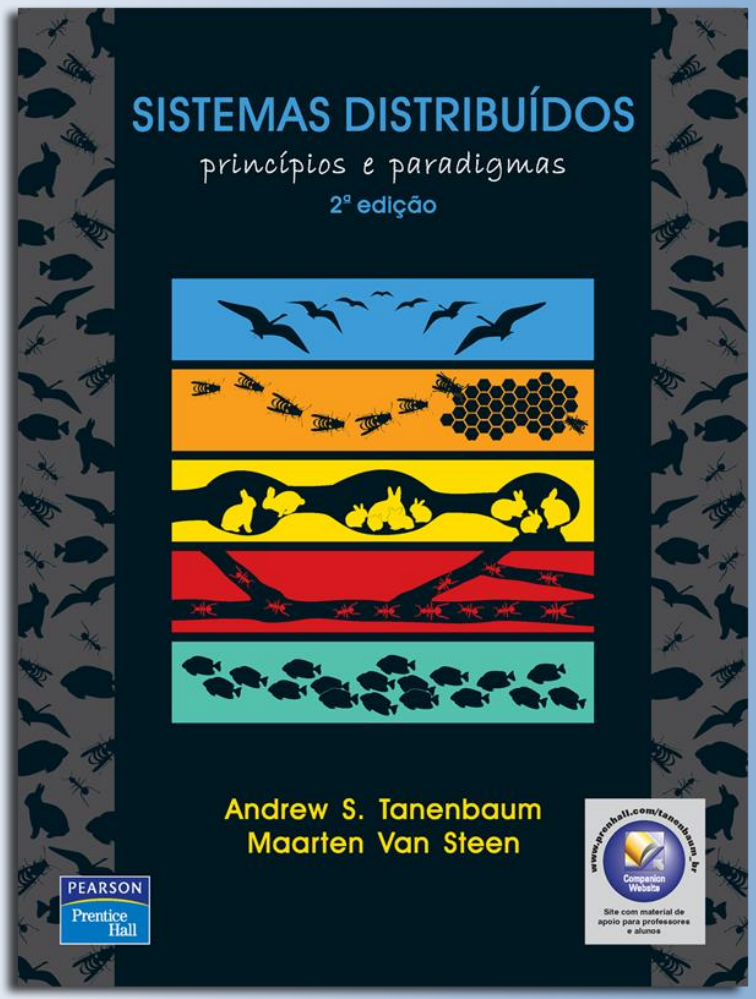
Arquiteturas



capítulo

2

Agenda

- Introdução
- Estilos Arquitetônicos
- Arquiteturas de Sistemas
- Arquiteturas x Middleware
- Autogerenciamento em SDs

Introdução

- SDs são complexas peças de software cujos componentes estão espalhados por várias máquinas.
- Arquitetura de software nos diz como vários componentes de software devem ser organizados e como devem interagir.
- A especificação final de uma arquitetura de software é denominada arquitetura de sistema.

Introdução

- Arquiteturas Centralizadas Tradicionais
  - Há um único servidor que implementa a maioria dos componentes de software, enquanto clientes remotos o acessam usando comunicação simples.
- Arquiteturas Descentralizadas
  - Máquinas desempenham papéis mais ou menos iguais.
- Arquiteturas Híbridas

Introdução

- Meta importante dos SDs
  - Separar aplicações das plataformas subjacentes
- Middleware
  - Adotar tal camada é uma decisão arquitetônica importante
  - Principal finalidade proporcionar transparência de distribuição
  - Necessidade de middleware adaptativo para fins de transparência.

2.1 Estilos Arquitetônicos

- **Componente** é uma unidade modular com interfaces requeridas e fornecidas bem definidas que é substituível dentro de seu ambiente.

2.1 Estilos Arquitetônicos

- **Componente** é uma unidade modular com interfaces requeridas e fornecidas bem definidas que é substituível dentro de seu ambiente.
- **Estilo Arquitetônico** é formulado em termos de componentes
  - do modo como esses componentes estão conectados uns aos outros,
  - dos dados trocados entre componentes.
  - da maneira como esses elementos são configurados em conjunto para formar um sistema.

2.1 Estilos Arquitetônicos

- **Componente** é uma unidade modular com interfaces requeridas e fornecidas bem definidas que é substituível dentro de seu ambiente.
- **Estilo Arquitetônico** é formulado em termos de componentes
  - do modo como esses componentes estão conectados uns aos outros,
  - dos dados trocados entre componentes.
  - da maneira como esses elementos são configurados em conjunto para formar um sistema.
- **Conector** é um mecanismo mediador da comunicação ou da cooperação entre componentes.
  - RPC, Passagem de Mensagens, etc.

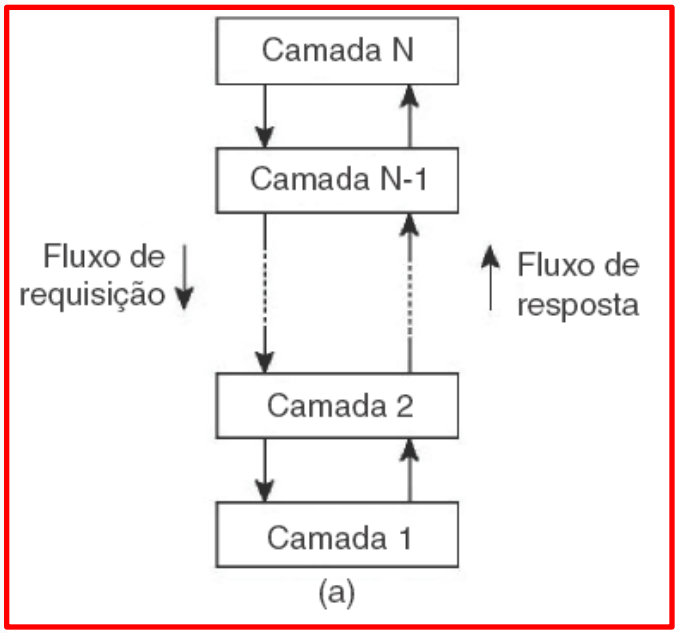
Estilos Arquitetônicos

- Usando componentes e conectores, podemos chegar a várias configurações, que foram classificadas em estilos arquitetônicos.
  - Arquitetura em camadas
  - Arquitetura baseada em objetos
  - Arquitetura centrada em dados
  - Arquitetura baseada em eventos

