Padrão Camadas

O Padrão Layers (Camadas)

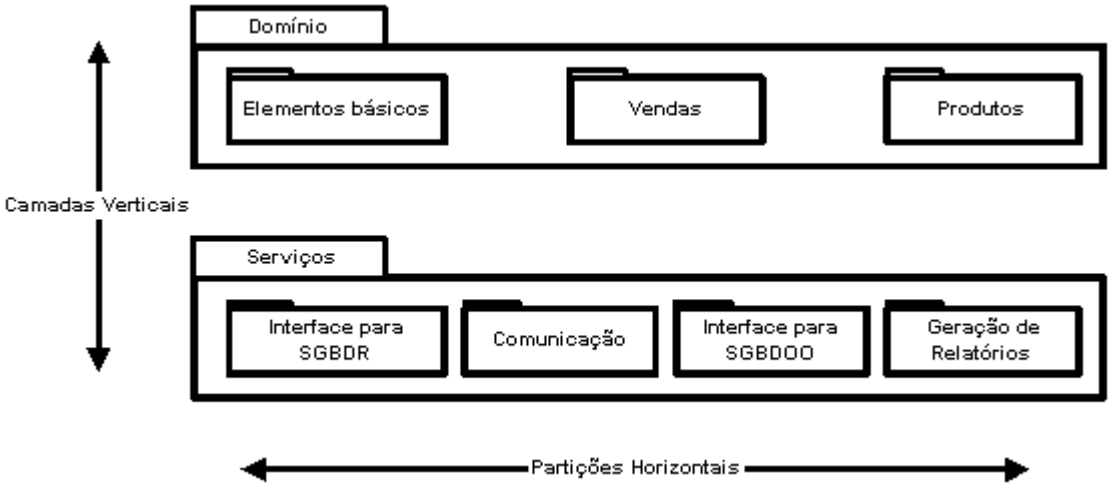
- Talvez o mais importante padrão arquitetural
  - Merece detalhamento

Problema

- Imagine que esteja projetando um sistema cuja característica principal é uma mistura de assuntos de alto nível com assuntos de baixo nível, em que os assuntos de alto nível usam os assuntos de baixo nível.
- A parte de baixo nível está frequentemente perto do hardware
- A parte de mais alto nível está frequentemente perto do usuário
- O fluxo de comunicação tipicamente consiste de pedidos fluindo do alto para o baixo níveis
  - As respostas andam na direção contrária

Solução:

- Decomposição do sistema: partições e camadas
- Uma estrutura elegante pode freqüentemente ser elaborada usando camadas e partições
  -
    - Uma camada é um subsistema que adiciona valor a subsistemas de menor nível de abstração
    - Uma partição é um subsistema "paralelo" a outros subsistemas



- Lembre que subsistemas devem ser coesos (trata de responsabilidades fortemente relacionadas)
  - Tem forte acoplamento dentro de um subsistema
  - Tem fraco acoplamento entre subsistemas
- Para minimizar o acoplamento, camadas freqüentemente se comunicam apenas com as camadas "vizinhas"
- A decomposição pode terminar quando você atingir subsistemas com temas claros que sejam simples de entender
- Alguns sistemas muito pequenos não necessitam de camadas e partições

Implementação

- Descrevemos a seguir alguns detalhes de implementação do padrão

1. Defina o critério de abstração para agrupar tarefas em camadas

- Pode ser a distância conceitual do "chão"
- Pode ser baseado em outro critério dependente do domínio
- Pode ser baseado na complexidade conceitual
- Uma forma comum de criar camadas
  - Nas camadas inferiores, usa a distância do hardware
  - Nas camadas superiores, usa a complexidade conceitual

- Exemplo de camadas:
  - Elementos visíveis ao usuário
  - Módulos específicos da aplicação
  - Nível de serviços comuns
  - Nível de interface ao sistema operacional (ou outra plataforma como uma JVM)
  - O sistema operacional em si que pode estar estuturado em camadas (Microkernel)
  - O hardware

2. Determine o número de níveis de abstração (baseado no critério acima)

- Cada nível de abstração corresponde a uma camada
- Às vezes não é simples decidir se deve haver uma quebra de uma camada em subcamadas ou não
  - Camadas demais podem afetar o overhead
  - Camadas de menos comprometem a estrutura

