras fontes de informação necessárias à medida que a engenharia de software prossegue. Os conectores representam os relacionamentos entre os artefatos, e as propriedades identificam as características de cada item.

# Estilos de arquitetura

- cont.

**Estrutura de implementação.** "Os componentes podem ser pacotes, classes, objetos, procedimentos, funções, métodos etc., todos os quais são veículos para empacotar funcionalidade em vários níveis de abstração" [Bas03]. Os conectores incluem a habilidade de passar dados e controle, compartilhar dados, "uso" e "instância-de". As propriedades concentram-se nas características de qualidade (por exemplo, facilidade de manutenção, reusabilidade) resultantes quando é implementada uma estrutura.

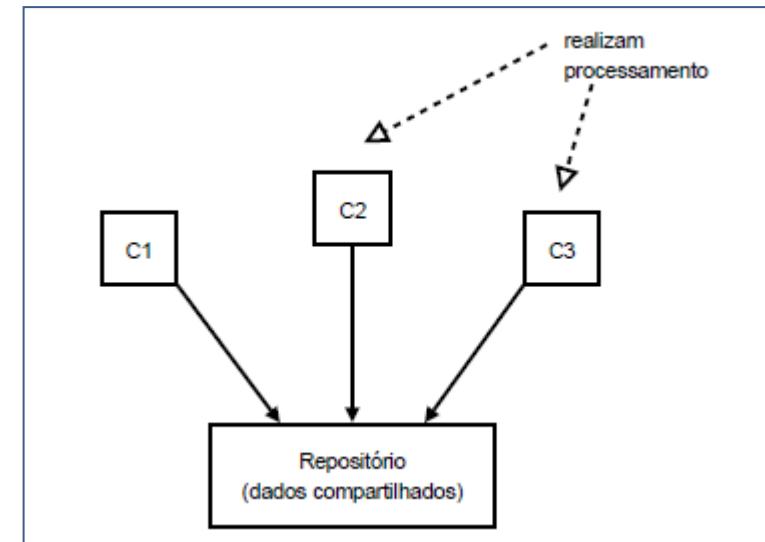
# Taxonomia dos estilos de arquitetura 1

## 1 . Arquiteturas centralizadas em dados

- Um repositório de dados (por exemplo, um arquivo ou banco de dados) reside no centro dessa arquitetura e é em geral acessado por outros componentes que atualizam, acrescentam, eliminam ou de alguma outra maneira modificam dados contidos no repositório.
- O software-cliente acessa um repositório central.
- Em alguns casos o repositório de dados é passivo, ou seja, o software-cliente acessa os dados independentemente de quaisquer alterações nos dados ou das ações de outros softwares-clientes.
- Uma variação dessa abordagem transforma o repositório em um “quadro-negro” que envia notificações ao software-cliente quando os dados de seu interesse mudam.

# Taxonomia dos estilos de arquitetura 1

- **A) Estilo Repositório** compõe-se de:
  - (1) o repositório propriamente dito, uma estrutura que armazena e centraliza os dados do sistema.
  - (2) um conjunto de componentes independentes entre si que operam sobre os dados do repositório.

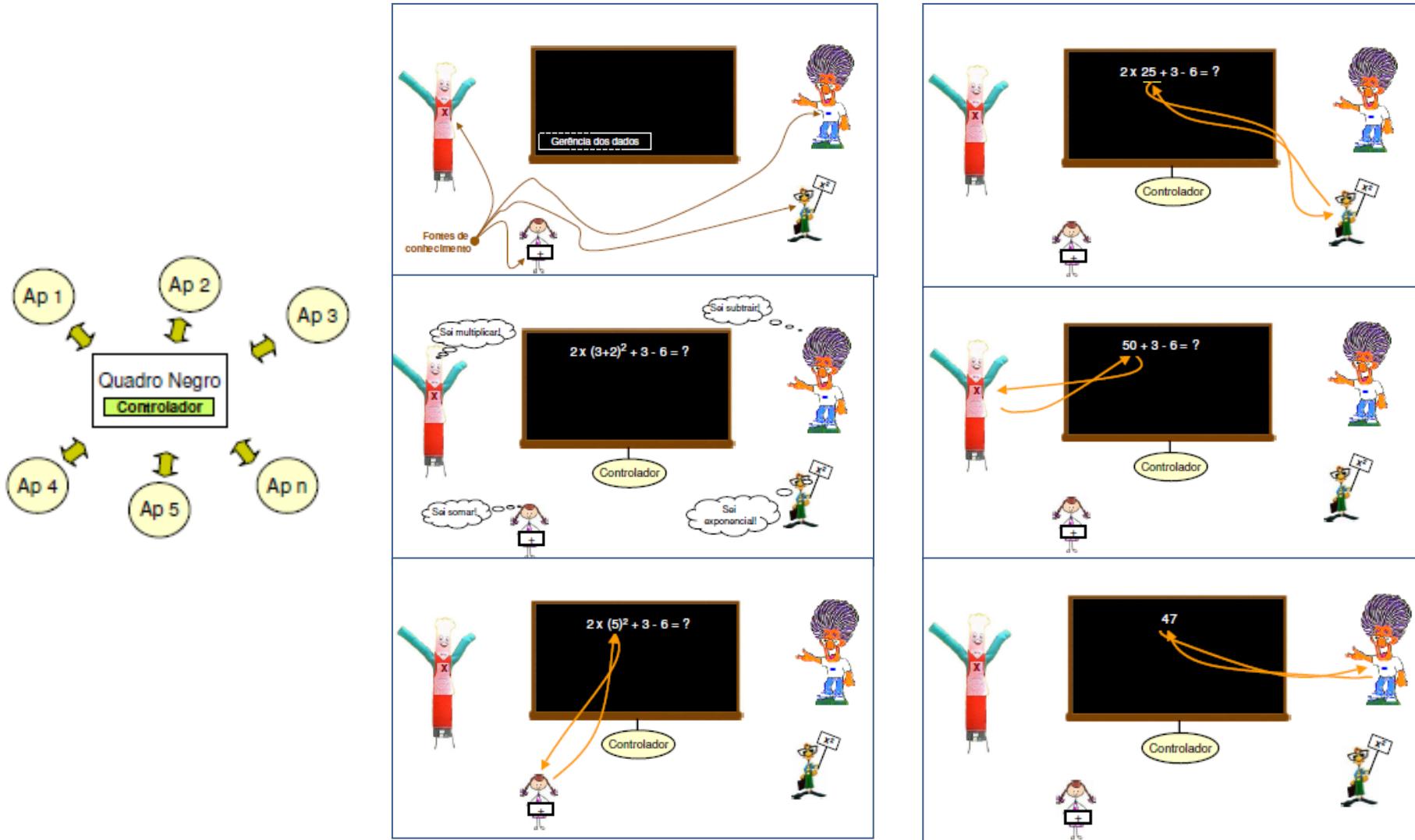


- **Exemplos:**
  - **ferramentas CAD** (“Computer Aided Design” ou Projeto Apoiado por Computador)
  - **ambientes integrados de desenvolvimento de software**
    - onde diversas ferramentas de modelagem, visualização, apoio à edição e outros operam de modo diferente sobre um mesmo conjunto de dados

# Taxonomia dos estilos de arquitetura 1

- B) **Estilo Quadro Negro** (Blackboard)
- Ele possui os mesmos dois tipos de componentes:
  - o repositório, chamado de blackboard, que armazena dados
  - componentes da base de conhecimento: subsistemas independentes, cada qual resolvendo aspectos específicos do problema
  - componente de controle: monitora mudanças no blackboard e decide as ações
- Nesse estilo, o estado do blackboard dispara a escolha de qual componente será executado.
- Assim, cada componente contribui incrementalmente para a solução do problema que o sistema objetiva resolver.
- Os componentes comunicam-se exclusivamente através do repositório.
- **Exemplo:**
- Utilizado em aplicações que exigem interpretação de elementos complexos, tais como processamento de sinais, reconhecimento de padrões e de linguagem natural.

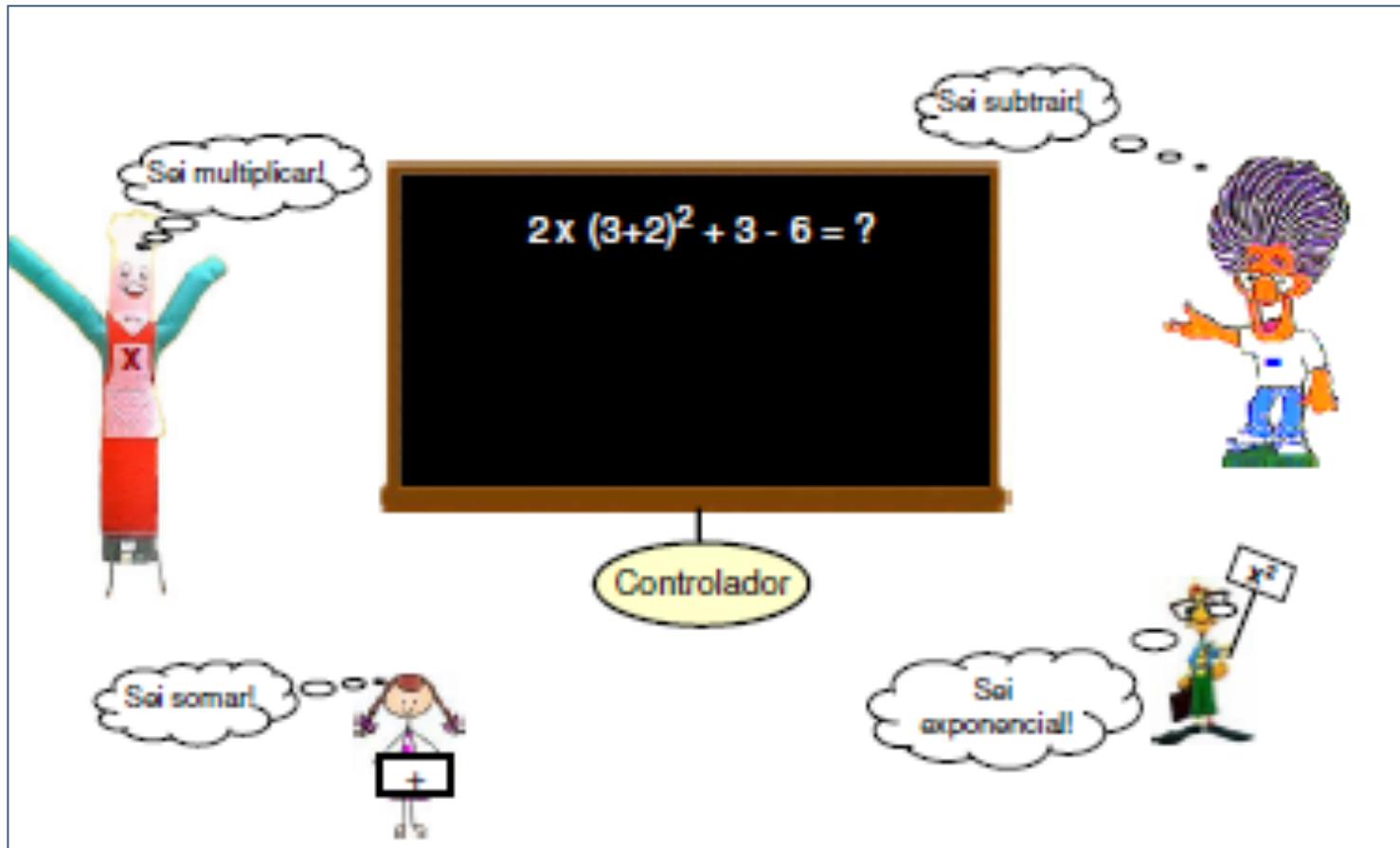
# Taxonomia dos estilos de arquitetura 1