2.1 Estilos Arquitetônicos

1. Arquitetura em camadas
   - Componentes organizados em camadas, onde componentes da camada L_i pode chamar métodos da camada L_{i-1}, mas não o contrário;
   - Modelo amplamente adotado pela comunidade de redes
   - O controle flui de camada para camada
     - As requisições descem pela hierarquia
     - Os resultados fluem para cima

2.1 Estilos Arquitetônicos (a) em camadas e (b) baseado em objetos



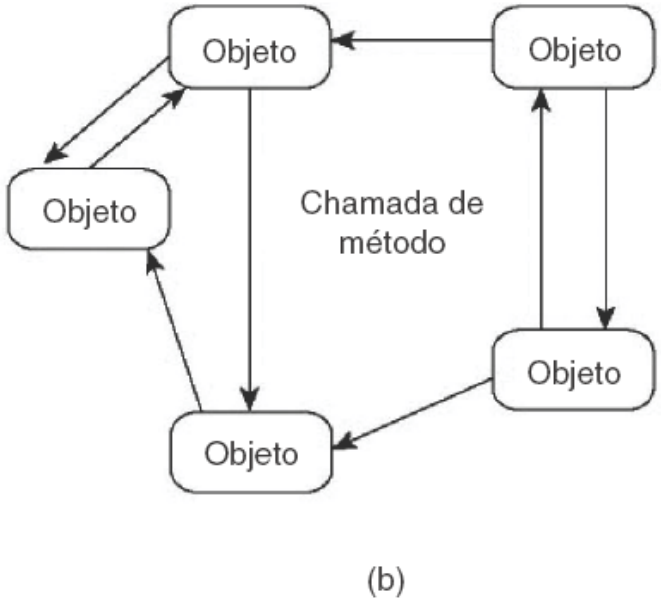
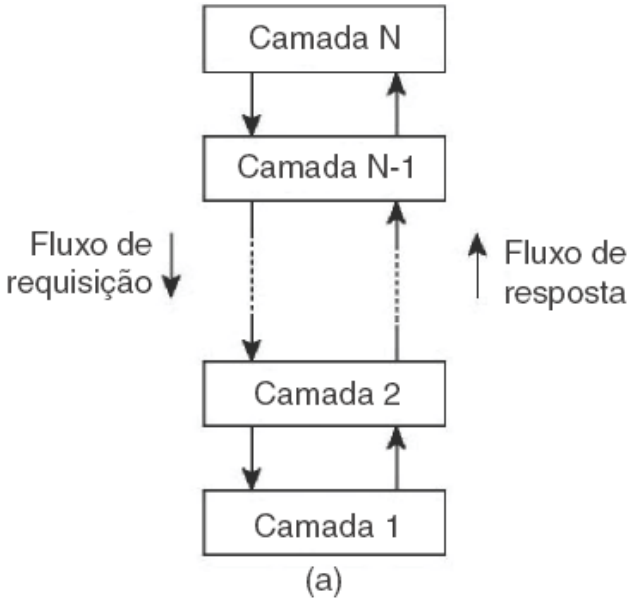


Figura 2.1 Estilo arquitetônico (a) em camadas e (b) baseado em objetos.

2.1 Estilos Arquitetônicos

1. Arquiteturas baseadas em objetos
   - Uma **classe** é uma estrutura que abstrai um conjunto de objetos com características similares.
   - Um **objeto** é uma instância, ou seja, um exemplar de uma classe.
   - Objetos correspondem às definições de componentes, que são conectados por meio de chamadas de procedimento remotas;

2.1 Estilos Arquitetônicos (a) em camadas e (b) baseado em objetos



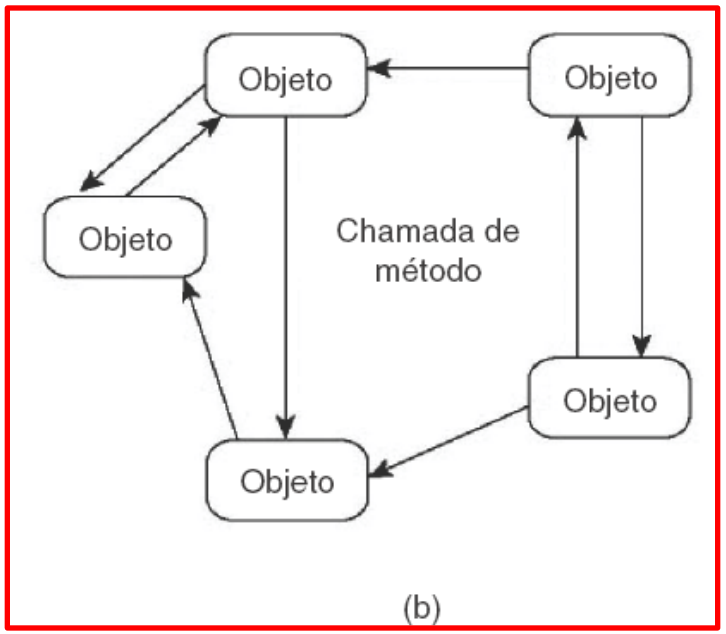


Figura 2.1 Estilo arquitetônico (a) em camadas e (b) baseado em objetos.

2.1 Estilos Arquitetônicos

3. Arquiteturas centradas em dados
   - Processos se comunicam por meio de repositório comum.
   - Praticamente tudo no GNU/Linux é arquivo.
   - Sistemas Distribuídos na Web são em sua maioria centrados em dados
     - Processos se comunicação por meio da troca e uso desses dados, e não diretamente.