3. Dê um nome e atribua tarefas a cada camada

- Para a camada de cima, a tarefa é a tarefa global do sistema, do ponto de vista do usuário
- Outras camadas existem para ajudar as camadas de cima
- Pode-se proceder de baixo para cima, mas isso requer bastante experiência
  - Melhor proceder de cima para baixo

4. Especifique os serviços

- O princípio básico é de separar as camadas uma das outras
  - Nenhum módulo abrange duas camadas
- Tente manter mais riqueza acima e mesno abaixo
  - Isso ajuda a prover bons serviços de alto nível para o programador "em cima"
  - Chamamos isso de "pirâmide invertida de reuso"

5. Refine as camadas

- Itere nas 4 etapas acima
- Dificilmente se acerta o critério de abstração de primeira, antes de ver que camadas estão surgindo
- Quando as camadas surgem, pode-se verificar se o critério de abstração está ok

6. Especifique uma interface para cada camada

- Black box (preferível)
  - A camada J nada sabe sobre a camada J-1 e usa uma interface "plana" para acessá-la
  - Pode usar o padrão Façade para organizar a interface
- White box
  - A camada J conhece detalhes da estrutura interna da camada J-1 e usa esse conhecimento ao acessar a camada
    - Por exemplo, conhece vários objetos ou componentes internos da camada

7. Estruture as camadas individuais

- Quebre a camada em subsistemas (partições) menores se ela for complexa

8. Especifique a comunicação entre camadas

- Normalmente, usa-se o modelo "push"
  - A camada J passa a informação necessária para a camada J-1 ao chamá-la
- O modelo pull pode ser usado mas cria mais acoplamento entre camadas
  - Veja discussão de push versus pull no padrão Observer

9. Desacople camadas adjacentes

- Evite situações em que a camada de baixo sabe algo sobre seus clientes (camada acima)
  - Acoplamento unidirecional é preferível

- Por isso, modelo push é melhor

- Para fazer uma chamada de baixo para cima mas ainda manter desacoplamento, use callbacks
  - A camada de cima registra métodos de callback junto à camada de baixo para eventos específicos
  - Quando o evento ocorre na camada de baixo, ela chama o callback
  - Esta desacoplado porque o método está sempre presente na camada de cima
  - Assim, mantemos acoplamento unidirecional, embora haja comunicação bidirecional
  - Um padrão que ajuda a encapsular callbacks é Command
- Claro que a grande técnica de desacoplamento é de usar uma interface

10. Projete uma estratégia de tratamento de erros

- O esforço de programação e overhead podem ser grandes para tratar erros numa arquitetura em camadas
- Um erro pode ser tratado na camada onde foi descoberto ou numa camada superior
  - No segundo caso, deve haver mapeamento para tornar o erro semanticamente reconhecível pela camada de cima
    - Exemplo: Não apresente ao usuário uma mensagem dizendo "Exceção IllegalArgumentException in DoItNow()"
- Tente tratar erros na camada mais baixa possível
  - Isso simplifica todas as camadas superiores que não sabem nem devem saber do erro
  - "O componente mais rápido, mais robusto e mais barato é aquele que não está aí"

Exemplos

- Sietmas de informação implementadas usando Arquiteturas em N Camadas

- Vide adiante
- Máquinas Virtuais
  - Java usa uma VM para isolar camadas de alto nível de detalhes de baixo nível (hardware, plataforma operacional)
- APIs
  - Uma API é uma camada que encapsula camadas inferiores de funcionalidade comumente empregada
- Windows
  - Windows usa as seguintes camadas para se isolar do hardware
    - System Services
    - Resource Management (Processor, I/O, Virtual Memory, Security, ...)
    - Kernel
    - Hardware Abstraction Layer (HAL)
    - Hardware

Consequências

Vantagens

- Reuso de camadas
  - Devido à boa encapsulação provida pelo uso de interfaces
- Suporte para a padronização
  - Certas camadas importantes podem ser padronizadas
    - ex. API POSIX
    - ex. J2EE
  - Permite usar implementações de vários fornecedores
- Dependências são mantidas locais
  - Alterações de implementações que não alterem as interfaces não saem da camada