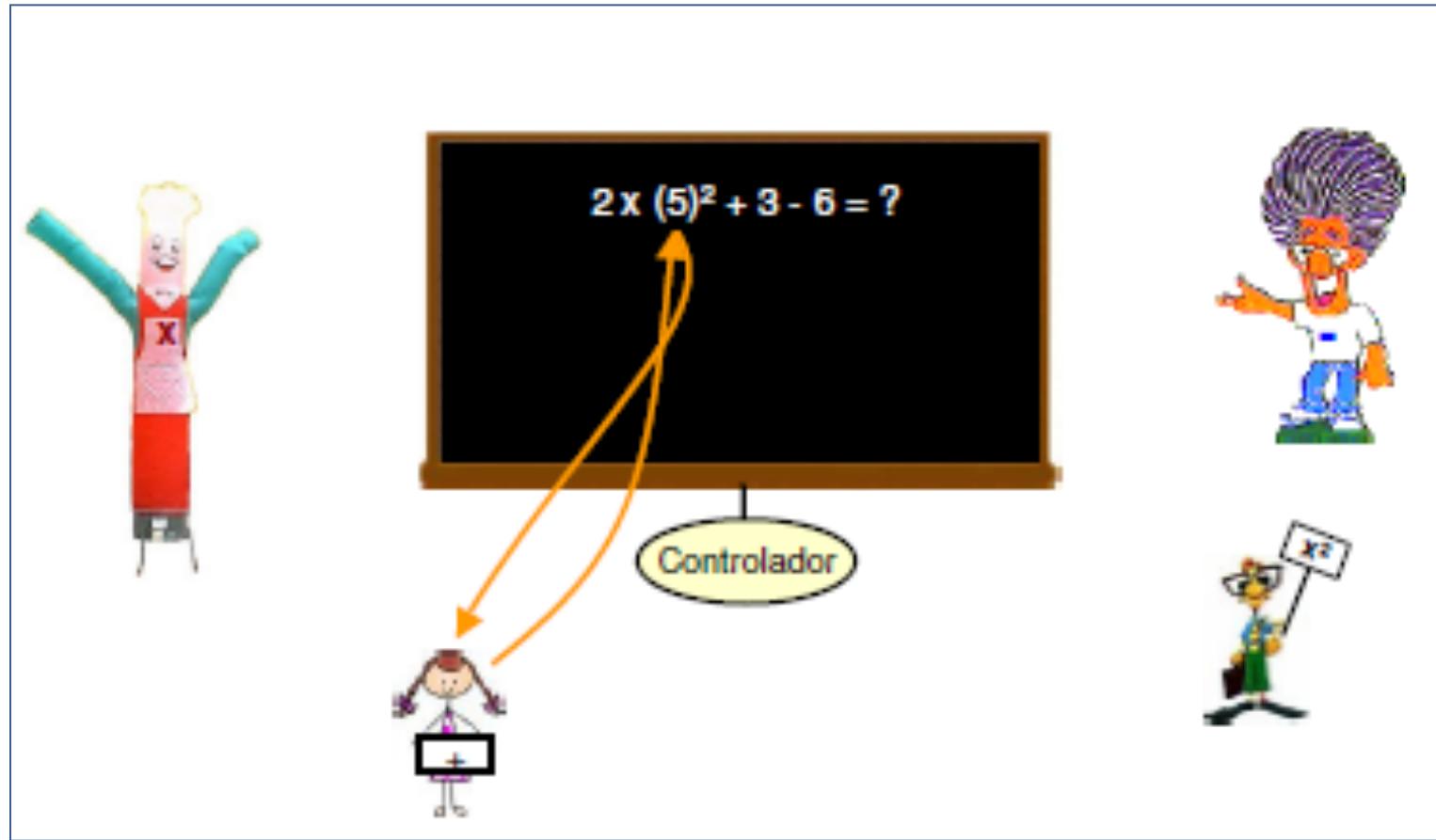
# Taxonomia dos estilos de arquitetura 1



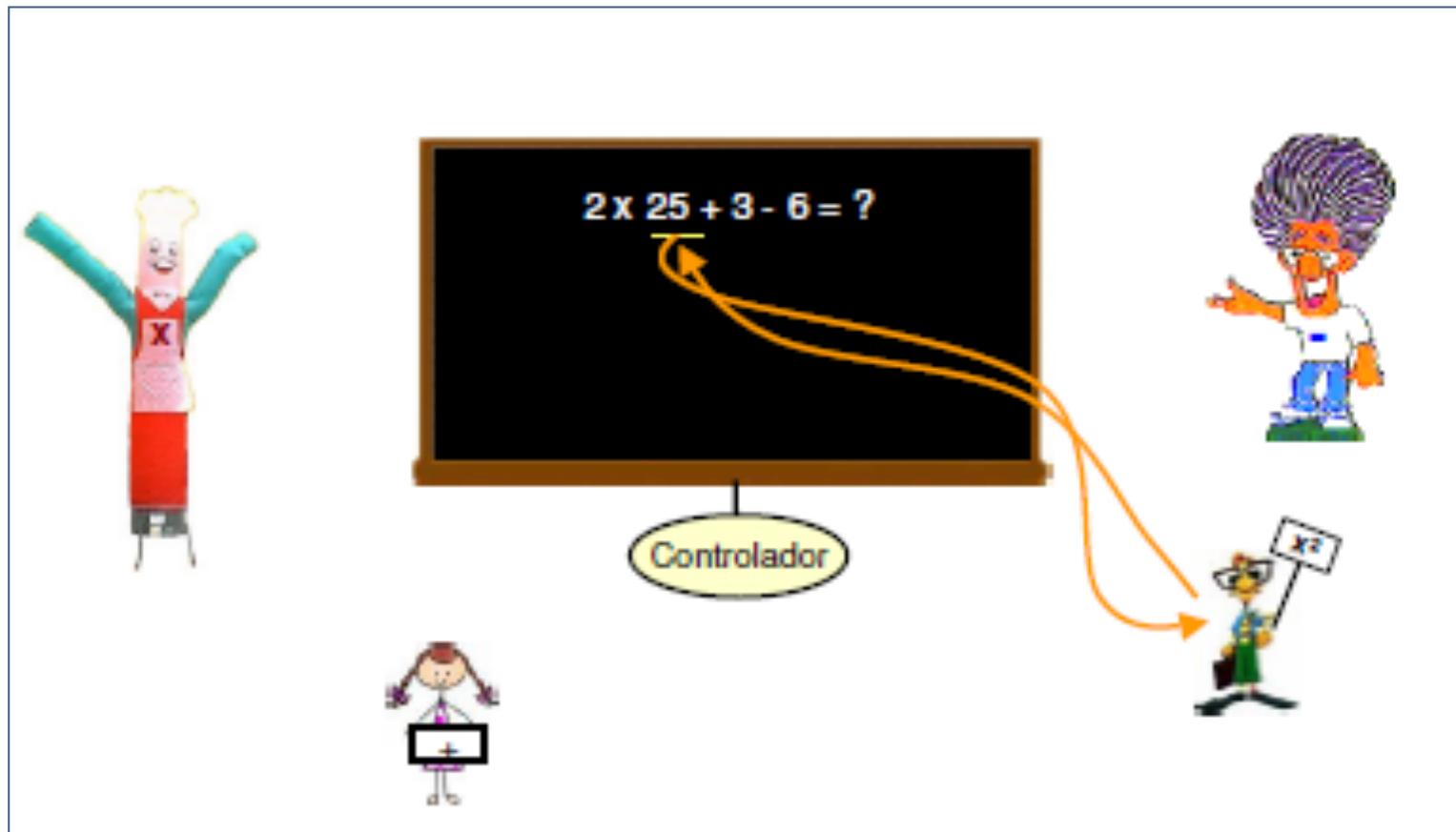
# Taxonomia dos estilos de arquitetura 1



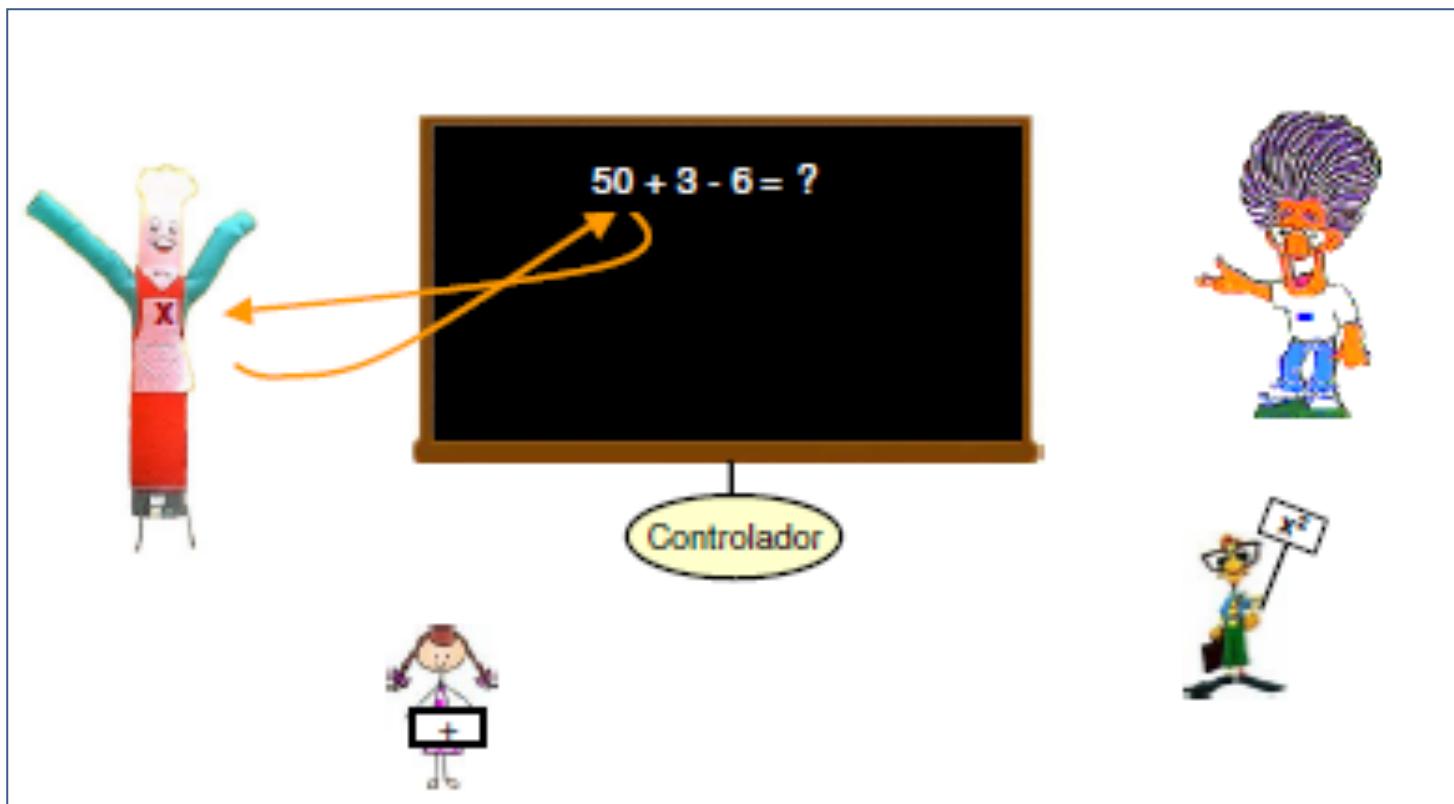
# Taxonomia dos estilos de arquitetura 1