2.1 Estilos Arquitetônicos (a) baseados em eventos e (b) centradas em dados

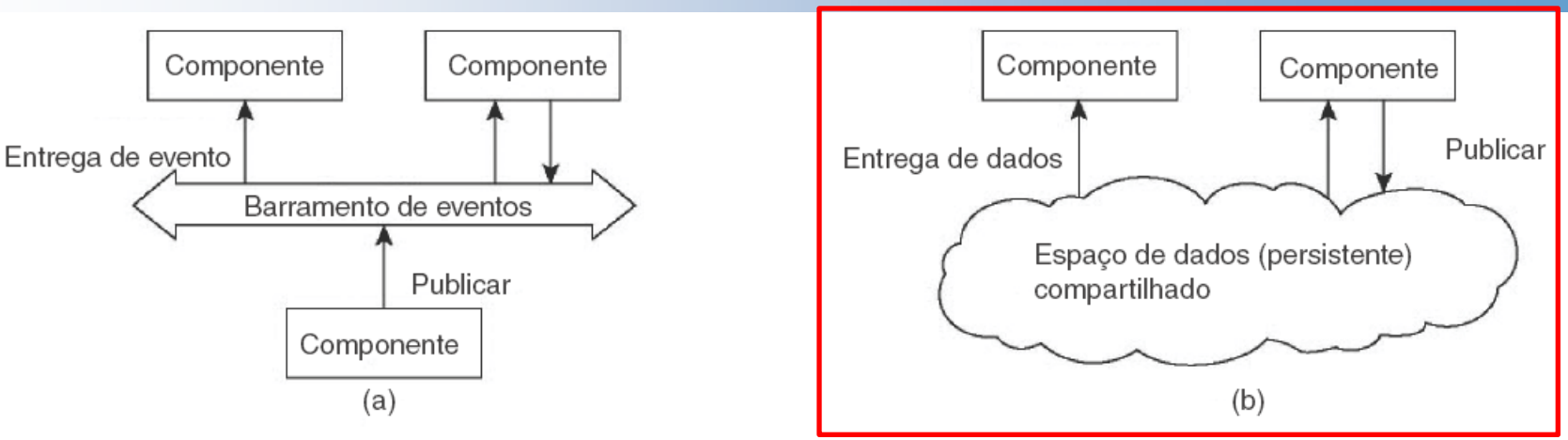
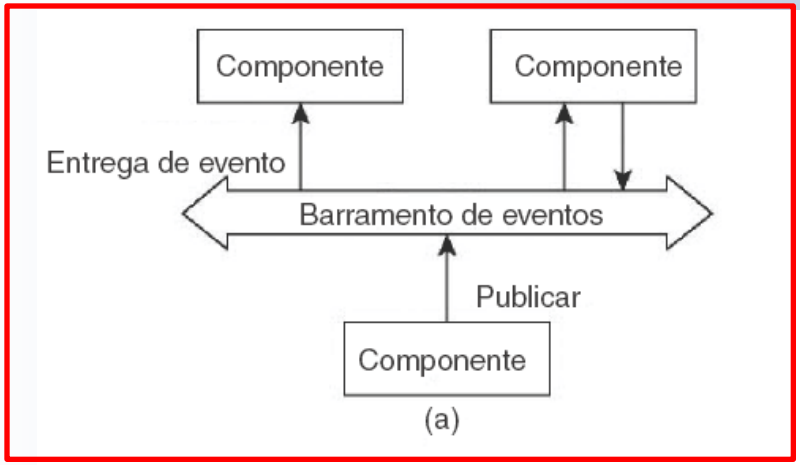


Figura 2.2 Estilo arquitetônico (a) baseado em eventos e (b) de espaço de dados compartilhado.

2.1 Estilos Arquitetônicos

3. Arquiteturas baseadas em eventos
   - Processos se comunicam por meio de propagação de eventos
     - Opcionalmente podem transportar dados;
   - Sistemas **Publicar/Subscrever**;
     - Processos publicam eventos, e o middleware repassa-os somente para os processos que subscreveram esses eventos.
   - São referencialmente desacoplados ou seja, fracamente acoplados.

2.1 Estilos Arquitetônicos (a) baseados em eventos e (b) centradas em dados



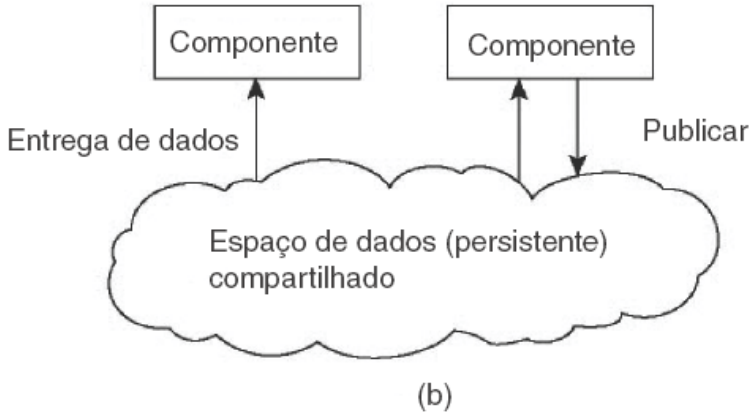


Figura 2.2 Estilo arquitetônico (a) baseado em eventos e (b) de espaço de dados compartilhado.

2.1 Estilos Arquitetônicos

- Estilos podem ser híbridos, como arquiteturas baseadas em eventos juntamente com centradas em dados
- Nesse caso são conhecidas como espaços compartilhados de dados.
  - Processos são desacoplados no tempo, ou seja, não precisam estar ambos ativos quando ocorrer a comunicação.

2.2 Arquiteturas de Sistemas

2.2.1 Arquiteturas Centralizadas

- Modelo Cliente-Servidor
  - Pensar em termos de cliente que requisitam serviços de servidores nos ajudam a entender e gerenciar a complexidade dos SDs.
  - Processos são divididos em 2 grupos

2.2 Arquiteturas de Sistemas

- Modelo Cliente-Servidor
  - Processo Servidor
    - Implementa o um serviço específico.
    - Ex: Serviço de Sistema de Arquivos, Serviço de BD.
  - Processo Cliente
    - Requisita um serviço de um servidor
    - Envia um requisição e espera uma resposta.