- Intercambiabilidade
  - Implementações podem ser trocadas

Desvantagens

- Cascatas de mudanças de comportamento
  - Sob certas situções, alterações de comportamento numa camada podem fazer cascata
    - ex. alteração de um enlace de 10 Mbps para 1Gbps pode engargalar camadas superiores
- Menor eficiência
  - Camadas adicionam overhead
  - Um "mar monolítico de objetos" é mais eficiente
    - Porém, mais acoplado

Arquiteturas em n camadas

- Faremos um resumo histórico das arquiteturas de aplicações
- É uma aplicação muito importante do padrão Layers

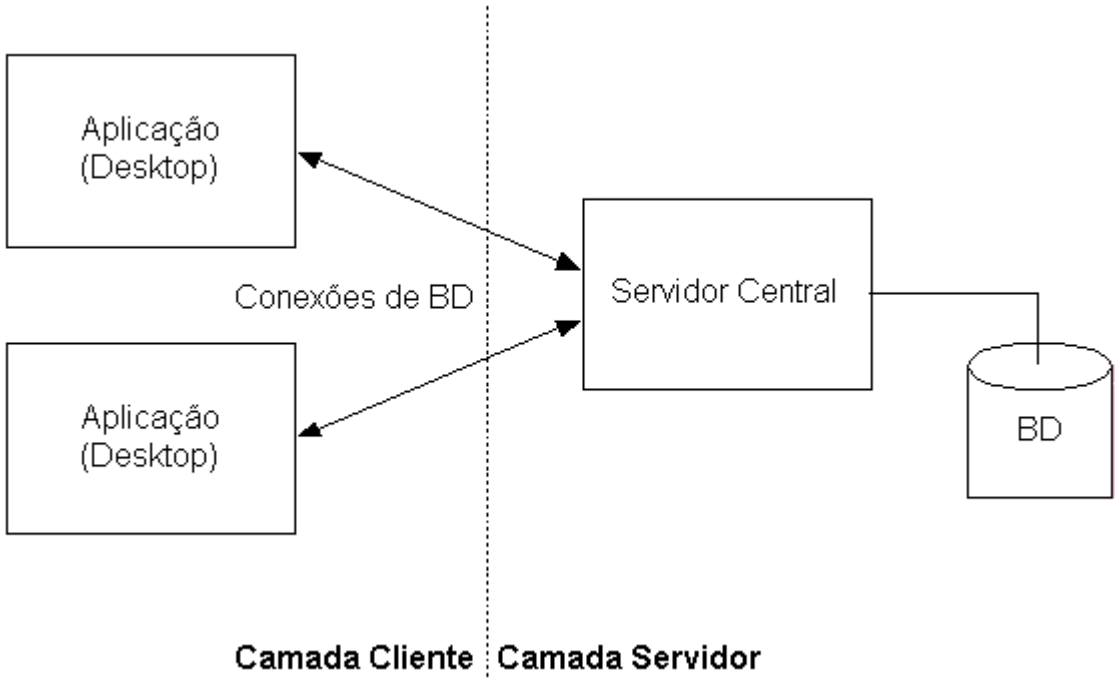
Arquitetura centralizada

- Dominantes até década de 80 como arquitetura corporativa
- Problema básico: interface não amigável