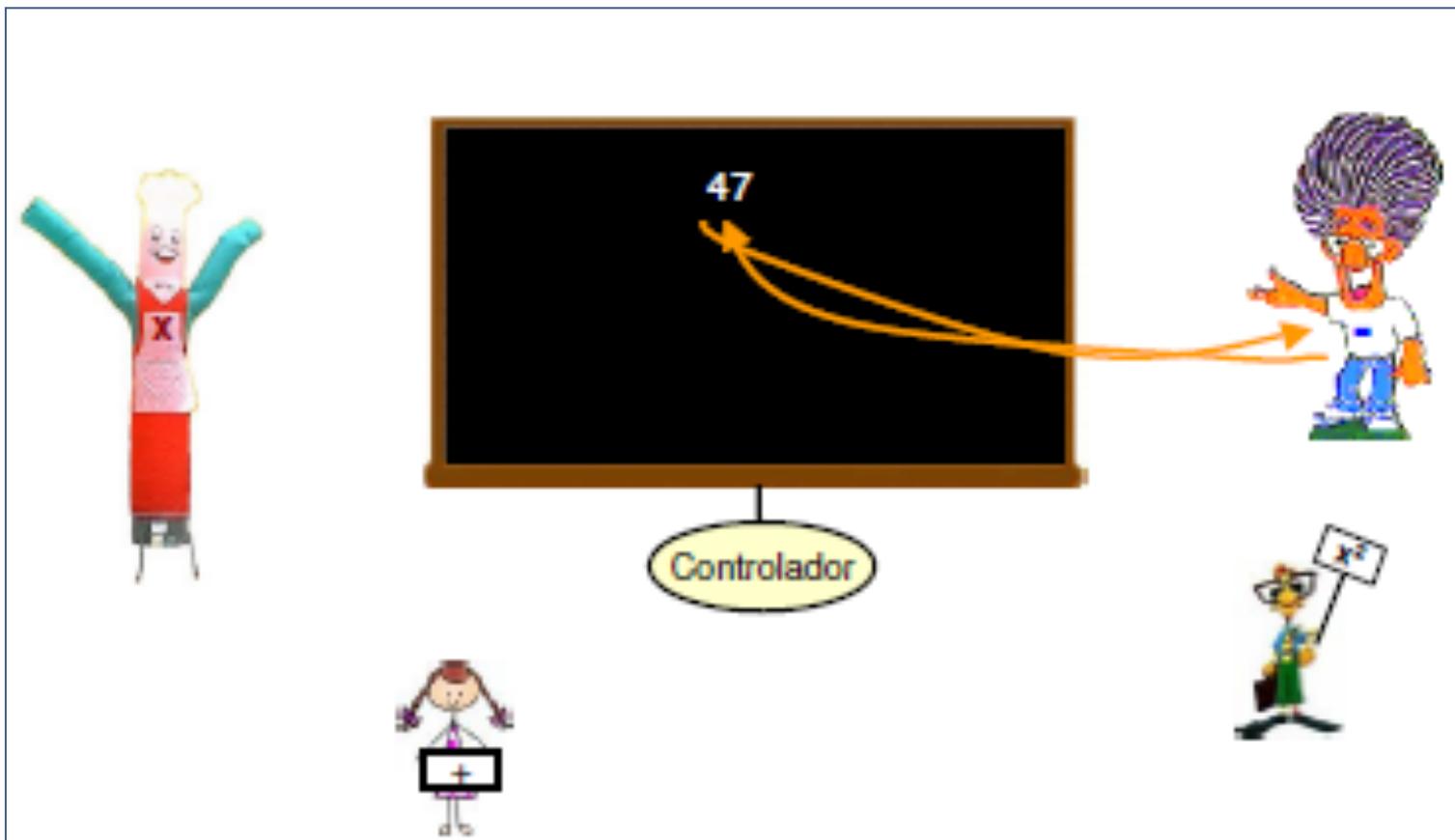
# Taxonomia dos estilos de arquitetura 1



# Taxonomia dos estilos de arquitetura 1

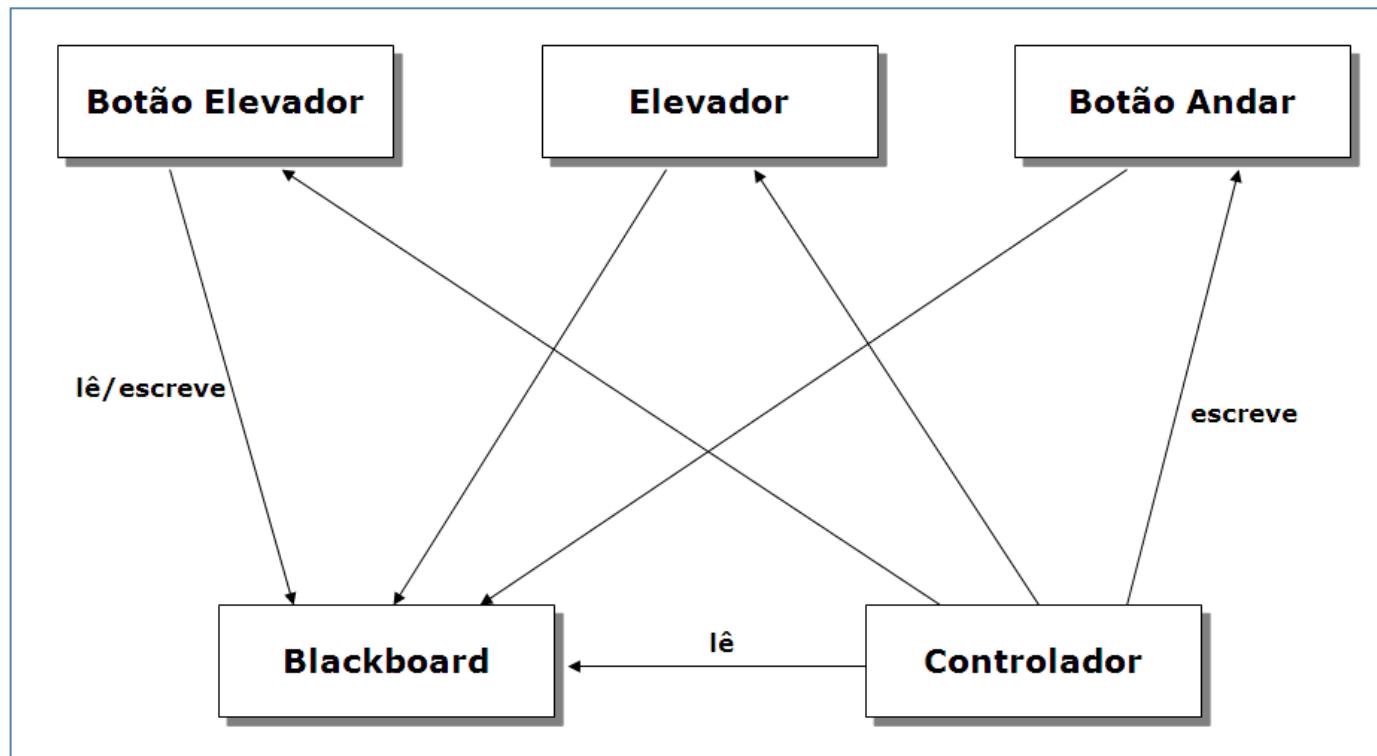


# Taxonomia dos estilos de arquitetura 1



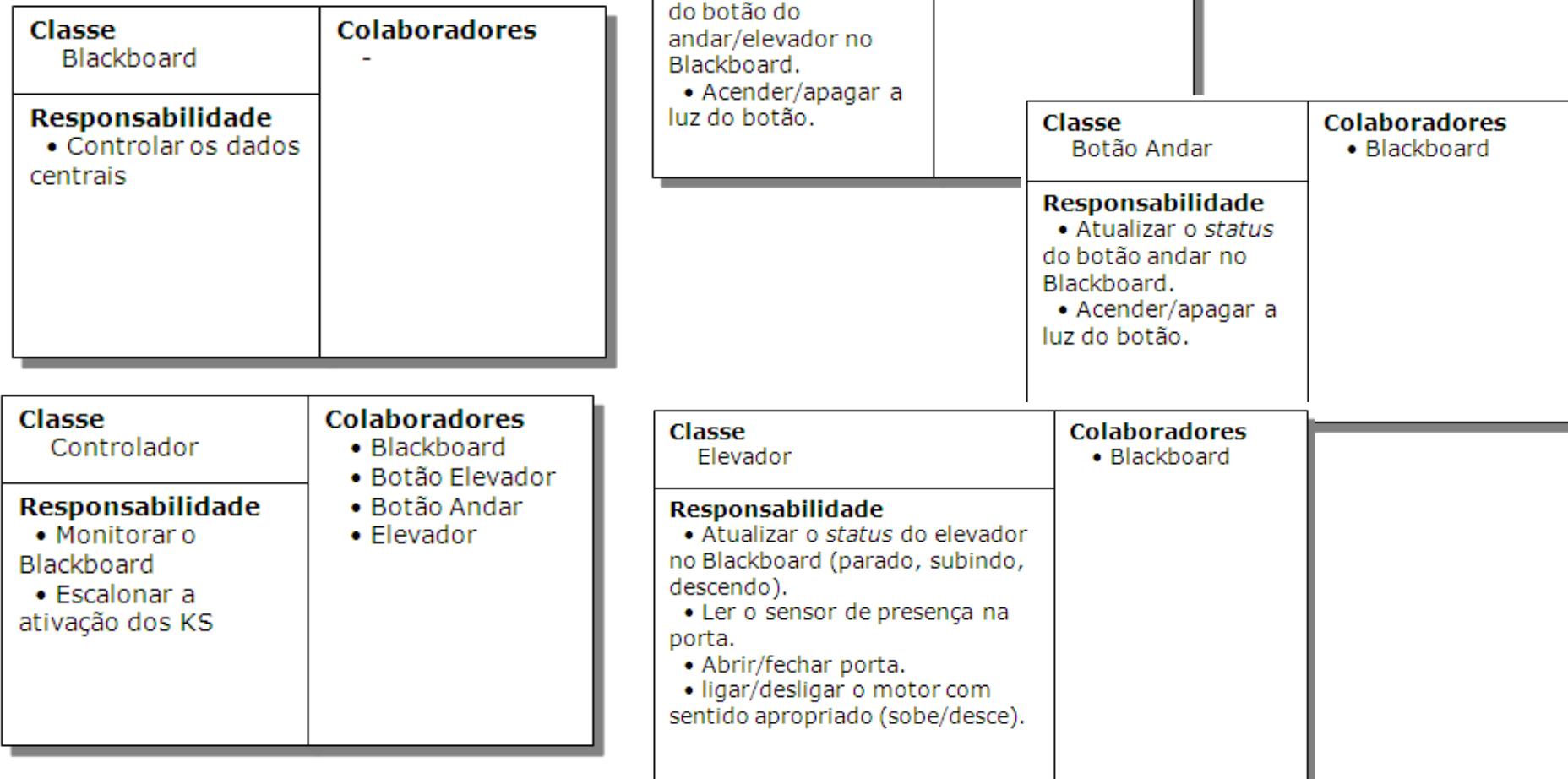
# Exercício

- Resolução:



# Exercício

- Classes

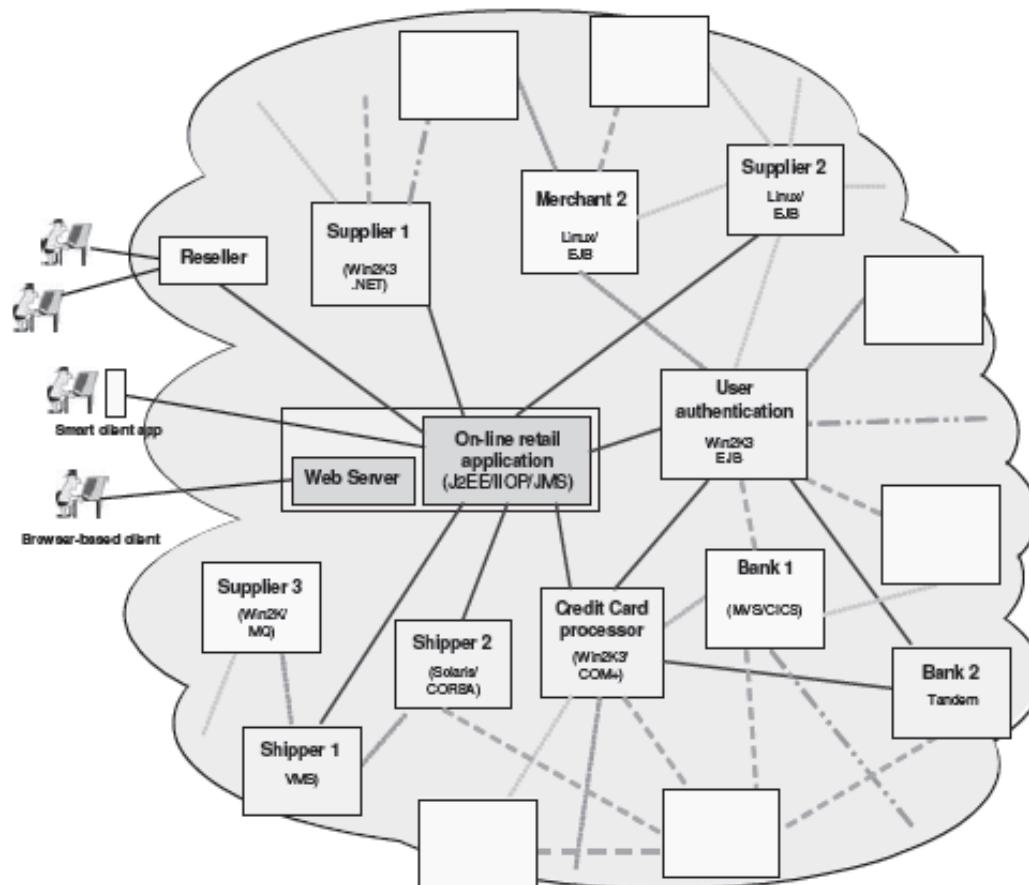


# Arquitetura orientada a serviços (SOA)

- **Arquitetura orientada a serviços (SOA)**
- Os Web services são realmente apenas uma outra tecnologia de integração de aplicativos, conceitualmente pouco diferente para CORBA, J2EE, DCOM ou qualquer um dos seus concorrentes.
- Todas essas tecnologias são muito semelhantes: **os aplicativos cliente podem descobrir servidores, descobrir quais serviços que eles estão oferecendo e invocar o funções que eles fornecem.**
- O que é diferente sobre sistemas orientados a serviços e suas tecnologias de apoio a Web Services é que estas aplicações e os servidores são esperados agora para, potencialmente, serem controladas e **executadas por fora das organizações e acessados através da Internet pública.**
- O resultado desta mudança de foco é um conjunto de normas e **princípios arquitetônicos que enfatizam interoperabilidade**, fazendo o menor número de suposições possíveis sobre como prestadores de serviços e consumidores trabalham internamente e que detalhes de implementação eles têm em comum

# Arquitetura orientada a serviços (SOA)

- Ex



Example service-based retail application

# Arquitetura orientada a serviços (SOA)

- **Conceito**

O “S” do SOA - **Serviço**

- É uma tarefa repetitiva de negócios – Ex. Verificar crédito cliente; abrir nova conta
- É um componente, **altamente coeso e fracamente acoplado que** encapsula uma função de negócio reutilizável
- Recebe requisições e responde encapsulando todo o detalhe do seu Processamento.
- Executa um **ciclo completo de trabalho e não depende do estado (stateless)** de outros componentes externos.
- É uma unidade de trabalho feita por um fornecedor de serviço para fornecer resultados finais requeridos por um consumidor de serviço
- É invocado através de protocolos de comunicação independentes da localização e da tecnologia de suporte

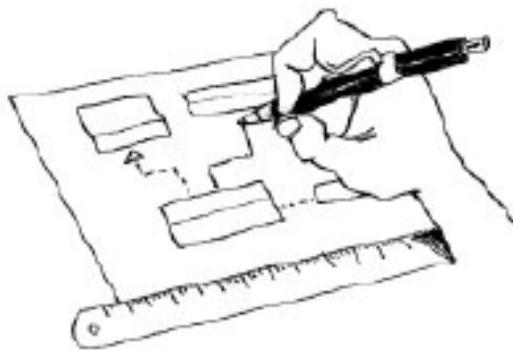
# Arquitetura orientada a serviços (SOA)

- O “O” do SOA – Orientado a serviços

CEO	Para o CEO é uma forma crucial de criar uma corporação conectada e responder melhor às demandas de clientes e pressões de mercado.
CIO	Para o CIO é uma possibilidade de proteger investimentos existentes de TI sem inibir o desenvolvimento de novas capacidades. É a forma de utilizar TI como alavancador da empresa ao invés de barreiras.
Analista de Negócio	Para os analistas de negócios é uma forma de trazer investimentos e ativos alinhados com a estratégia e processos de negócios de Negócio.
Executivos de TI	Para os Executivos e Gerentes de TI é uma forma de efetivamente integrar sistemas heterogêneos. Possibilitando uma melhor gestão da complexidade de TI e responder eficientemente às necessidades de negócio.
Desenvolvedores de TI	Aos desenvolvedores é o caminho para se criar aplicações dinâmicas e colaborativas e melhorar a reutilização de TI.

# Arquitetura orientada a serviços (SOA)

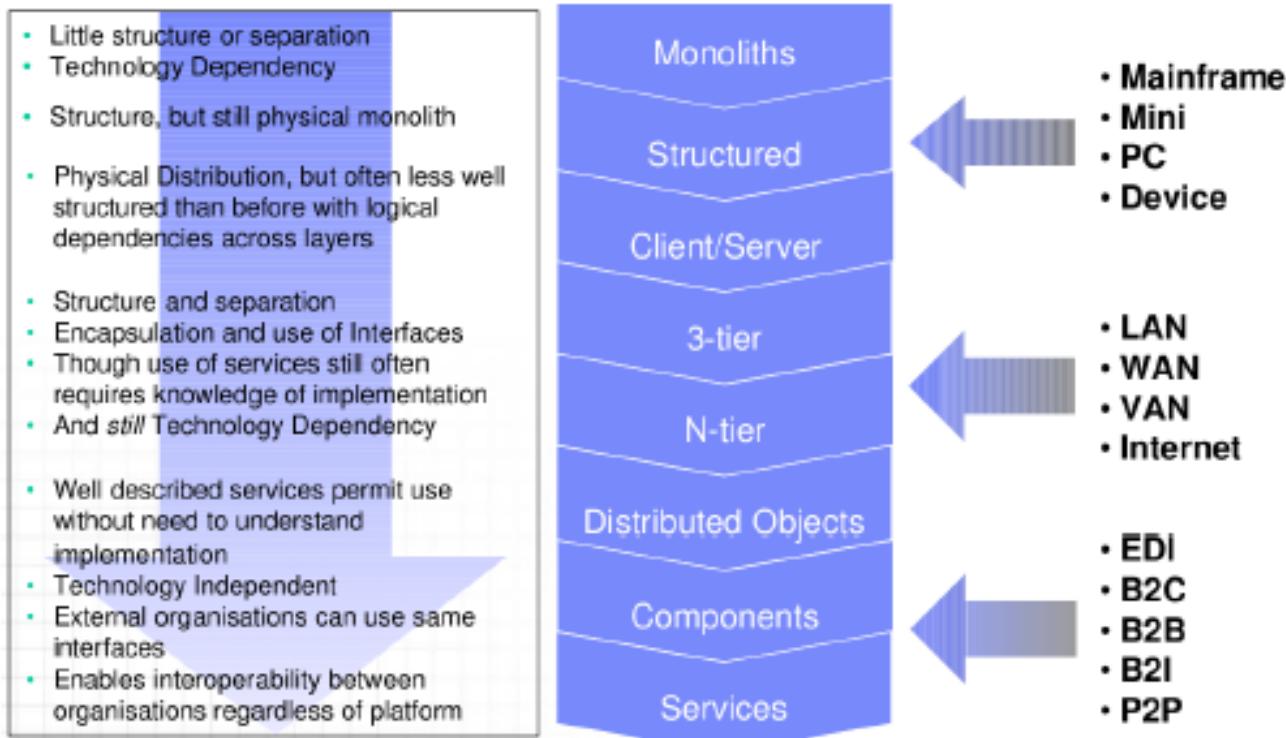
- O “A” do SOA - **Arquitetura**
- É a estrutura do sistema composta pelos elementos de software, propriedades visíveis destes elementos e o relacionamento entre eles.
- É um estilo de arquitetura que suporta a integração dos negócios com serviços conectados.



# Arquitetura orientada a serviços (SOA)

- Evolução das arquiteturas

## Evolução das Arquiteturas



# Arquitetura orientada a serviços (SOA)

- Tecnologias ligadas ao SOA

## **POO (Programação Orientada a Objetos)**

- POO é um paradigma de desenvolvimento de softwares (Objetos, Classes, Métodos, herança, polimorfismo,etc...)

## **WOA (Web Oriented Architeture)**

- Os softwares SOA utilizados na internet.

No WOA os artefatos são conhecidos como recursos, que são quaisquer artefatos que possam ser identificados por uma **URI (Universal Resource Identifier)**, basicamente o endereço do recurso.

## **Web Services**

- Os serviços encontrados para WEB são conhecidos como WEB Services
- são componentes que permitem às aplicações enviar e receber dados em formato XML
- são padronizados segundo **UDDI (Universal Description, Discovery and Integration)**

# Arquitetura orientada a serviços (SOA)

- Tecnologias

## **UDDI (Universal Description, Discovery and Integration)**

- Especificação que define um **serviço de registro** para Web Services.
- Provedores de serviços podem utilizar UDDI para publicar os serviços que eles oferecem.

## **WSDL (Web Services Description Language)**

- Trata-se de um documento escrito em XML que além de **descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis**
- padrão baseado em XML para descrever o serviço como no COM, onde ele traz os métodos do webservice.

## **SOAP (Simple Object Access Protocol)**

- **Protocolo padronizado para troca de informações estruturadas entre plataforma descentralizada e distribuída usando como base o XML.**
- Envelope que define a estrutura para descrever o conteúdo da mensagem e como processá-lo

# Arquitetura orientada a serviços (SOA)

- **O XML** é ideal para comunicação entre redes heterogêneas porque suas propriedades baseadas em texto tornam-o independente de plataformas.
- **O SOAP** (Simple Object Access Protocol) é a estrutura (framework) para a comunicação baseada em XML. O SOAP, um protocolo baseado em XML para troca de informações entre sistemas distribuídos, foi projetado para ser um protocolo simples e leve, a ser usado no acesso a objetos em servidores remotos através de mensagens XML. Uma mensagem SOAP consiste em um documento XML que contém informações a serem transmitidas de um sistema para outro. Por exemplo, você pode enviar uma mensagem XML para ativar uma chamada de função na máquina de destino. Quando a máquina de destino termina de processar a função, ela envia o resultado de volta à máquina de origem na forma de uma mensagem SOAP.
- Você também pode usar o HTTP para transportar mensagens SOAP para o servidor de destino. O HTTP facilita o transporte de mensagens SOAP entre os sistemas porque os firewalls normalmente não bloqueiam o acesso à porta HTTP. E, ao transferir mensagens em XML baseado em texto, os servidores em execução em diferentes plataformas ainda serão capazes de se comunicar. Embora o SOAP costume fazer uso do HTTP como o protocolo de transmissão, ele também usa sem problemas protocolos como FTP, SMTP etc.