2.2 Arquiteturas de Sistemas

- Modelo Cliente-Servidor

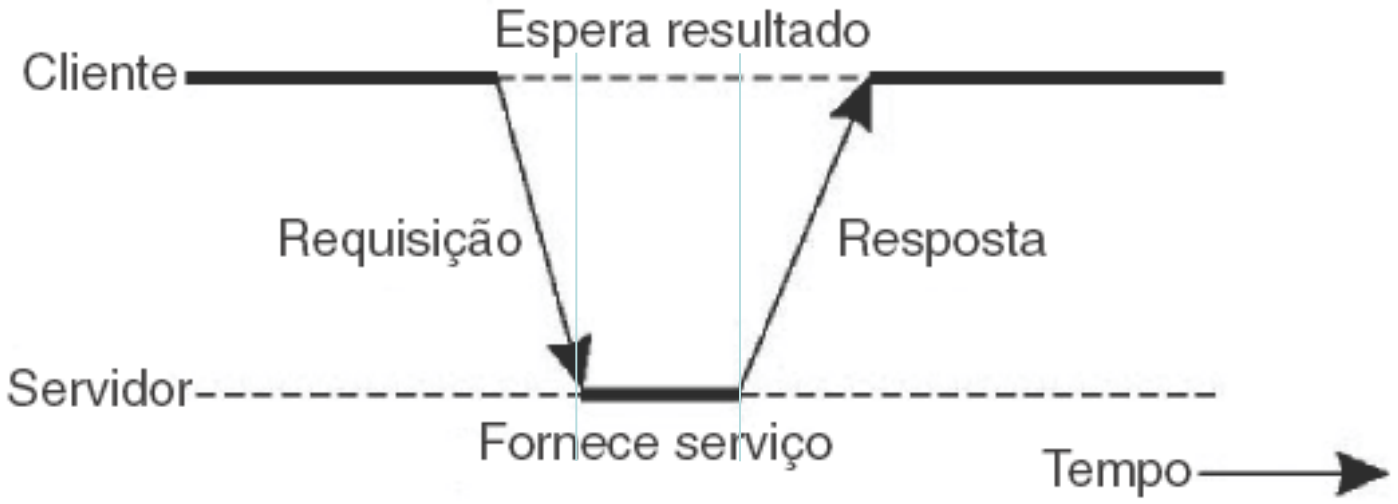


Figura 2.3 Interação geral entre um cliente e um servidor.

2.2 Arquiteturas de Sistemas

- Modelo Cliente-Servidor
  - Orientado ou não-orientado a conexão (TCP ou UDP)?
    - Protocolos sem conexão são mais eficientes.
    - Protocolos com conexão são mais seguros contra falhas.
  - Reenvio de requisições há outras implicações além do tempo?
  - O que falhou? A requisição ou resposta?

2.2 Arquiteturas de Sistemas

- Modelo Cliente-Servidor
  - Se a resposta se perdeu, reenviar uma requisição pode resultar em executar a operação 2 vezes.
    - E se a operação for algo do tipo, transfira 10 mil da minha conta?
    - E se a operação for algo do tipo, informe quanto dinheiro tenho?
  - Quando uma operação pode ser repetida várias vezes sem causar danos, diz-se que ela é idempotente.

2.2 Arquiteturas de Sistemas

- Modelo Cliente-Servidor
- Não orientado a conexão
  - Rede subjacente razoavelmente confiável, como em redes locais
- Orientado a conexão
  - Desempenho lento não indicado em rede local.
  - Solução indicada em sistemas de longa distância, onde a comunicação é inerentemente não confiável.

Questão

- 1. Se um cliente e um servidor são colocados longe um do outro, podemos ver a latência da rede dominando o desempenho global. Como podemos resolver este problema?

- 1. Isso realmente depende de como o cliente está organizado. Pode ser possível dividir o código do lado cliente (client-side) em pequenas partes, que podem ser executadas separadamente. Nesse caso, quando uma dessas partes está aguardando a resposta do servidor, podemos usar outra parte. Alternativamente, podemos reorganizar o cliente para que ele possa fazer outros trabalhos, depois de ter enviado uma solicitação ao servidor. Esta última solução eficaz substitui a comunicação cliente-servidor sincronizada por uma forma de comunicação assíncrona.

2.2 Arquiteturas de Sistemas

- Modelo Cliente-Servidor
- O que é realmente um cliente e um servidor?
  - Ex: Um servidor de BD Distribuído pode agir como cliente um cliente porque está repassando requisições para diferentes servidores de arquivos.
- Considerando que muitas aplicações cliente-servidor, visam dar suporta ao acesso do usuário a banco de dados, criou-se o seguinte estilo arquitetônico em 3 camadas.

2.2.1 Arquiteturas Centralizadas

Camadas de Aplicação

- Divididas em 3 níveis:
  1. Nível de interface de usuário
     - Contém tudo o que é necessário para fazer interface com o usuário
  2. Nível de processamento
     - Contém as aplicações
  3. Nível de dados
     - Gerencia os dados propriamente ditos