Arquitetura em 2 camadas

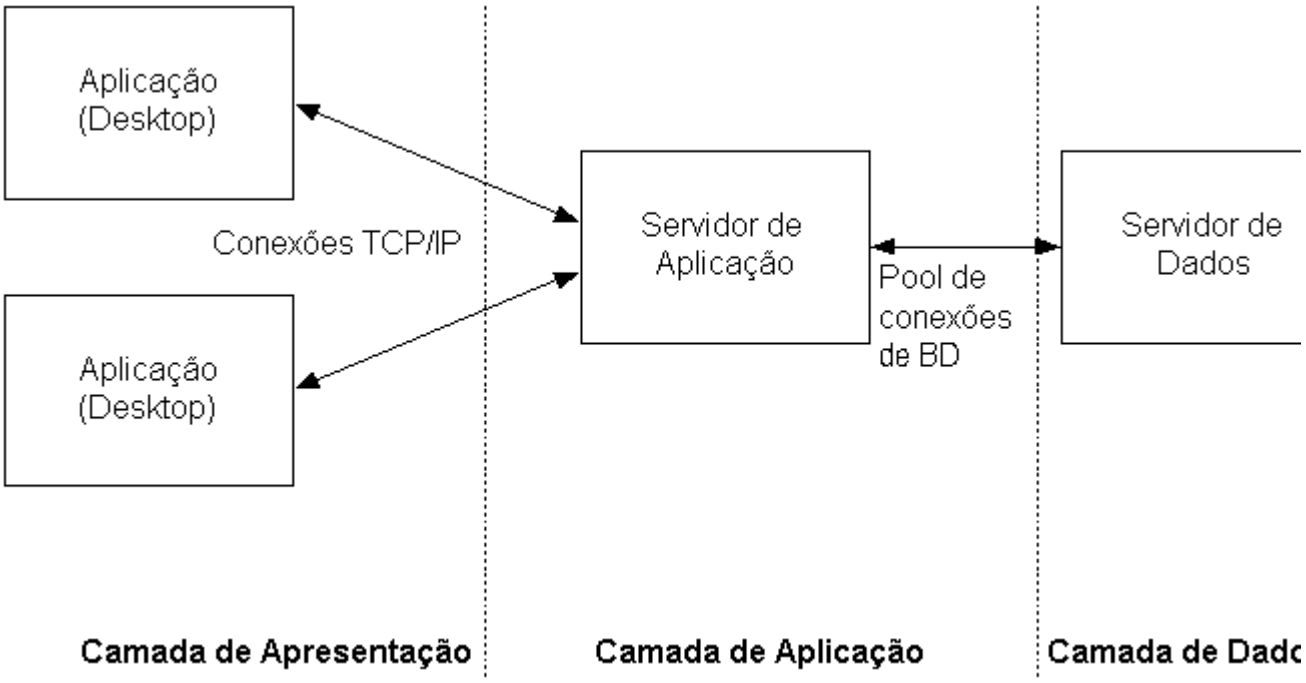
- Sistemas em camadas surgiram para:
  - Melhor aproveitar os PCs da empresa
  - Oferecer sistemas com interfaces gráficas amigáveis
  - Integrar o desktop e os dados corporativos
- Em outras palavras, permitiram aumentar a escalabilidade de uso de Sistemas de Informação
- Os primeiros sistemas cliente-servidor eram de duas camadas
  - Camada cliente trata da lógica de negócio e da UI

- Camada servidor trata dos dados (usando um SGBD)



Arquitetura em 3 camadas

- A arquitetura cliente/servidor em 2 camadas sofria de vários problemas:
  - Falta de escalabilidade (conexões a bancos de dados)
  - Enormes problemas de manutenção (mudanças na lógica de aplicação forçava instalações)
  - Dificuldade de acessar fontes heterogêneas (legado CICS, 3270, ...)
- Inventou-se a arquitetura em 3 camadas
  - Camada de apresentação (UI)
  - Camada de aplicação (business logic)
  - Camada de dados