# Arquitetura orientada a serviços (SOA)

- Em vez de enviar informações de marcação para o usuário, é mais eficiente enviar apenas os dados solicitados. Uma vez chamados os dados, os Web Services — como uma função exposta pelo servidor Web — retornarão apenas os resultados apropriados. Esse modelo é muito similar aos aplicativos Web. Na verdade, os Web services não são muito diferentes dos aplicativos Web, a não ser por uma notável exceção: os aplicativos Web oferecem tanto a funcionalidade como os elementos de interface de usuário, ao passo que os Web services se concentram apenas na funcionalidade
- Para usar os serviços, o consumidor (nesse caso, o aplicativo) precisa estar a par de informações como o tipo de entrada exigida e o resultado gerado pelo Web service. No caso de um site da bolsa de valores, o programador precisa conhecer as informações esperadas pelo Web service e as informações que este retornará, para que o aplicativo que esteja escrevendo possa se comunicar com o Web service de forma apropriada. Você pode usar o **WSDL** (Web Services Description Language) para criar um documento XML que especifique os Web services oferecidos pelo servidor. O arquivo WSDL se assemelha a um contrato legal, já que descreve exatamente como as duas partes (provedor de Web services e consumidor de Web services) deverão se comunicar e cooperar entre si. Além disso, especifica o formato de mensagem esperado pelo Web service. O WSDL preserva o consumidor de Web services do trabalho interno ocorrido nos Web services (veja a Figura 1).

# Arquitetura orientada a serviços (SOA)

- Com tantos Web services disponíveis na rede, você deve estar se perguntando como fazer para encontrá-los. Da mesma forma que os mecanismos de busca como Yahoo! e Google o ajudam a localizar os dados certos nas páginas Web, o **UDDI** (Universal Description, Discovery, and Integration) define uma maneira padrão de os Web services publicarem informações e anunciem seus serviços. O UDDI é basicamente um diretório de Web services que oferece às empresas uma maneira fácil de registrar e localizar serviços.
- O diretório permite que os consumidores de Web services executem buscas programáticas por Web services relevantes. Assim que um consumidor souber onde reside um dado Web service, ele precisará localizar o WSDL do Web service. Os Web services fornecem um arquivo de busca em formato XML. Você pode juntar os vários componentes descritos até aqui para criar um Web service amplamente funcional. Veja como funciona: o consumidor de Web services consulta o UDDI em relação a um Web service e o UDDI retorna a lista de Web services disponíveis. Em seguida, o consumidor envia uma solicitação do documento buscado ao Web service e este o retorna. O consumidor então envia um pedido de um documento WSDL ao Web service, e este também o retorna. Por fim, o consumidor envia uma solicitação de serviço e o Web service retorna uma resposta de serviço

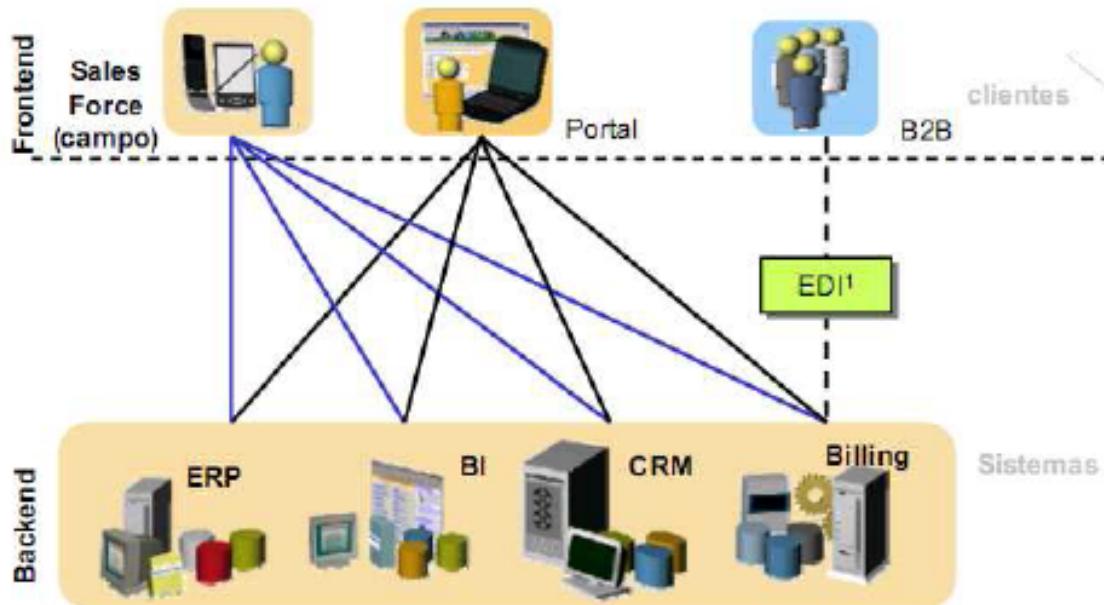
# Arquitetura orientada a serviços (SOA)

- There is a strong theme of metadata and policy running through the Web services standards.
- SOAP services are normally described using WSDL (Web Services Description Language) and can be located by searching a UDDI (Universal Description, Discovery and Integration) directory.
- WSDL is used to describe Web services, including their interfaces, methods and parameters.

# Arquitetura orientada a serviços (SOA)

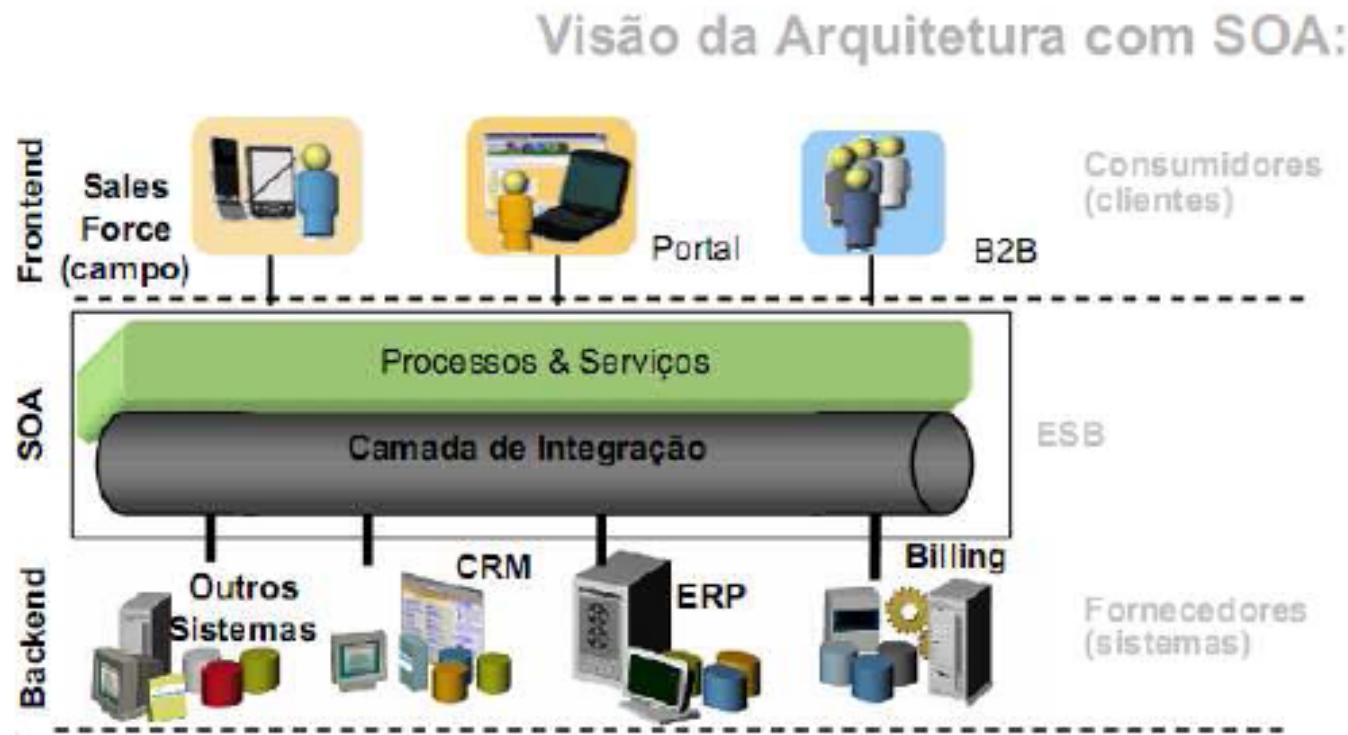
- Visão Tradicional

Visão da Arquitetura (tradicional):



# Arquitetura orientada a serviços (SOA)

- Visão com SOA



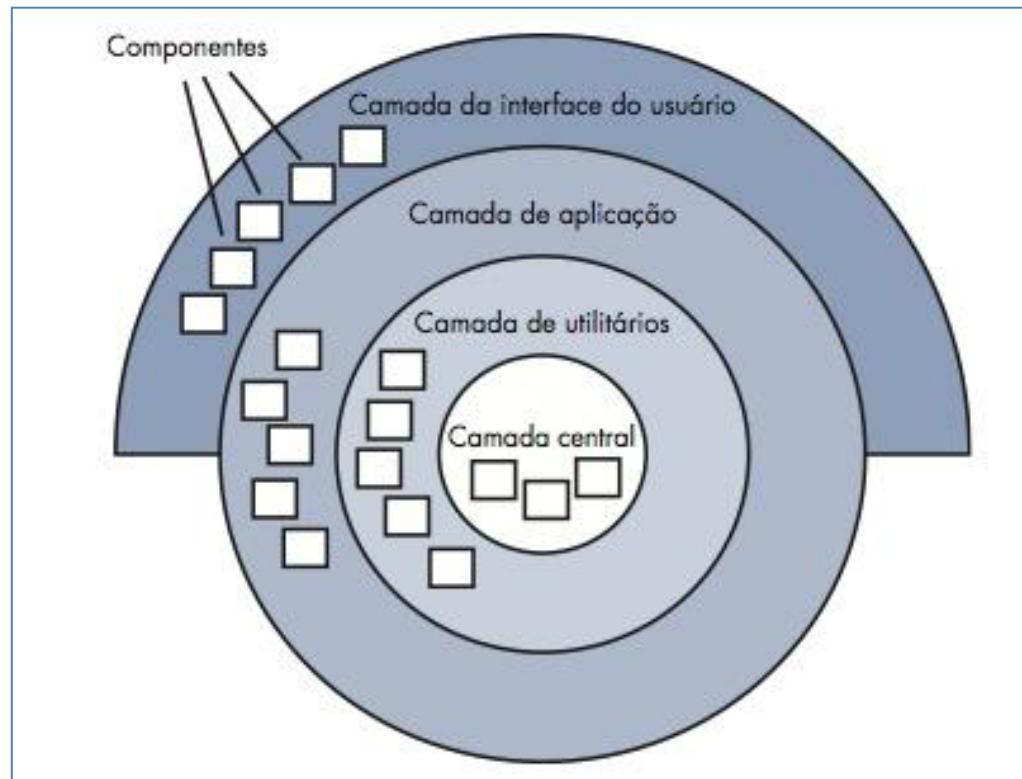
# Estilos de arquitetura

## 4. Arquitetura em camadas

- São definidas várias camadas diferentes, cada uma realizando operações que progressivamente se tornam mais próximas do conjunto de instruções de máquina.
  - Na camada mais externa, os componentes atendem operações de interface do usuário.
  - Na camadas mais interna, os componentes realizam a interface com o sistema operacional.
  - As camadas intermediárias fornecem serviços utilitários e funções de software de aplicação.
- Esses estilos de arquitetura são apenas um pequeno subconjunto dos disponíveis.
- Assim que a engenharia de requisitos revelar as características e restrições do sistema a ser construído, o estilo e/ou combinação de padrões de arquitetura que melhor se encaixa nessas características e restrições pode ser escolhido.
- Em muitos casos, mais de um padrão poderia ser apropriado.
- Por exemplo, um estilo em camadas (apropriado para a maioria dos sistemas) pode ser combinado com uma arquitetura centralizada em dados em diversas aplicações de BD.