Organização em 3 Camadas de Aplicação

- Um mecanismo de busca da Internet

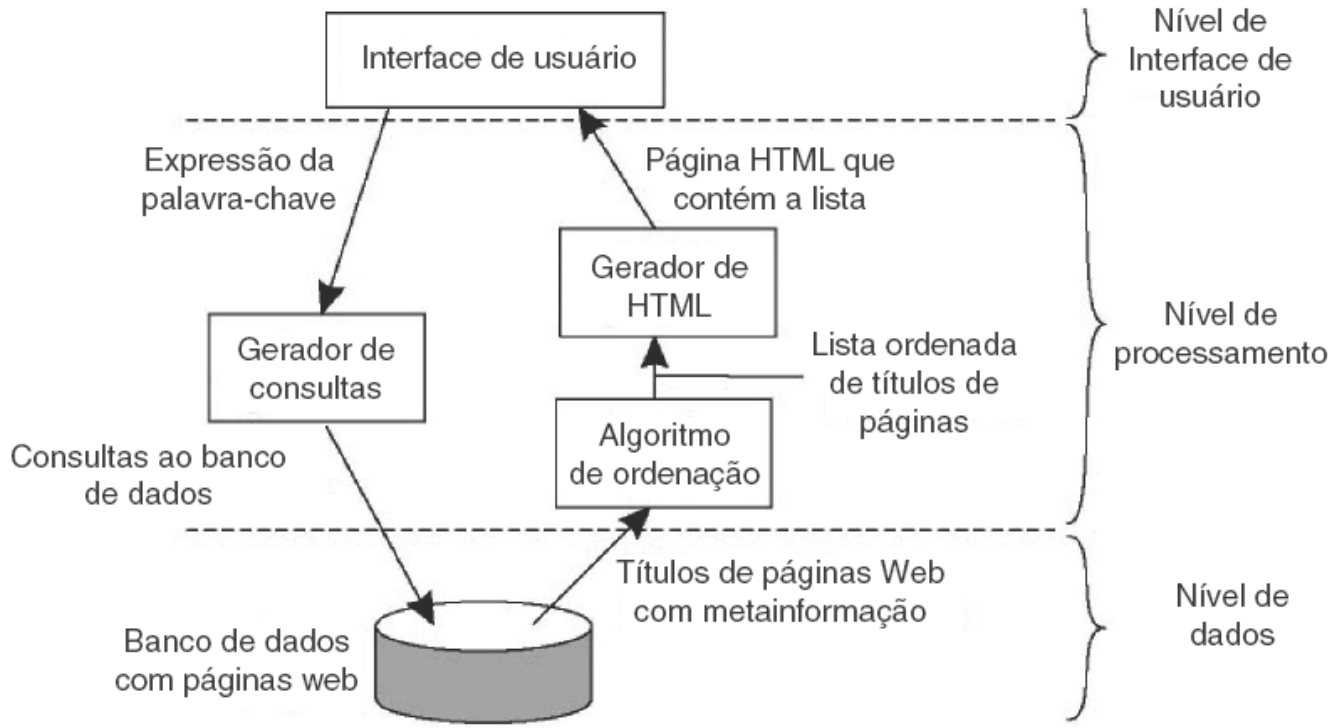


Figura 2.4 Organização simplificada de um mecanismo de busca da Internet em três camadas diferentes.

Questão

- 2. O que é uma arquitetura cliente-servidor de três níveis?

- 2. Uma arquitetura cliente-servidor de três níveis é constituída por três camadas lógicas, onde cada camada é, em princípio, implementada em uma máquina separada. A camada superior consiste de uma interface de utilização do cliente, a camada do meio contém a aplicação real (regras de negócio) e, a camada inferior implementa os dados que estão sendo utilizados.

2.2.1 Arquiteturas Centralizadas Arquiteturas multidivididas

- Com base na organização de três níveis lógicos discutida anteriormente, é necessária a distribuição física. A maneira mais simples, denominada arquitetura de duas divisões (físicas) é distribuída da seguinte forma:
  - Parte lógica do lado cliente
  - Parte lógica do lado servidor