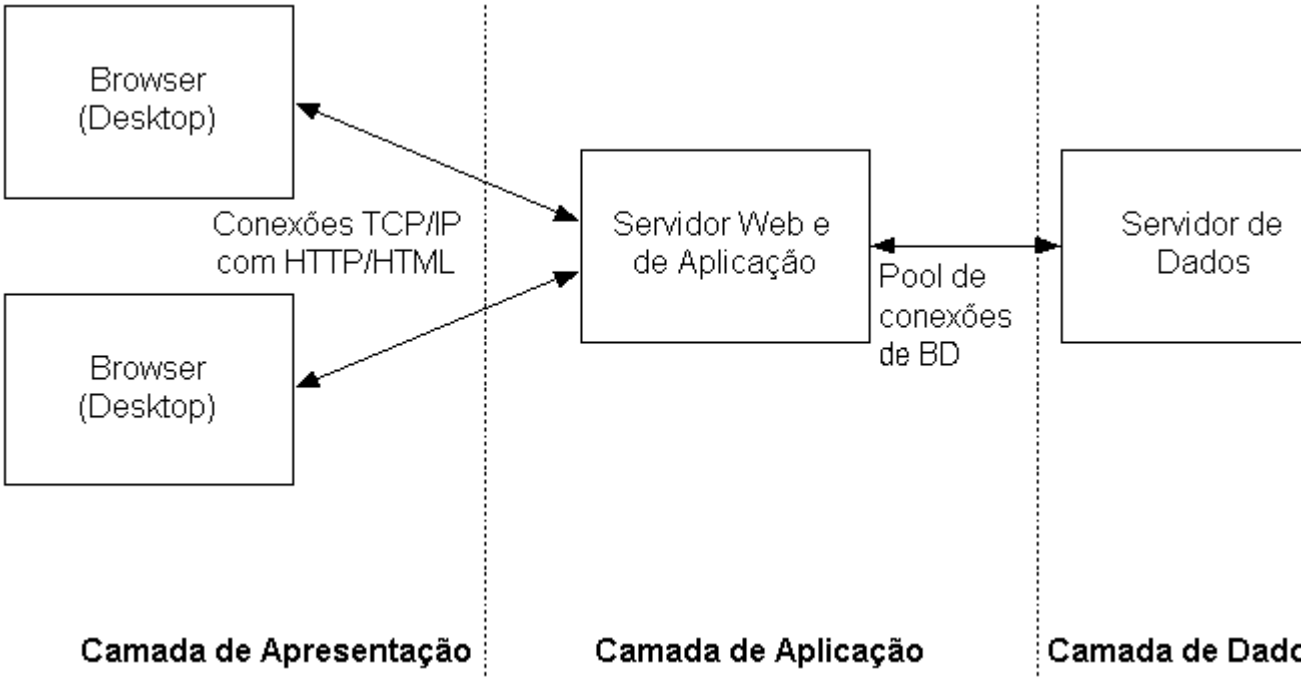


- Problemas de manutenção foram reduzidos, pois mudanças às camadas de aplicação e de dados não necessitam de novas instalações no desktop
- Observe que as camadas são lógicas
  - Fisicamente, várias camadas podem executar na mesma máquina
  - Quase sempre, há separação física de máquinas
  - Em inglês, usa-se "layer" e "tier"
    - Para certas pessoas, não tem diferença
    - Para outras, "tier" indicaria camada física

Arquitetura em 3/4 camadas Web-Based

- A arquitetura em 3 camadas original sofre de problemas:
  - A instalação inicial dos programas no desktop é cara
  - O problema de manutenção ainda persiste quando há mudanças à camada de apresentação
  - Não se pode instalar software facilmente num desktop que não está sob seu controle administrativo
    - Em máquinas de parceiros
    - Em máquinas de fornecedores
    - Em máquinas de grandes clientes