# Estilos de arquitetura

- Arquitetura em camadas

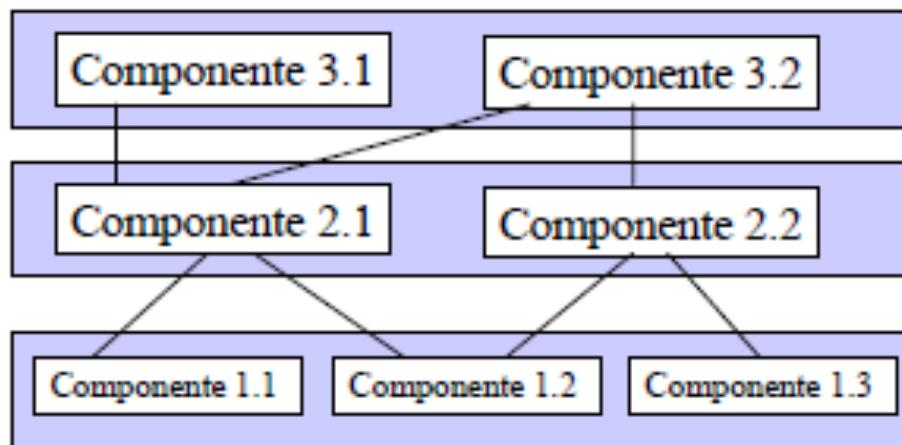


# Estilos de arquitetura

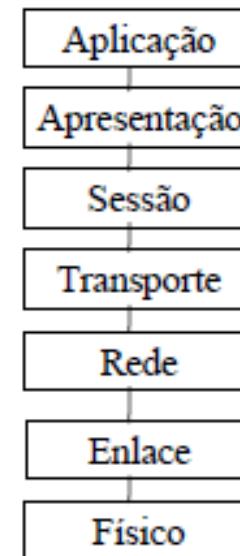
- Um sistema em *camadas* é *organizado* hierarquicamente, cada camada oferecendo serviço a camada acima dela e servindo como cliente da camada inferior.
- Componentes:
  - São representados por cada camada
- Conectores:
  - São definidos pelos protocolos que determinam como cada camada irá interagir com outra
  - Limitam as interações a camadas adjacentes
- Protocolos de rede são os melhores exemplos de arquiteturas em camadas
  - Definem um conjunto de regras e convenções que descrevem como programas de computador em diferentes máquinas comunicam-se
  - O formato, conteúdo e significado das mensagens são definidas

# Estilos de arquitetura

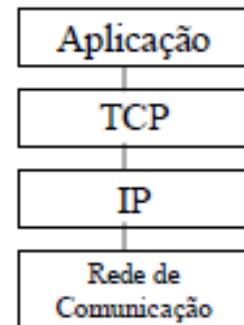
- Cada camada individual pode ser uma entidade complexa consistindo de diferentes componentes.



Modelo OSI da ISO



Modelo TCP/IP



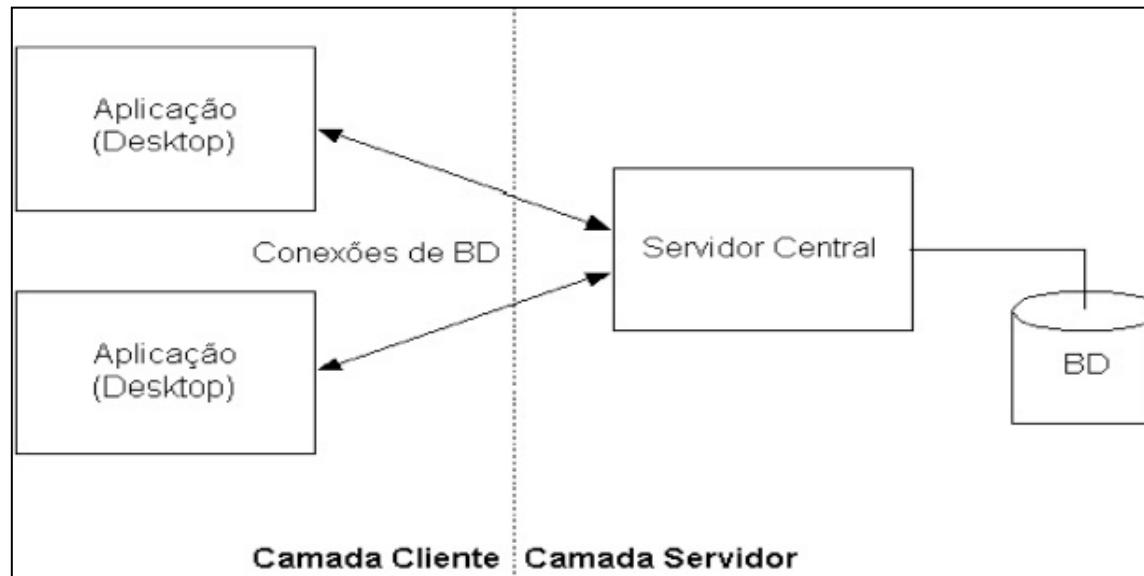
# Estilos de arquitetura

- Definir o **critério de abstração para agrupar** tarefas em camadas
  - Exemplo: a distância do hardware pode formar os níveis mais baixos e a complexidade conceitual os níveis mais altos
- Determinar o número de níveis de abstração de acordo com seu critério de abstração
- Nomear as camadas e determine as tarefas de cada uma delas
  - A tarefa da camada mais alta é a percebida pelo cliente
  - As tarefas das demais camadas visam ajudar a realização da tarefa da camada mais alta.
- Especificar os serviços
- Especificar uma interface para cada camada
- Refinar cada camada:
  - Estruturação de cada camada individualmente
  - Quando uma camada é complexa ela deve ser separada em componentes individuais e cada componente pode seguir um padrão ou estilo diferente.

# Estilos de arquitetura

- **A arquitetura tradicional Cliente/Servidor**
- A antiga estrutura centralizadora do processamento baseada em **Mainframes** foi sucedida por uma arquitetura que se organizava em duas 'camadas', chamada de arquitetura **Cliente/Servidor**.
- Basicamente essa estrutura consiste em **separar o processamento em duas partes** - uma parte é realizado na máquina do usuário (cliente) e a outra numa máquina poderosa que concentra as informações (servidor).
- Nos aplicativos Cliente/Servidor o computador servidor é diretamente conectado e responsável por cada computador cliente que está operando com suas informações.
- Desse modo, se existem 1000 usuários no mundo acessando um dado aplicativo, seu servidor terá que usar seus recursos para gerenciar todas as solicitações de cada um deles.
- A inviabilidade desse tipo de responsabilidade numa época em que alguns dos softwares mais populares da web são acessados por milhões de usuários diariamente criou a necessidade de uma nova arquitetura de software - uma arquitetura que fosse capaz de dar conectividade e produtividade a grandes quantidades de usuários simultâneos; surgiu a arquitetura em multicamadas.

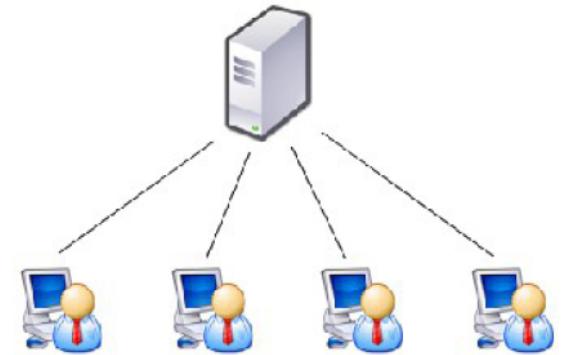
# Estilos de arquitetura



Fat client

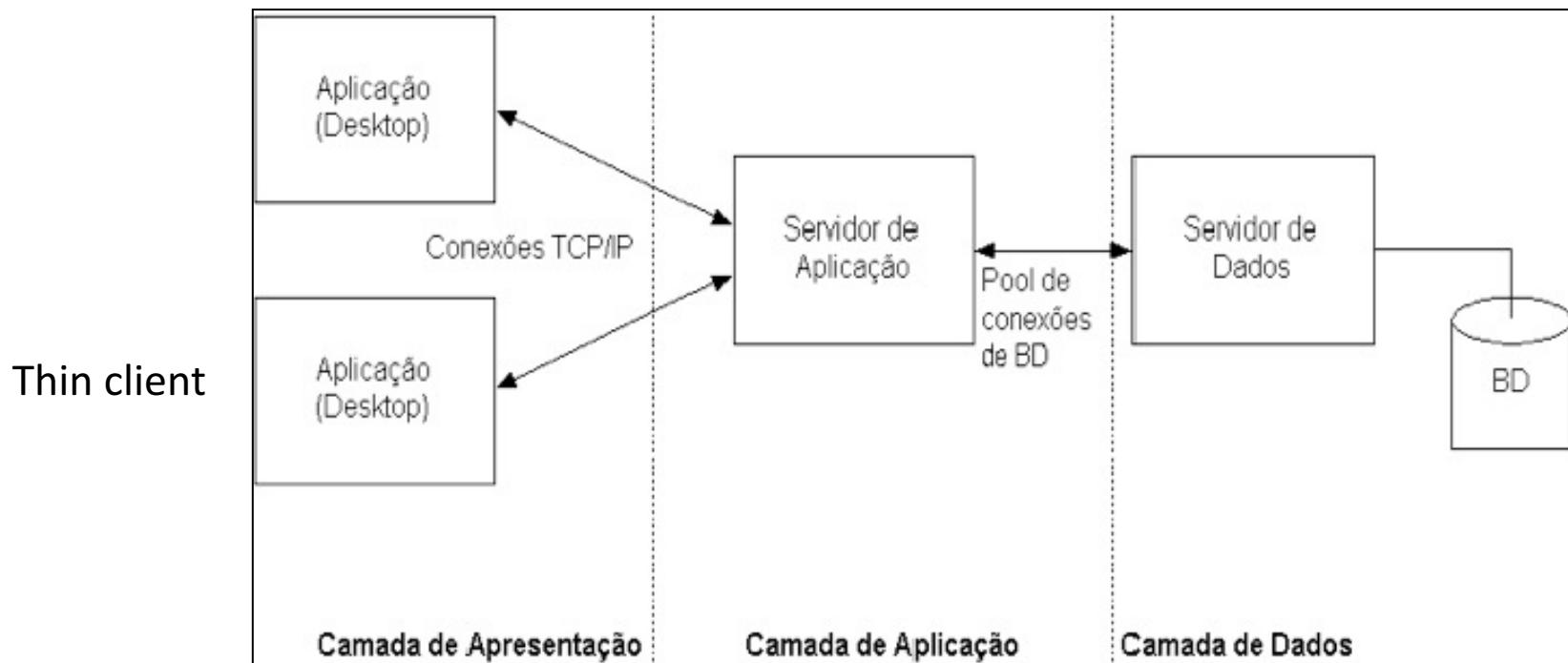
Os primeiros sistemas cliente-servidor eram em duas camadas:

- Camada cliente trata da lógica de negócio e da UI
- Camada servidor trata dos dados (usando um SGBD)