2.2.1 Arquiteturas Centralizadas
Arquiteturas multidivididas

- Alternativas de distribuição

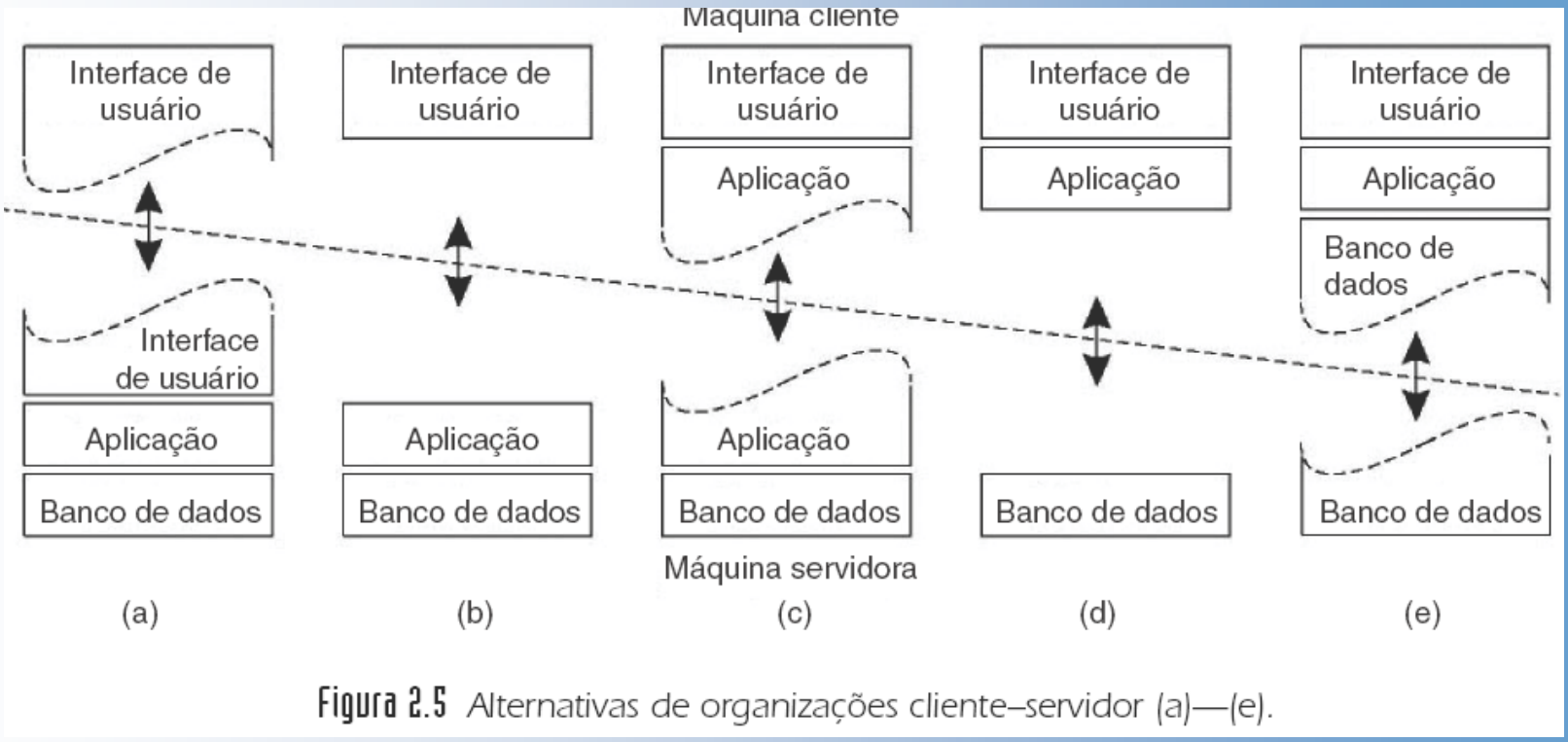


Figura 2.5 Alternativas de organizações cliente–servidor (a)—(e).

2.2.1 Arquiteturas Centralizadas
Arquiteturas multidivididas

- Alternativas de distribuição

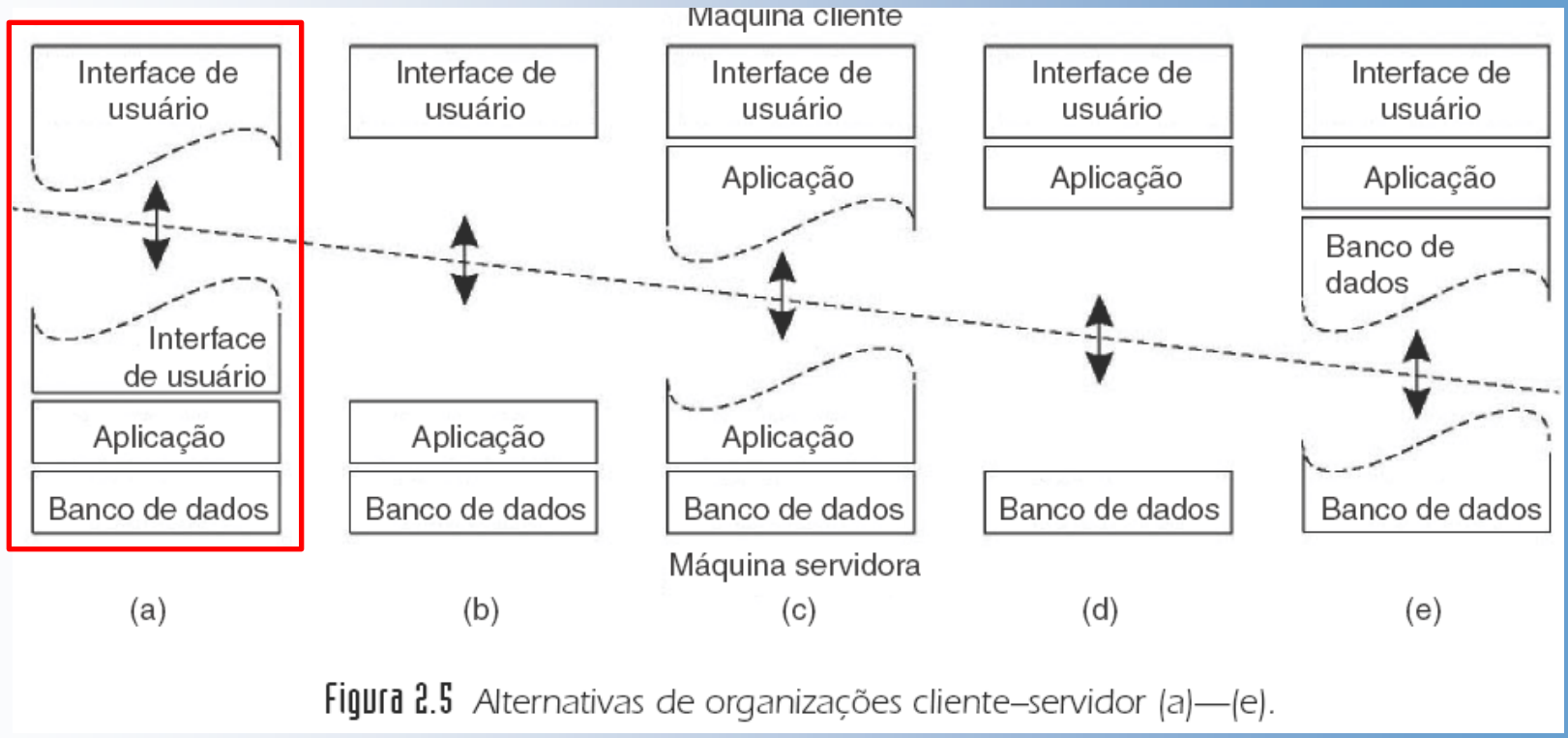


Figura 2.5 Alternativas de organizações cliente–servidor (a)—(e).