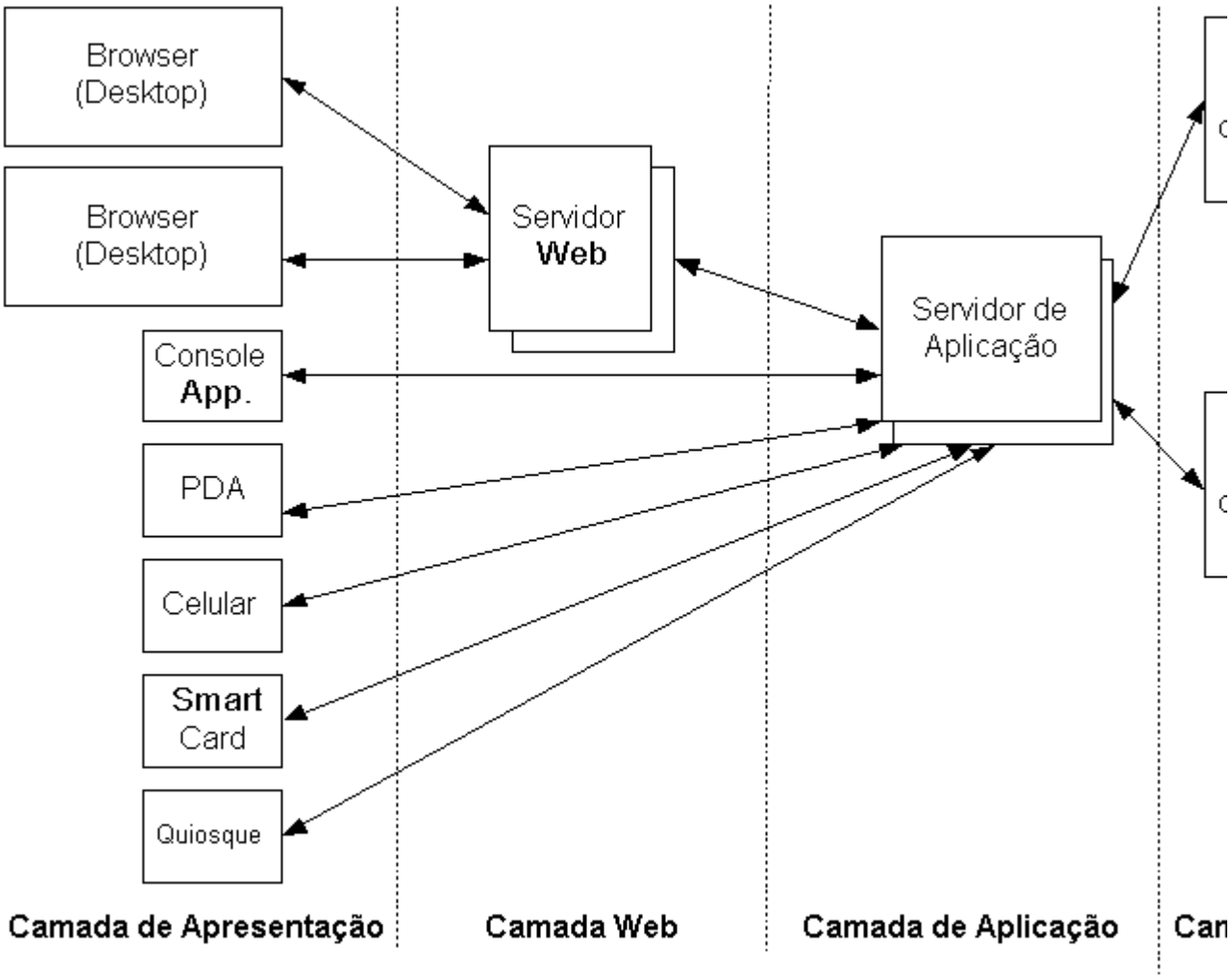
- Em máquinas na Internet
- Então, usamos o Browser como Cliente Universal
  - Conceito de Intranet
  - A camada de aplicação se quebra em duas: Web e Aplicação
  - Evitamos instalar qualquer software no desktop e portanto, problemas de manutenção
  - Evitar instalação em computadores de clientes, parceiros, fornecedores, etc.
- Às vezes, continua-se a chamar isso de 3 camadas porque as camadas Web e Aplicação freqüentemente rodam na mesma máquina (para pequenos volumes)



Arquitetura distribuída em n camadas

- Os problemas remanescentes:
  - Não há suporte a Thin Clients (PDA, celulares, smart cards, quiosques, ...) pois preciso usar um browser (pesado) no cliente
  - Dificuldade de criar software reutilizável: cadê a componentização?
  - Fazer aplicações distribuídas multicamadas é difícil. Tem que:
    - Implementar persistência (*impedance mismatch* entre o mundo OO e o mundo dos BDs

relacionais)
  - Implementar tolerância a falhas com fail-over
  - Implementar gerência de transações distribuídas
  - Implementar balanceamento de carga
  - Implementar *resource pooling*
  - etc.
- As empresas não querem contratar PhDs para implementar sistemas de informação!
- O truque é introduzir *middleware* num servidor de aplicação que ofereça esses serviços automaticamente
- Além do mais, as soluções oferecidas (J2EE, .Net) são baseadas em componentes



- As camadas podem ter vários nomes:
  - Apresentação, interface, cliente
  - Web
  - Aplicação, Business
  - Dados, Enterprise Information System (EIS)

programa