# Estilos de arquitetura

- Três camadas

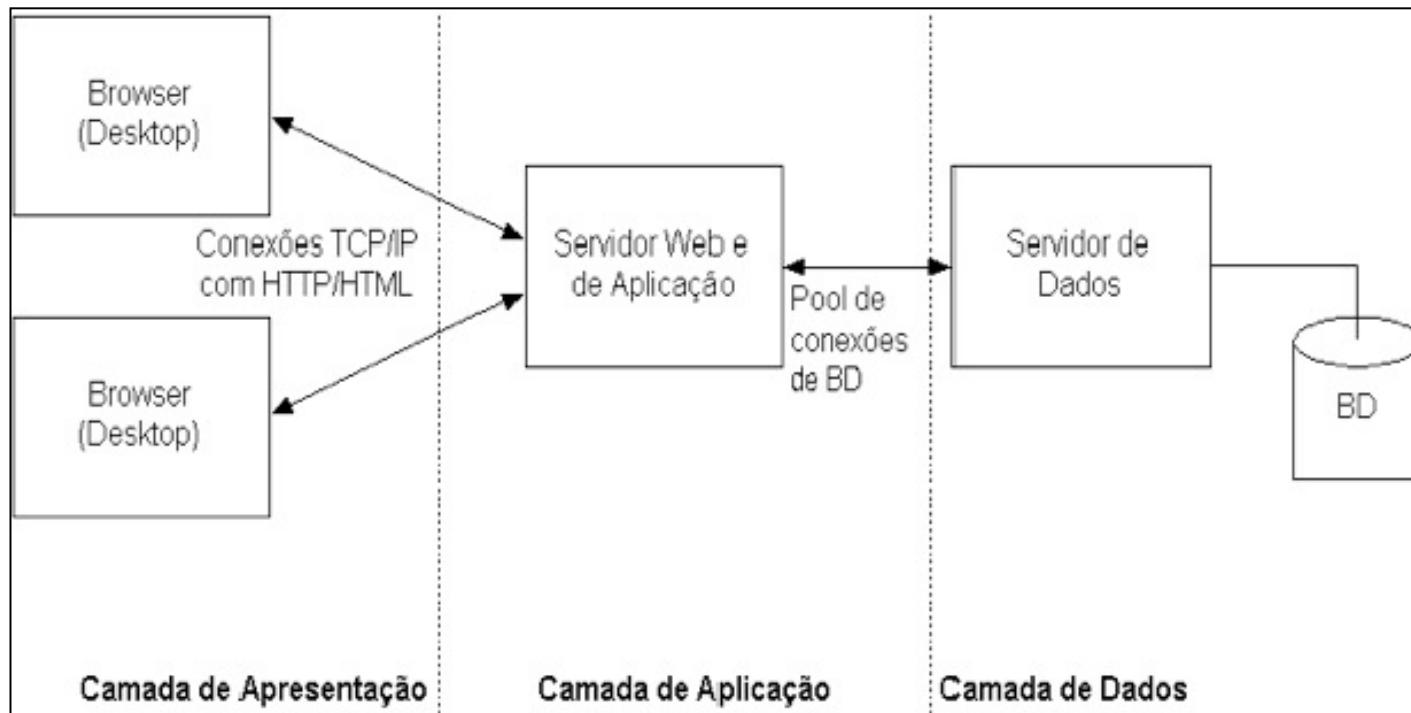


As camadas são lógicas:

- Fisicamente, várias camadas podem executar na mesma máquina
- Quase sempre, há separação física de máquinas

# Estilos de arquitetura

- Arquitetura em 3/4 camadas baseada na Web



browser

# Estilos de arquitetura

- Uso de browser como cliente;
- Conceito de Intranet
- A camada de aplicação se quebra em duas: Web e Aplicação
- Evitamos instalar qualquer software no desktop e portanto, problemas de manutenção
- Evita instalação em computadores de clientes, parceiros, fornecedores, etc.

# Estilos de arquitetura

- **O modelo em Multicamadas**
- Um software que trabalha em múltiplas camadas ainda é responsável por todas as conexões ligadas a ele, mas não o faz diretamente. Há uma separação clara entre as diversas responsabilidades envolvidas no desempenho de cada função, de modo que o programa não é mais dividido entre um servidor e um cliente, mas em três ou mais camadas que se comunicam entre si quando têm necessidade.
- No entanto, o que torna essa abordagem diferente é o fato de que cada uma de suas camadas funciona de forma separada e inteligente, levando em consideração a estrutura e a semântica de cada função a ser exercida. Isso é um imenso salto de qualidade, mas exige que o projeto e a implementação sejam, ambos, planejados e construídos de forma especial; assim é possível atingir o resultado desejado.

# Estilos de arquitetura

- **Implementando sistemas em multicamadas**

É necessário que o projeto tenha sido idealizado de modo a produzir e comportar uma arquitetura em multicamadas, para que seja possível implementar um aplicativo que apresente os benefícios esperados.

Uma arquitetura em três camadas pode ser implementada em três máquinas separadas, ou mesmo em módulos separados no mesmo computador; a grande diferença de implementação entre esta e a tradicional arquitetura cliente-servidor é a presença de um programa chamado de **Servidor de Aplicativos**.

Ele irá gerenciar o reaproveitamento de recursos e a conectividade - tanto com o banco de dados como com a camada de aplicação.

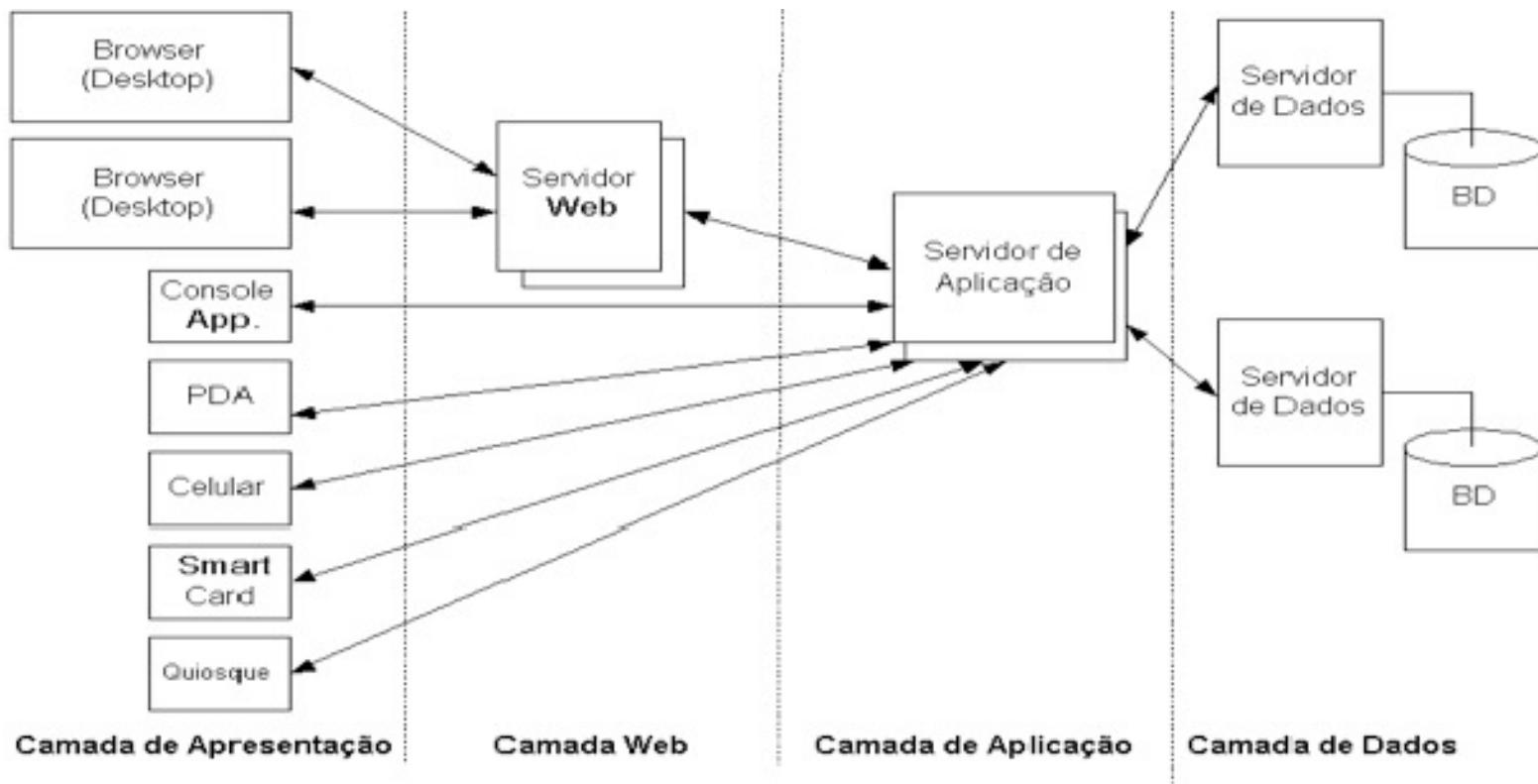
Sua presença, entretanto, cria dois importantes fatores a considerar: aumento de custo e de complexidade do desenvolvimento.

# Estilos de arquitetura

- Um modelo em 3 camadas para a web seria tipicamente organizado da seguinte maneira
- **Camada de Apresentação** - representada pelo navegador e pelo servidor web, organiza e exibe as informações para o usuário; recebe comandos e organiza a forma de entrada e saída. Geralmente é programada numa linguagem de script, e é de desenvolvimento mais barato.
- **Camada de Lógica de Aplicação** - representada por um conjunto de objetos que contém a lógica de negócios do aplicativo; esses objetos ficam hospedados num Servidor de Aplicativos. Pode ser reusada por várias formas de apresentação diferentes, e se comunica com a camada de serviços de dados para prover informações e acionar operações.
- **Camada de Serviços de Dados** - representada pelo servidor de dados e pela abstração de dados, acesso a programas legados e sistema de arquivos. Somente pode ser acessada pela camada de lógica de aplicação, o que aumenta a segurança do sistema

# Estilos de arquitetura

- Multicamadas



# Padrões de arquitetura

# Padrões de arquitetura

Um padrão de arquitetura, assim como um estilo arquitetural, impõe transformação no projeto de arquitetura. Entretanto, padrão difere de estilo em alguns modos fundamentais: (1) o escopo de um padrão é menos abrangente, concentrando-se em um aspecto da arquitetura e não na arquitetura em sua totalidade; (2) um padrão impõe uma regra sobre a arquitetura, descrevendo como o software irá tratar algum aspecto de sua funcionalidade em termos de infraestrutura (por exemplo, concorrência) [Bos00]; (3) os padrões de arquitetura (Seção 9.4) tendem a tratar questões comportamentais específicas no contexto da arquitetura (por exemplo, como as aplicações em tempo real tratam a sincronização ou as interrupções). Os padrões podem ser usados com um estilo de arquitetura para dar forma à estrutura global de um sistema. Na Seção 9.3.1, consideraremos os padrões e estilos de arquitetura e padrões mais comumente utilizados em software.

À medida que o modelo de requisitos é desenvolvido, poderemos perceber que o software deve tratar uma série de problemas mais amplos que envolvem toda a aplicação. Por exemplo, o modelo de requisitos para praticamente qualquer aplicação de comércio eletrônico depara com o seguinte problema: *Como oferecer uma ampla gama de produtos para uma ampla gama de clientes e permitir que esses clientes comprem nossos artigos on-line?*

# Padrões de arquitetura

O modelo de requisitos também define um contexto no qual essa questão deve ser respondida. Por exemplo, uma aplicação de comércio eletrônico que vende equipamento de golfe para clientes irá operar em um contexto diferente daquele de uma aplicação de comércio eletrônico que vende equipamentos industriais de preço elevado para empresas de médio e grande porte. Além disso, um conjunto de limitações e restrições poderia afetar a forma que tratamos o problema a ser resolvido.