Divisão da Interface do Usuário

- Ter na máquina do cliente somente a interface do usuário que é dependente do dispositivo.
- As aplicações tem controle remoto sobre a apresentação dos seus dados.
  - Sistemas de adaptação de conteúdo baseados em contexto

2.2.1 Arquiteturas Centralizadas
Arquiteturas multidivididas

- Alternativas de distribuição

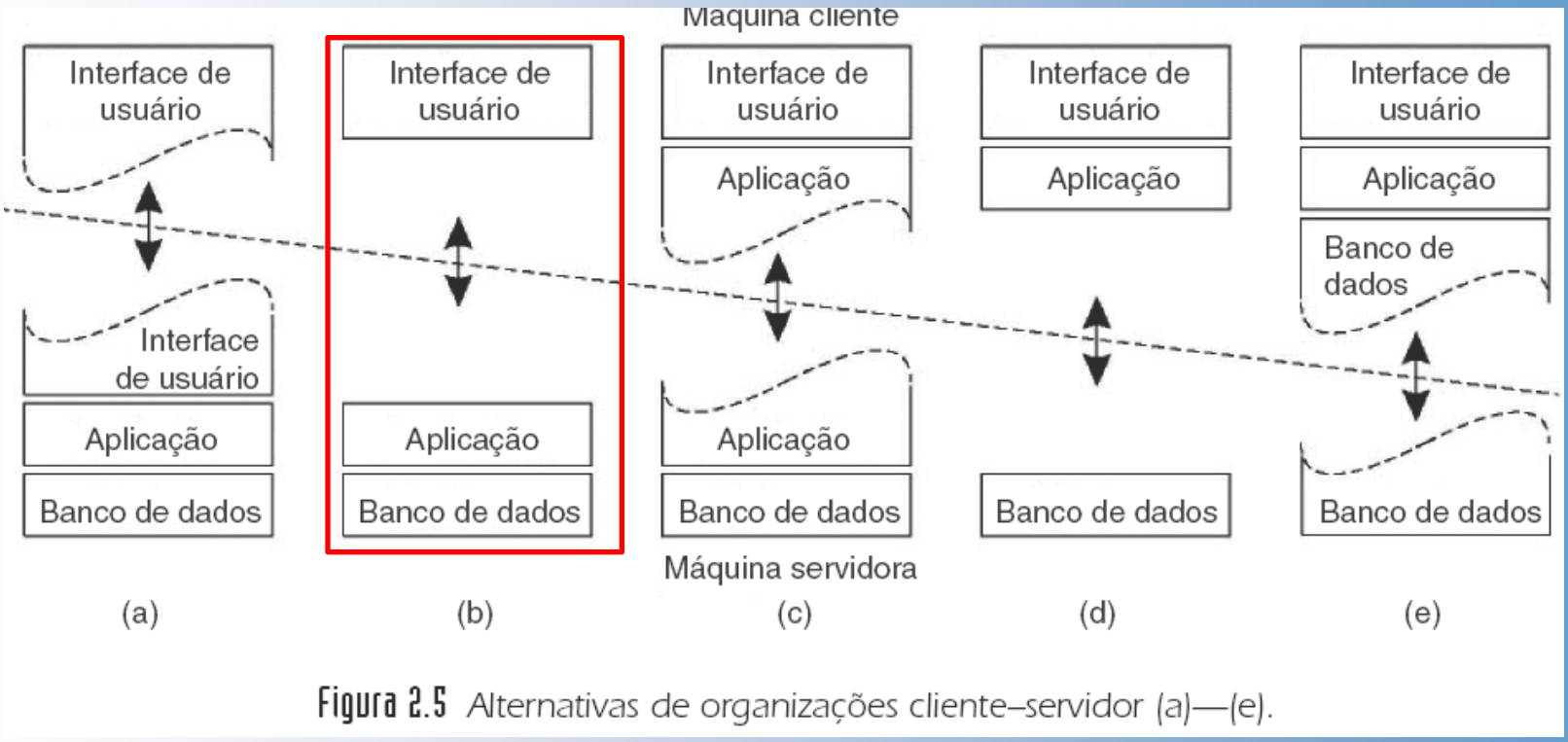


Figura 2.5 Alternativas de organizações cliente–servidor (a)—(e).

Divisão da Extremidade Frontal Gráfica

1. Uma máquina cliente que contém apenas os programas que implementam o nível (ou parte do nível) de interface de usuário.
2. Uma máquina do servidor que contém todo o resto, ou seja, os níveis de processamento e de dados.