Os padrões de arquitetura tratam um problema específico de aplicação em um contexto específico e sob um conjunto de limitações e restrições. O padrão propõe uma solução de arquitetura capaz de servir como base para o projeto da arquitetura.

Citamos anteriormente, neste capítulo, que a maioria das aplicações enquadra-se em um domínio ou gênero específico e que um ou mais estilos de arquitetura poderiam ser apropriados para aquele gênero. Por exemplo, o estilo de arquitetura geral para uma aplicação poderia ser de chamadas e retornos ou orientado a objetos. Porém, nesse estilo, encontraremos um conjunto de problemas comuns que poderiam ser mais bem tratados com padrões de arquitetura específicos. Alguns desses problemas e uma discussão mais completa de padrões de arquitetura são apresentados no Capítulo 12.

# Organização e refinamento

Pelo fato de o processo de projeto muitas vezes nos dar uma série de alternativas de arquitetura, é importante estabelecer um conjunto de critérios de projeto que podem ser usados para avaliar o projeto da arquitetura obtido. As seguintes questões [Bas03] dão uma visão mais clara sobre um estilo de arquitetura:

**Controle.** Como o controle é gerenciado na arquitetura? Existe uma hierarquia de controle distinta e, em caso positivo, qual o papel dos componentes nessa hierarquia de controle? Como os componentes transferem controle no sistema? Como o controle é compartilhado entre os componentes? Qual a topologia de controle (ou seja, a forma geométrica que o controle assume)? O controle é sincronizado ou os componentes operam de maneira assíncrona?

**Dados.** Como os dados são transmitidos entre os componentes? O fluxo de dados é contínuo ou os objetos de dados são passados esporadicamente para o sistema? Qual o modo de transferência de dados (ou seja, os dados são passados de um componente para outro ou os dados estão disponíveis globalmente para ser compartilhados entre os componentes de sistema)? Os componentes de dados (por exemplo, um quadro-negro ou repositório) existem e, em caso positivo, qual o seu papel? Como os componentes funcionais interagem com os componentes de dados? Os componentes de dados são passivos ou ativos (isto é, o componente de dados interage ativamente com outros componentes do sistema)? Como os dados e controle interagem no sistema?

Essas questões dão ao projetista uma avaliação prévia da qualidade do projeto e formam a base para análise mais detalhada da arquitetura.

# Componente

- O que é

*Componente* é um bloco construtivo modular para software de computador. Mais formalmente, a Especificação da Linguagem de Modelagem Unificada da OMG (*OMG Unified Modeling Language Specification* [OMG03a]) define componente como "... Uma parte modular, possível de ser implantada e substituível de um sistema que encapsula implementação e expõe um conjunto de interfaces".

- Visão OO

No contexto da engenharia de software orientada a objetos, um componente contém um conjunto de classes colaborativas.<sup>1</sup> Cada classe contida em um componente foi completamente elaborada para incluir todos os atributos e operações relevantes à sua implementação. Como parte da elaboração do projeto, todas as interfaces que permitem que as classes se comuniquem e colaborem com outras classes de projeto também precisam ser definidas. Para tanto, começamos com o modelo de requisitos e elaboramos as classes de análise (para componentes que se relacionam com o domínio do problema), bem como as classes de infraestrutura (para componentes que oferecem suporte a serviços para o domínio do problema).

# Engenharia de Software baseada em Componente

- Engenharia de software baseada em componentes (CBSE) (também conhecida como desenvolvimento baseado em componentes (CBD)) é um ramo da engenharia de software que enfatiza a separação de preocupações em relação à ampla funcionalidade disponível de um determinado sistema de software.
- É uma abordagem baseada na reutilização para definição, implementação e composição de componentes independentes de baixo acoplamento em sistemas.
- Esta prática tem como objetivo trazer um grau igualmente amplo de benefícios, tanto a curto prazo e a longo prazo para o software em si e para as organizações que patrocinam esse tipo de software.
- Os engenheiros de software consideram componentes como parte da plataforma de partida para orientação a serviço.
- Componentes desempenham esse papel, por exemplo, em serviços web e, mais recentemente, em arquiteturas orientadas a serviços (SOA), em que um componente é convertido pelo serviço web em um serviço e, posteriormente, herda mais características que vão além de um componente comum.

# Engenharia de Software baseada em Componente

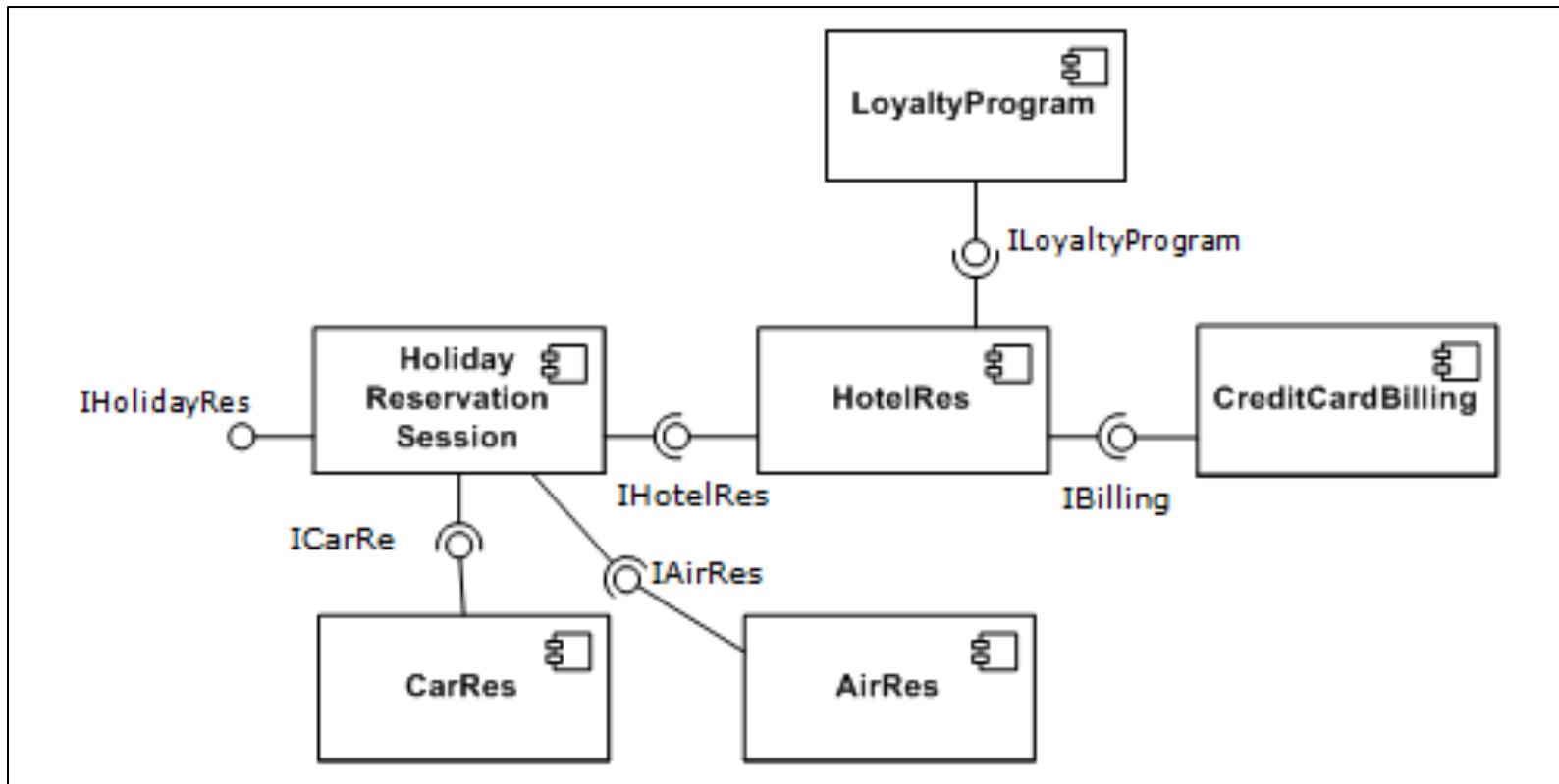
- **Definição e características de componentes**
- Um componente de software individual é um pacote de software, um serviço web, um recurso da Web, ou um módulo que encapsula um conjunto de funções relacionadas (ou dados).
- Todos os processos do sistema são colocadas em componentes separados, de modo que todos os dados e das funções dentro de cada componente são semanticamente relacionados (tal como com o conteúdo de classes). Devido a este princípio, muitas vezes é dito que os componentes são modulares e coesos.
- No que se refere a uma coordenação para todo o sistema, os componentes se comunicam uns com os outros através de interfaces.
- Quando um componente oferece serviços para o resto do sistema, ele adota uma interface que especifica os serviços que podem utilizar outros componentes, e como eles podem fazê-lo.
- Esta interface pode ser vista como uma assinatura do componente - o cliente não precisa saber sobre o funcionamento interno do componente (implementação), a fim de fazer uso dele. Este princípio resulta em componentes referidos como encapsulados.

# Engenharia de Software baseada em Componente

- A ilustração UML representa *interfaces fornecidas* por um símbolo de pirulito ligado ao bordo exterior do componente.
- No entanto, quando um componente precisa usar outro componente para funcionar, ele adota uma *interface usada* (used interface) que especifica os serviços de que necessita.
- Na ilustração UML, as *interfaces usadas* são representados pelo símbolo encaixe aberto ligado ao bordo exterior do componente.
- Outro atributo importante dos componentes é que eles são substituíveis, de modo que um componente pode substituir outro (em tempo de design ou de execução), se o componente sucessor satisfaz os requisitos do componente inicial (expressa através das interfaces).
- Por conseguinte, os componentes podem ser substituídos por uma versão atualizada ou uma alternativa sem quebrar o sistema em que o componente opera.

# Engenharia de Software baseada em Componente

- Exemplo:



# Engenharia de Software baseada em Componente

- Como uma regra geral para engenheiros substituindo componentes, o componente B pode substituir imediatamente o componente A, se o componente B fornece pelo menos o que o componente A fornecia e não usa mais do que o componente A usava.
- Componentes de software muitas vezes tomam a forma de objetos (e não classes) ou coleções de objetos (de programação orientada a objetos), de uma forma binária ou textual, aderente a alguma linguagem de descrição de interface (IDL) para que o componente possa existir de forma autônoma de outros componentes em um computador.

# Engenharia de Software baseada em Componente

- Reutilização é uma característica importante de um componente de software de alta qualidade.
- Os programadores devem projetar e implementar componentes de software de tal forma que muitos programas diferentes podem reutilizá-los.
- Além disso, testes de usabilidade baseado em componentes devem ser considerados quando os componentes de software interagirem diretamente com os usuários.  
É preciso esforço e conscientização significativa para escrever um componente de software que é efetivamente reutilizável.
- O componente tem de ser:
  - totalmente documentado
  - exaustivamente testado
    - robusto - com abrangente verificação de validade de entrada
    - capaz de passar de volta mensagens de erro apropriadas ou códigos de retorno
  - projetado com a consciência de que ele será colocado para usos imprevistos.

# Engenharia de Software baseada em Componente

- **Arquitetura**
- Um computador executando vários componentes de software é muitas vezes chamado de um servidor de aplicativo.
- Esta combinação de servidores de aplicativos e componentes de software é geralmente chamado de computação distribuída.
- Aplicação no mundo real disso é, por exemplo, aplicações financeiras ou software de negócios.