2.2.1 Arquiteturas Centralizadas
Arquiteturas multidivididas

- Alternativas de distribuição

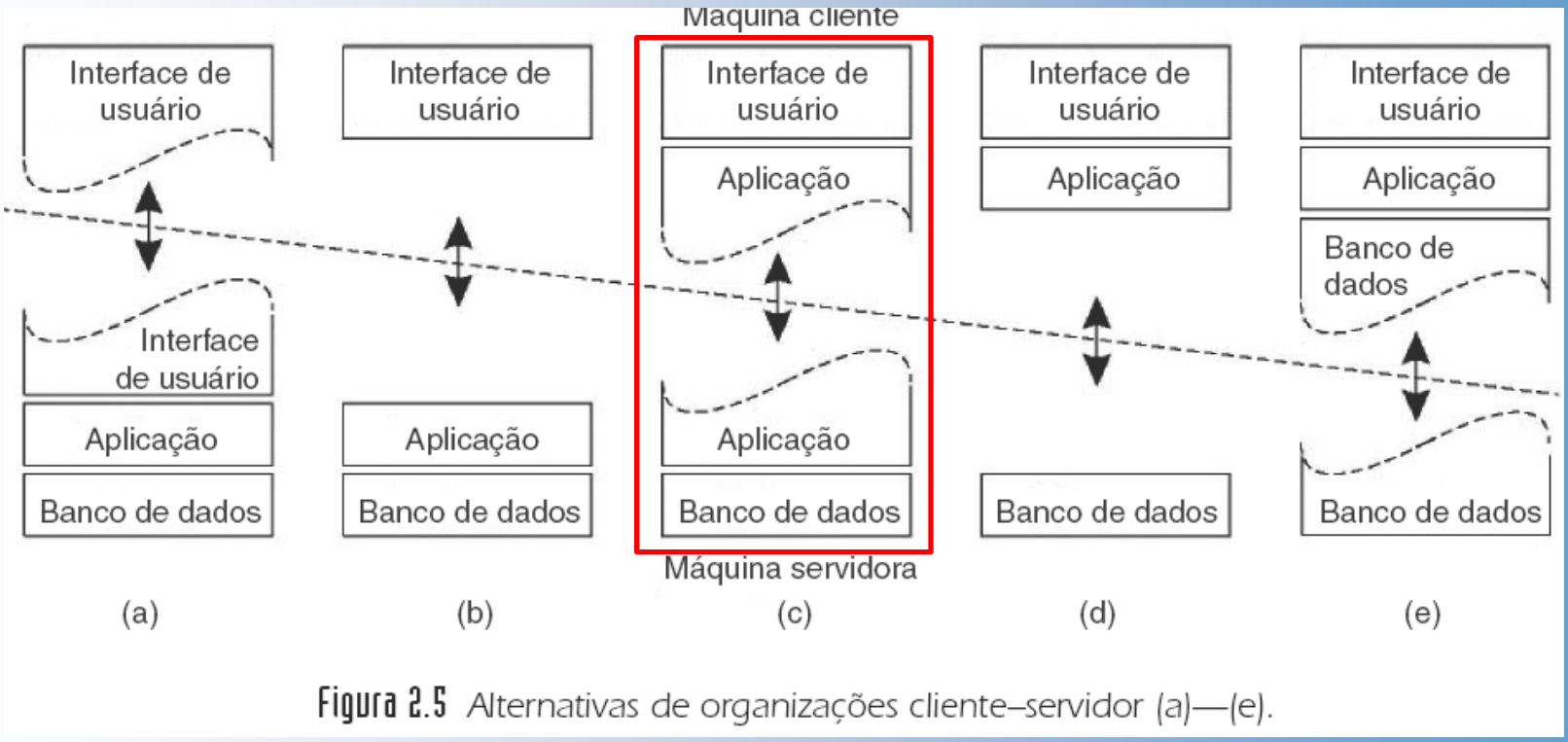


Figura 2.5 Alternativas de organizações cliente–servidor (a)—(e).

Divisão da Aplicação

1. Há parte do processamento no cliente
2. Ex: A verificação da consistência e correção dos dados preenchidos são verificados no cliente, antes de serem processado .

2.2.1 Arquiteturas Centralizadas
Arquiteturas multidivididas

- Alternativas de distribuição

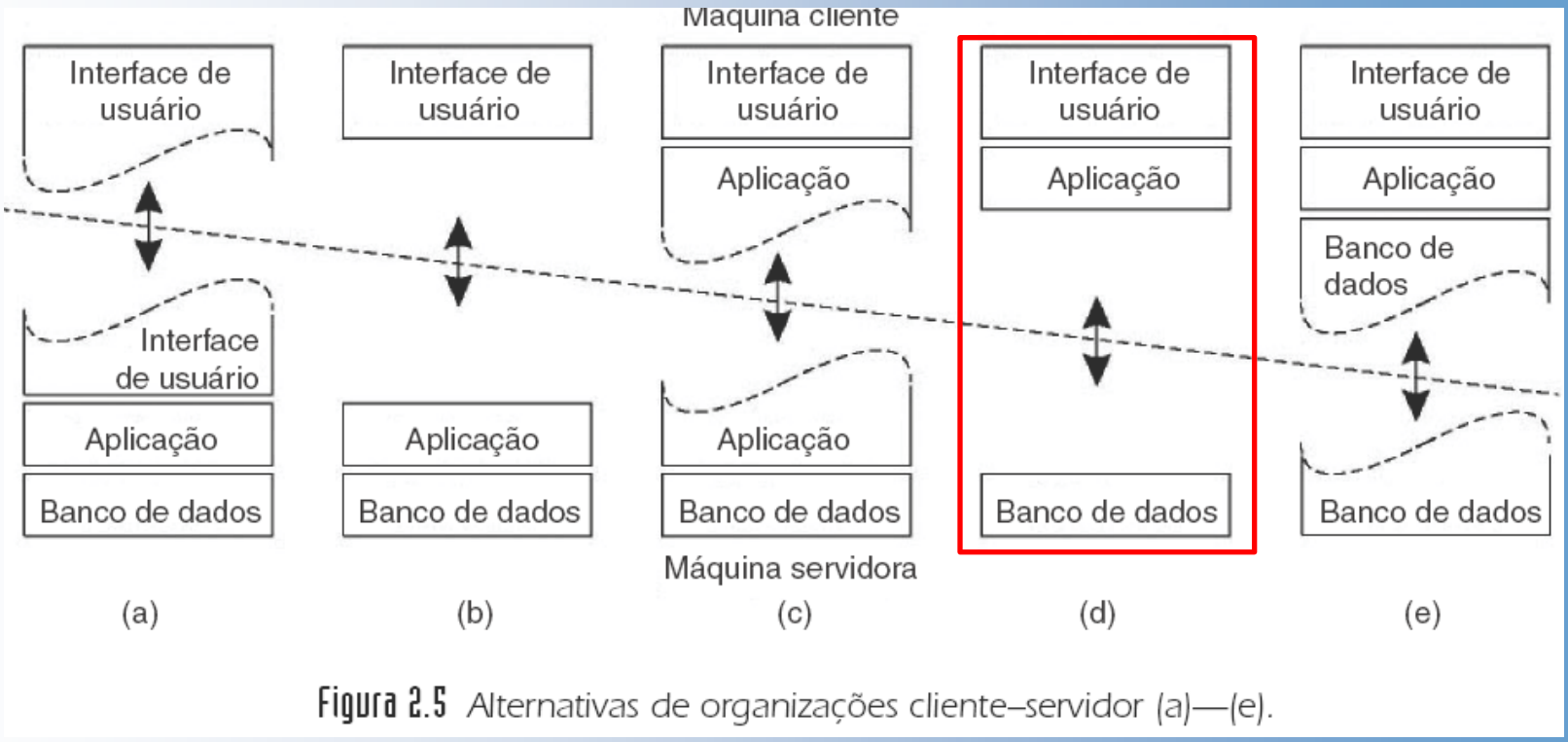


Figura 2.5 Alternativas de organizações cliente–servidor (a)—(e).

Divisão da Extremidade da Aplicação

1. Quando grande parte da aplicação executa na máquina cliente, mas todas as operações com arquivos e entradas em banco de dados vão para o servidor.

2.2.1 Arquiteturas Centralizadas
Arquiteturas multidivididas

- Alternativas de distribuição

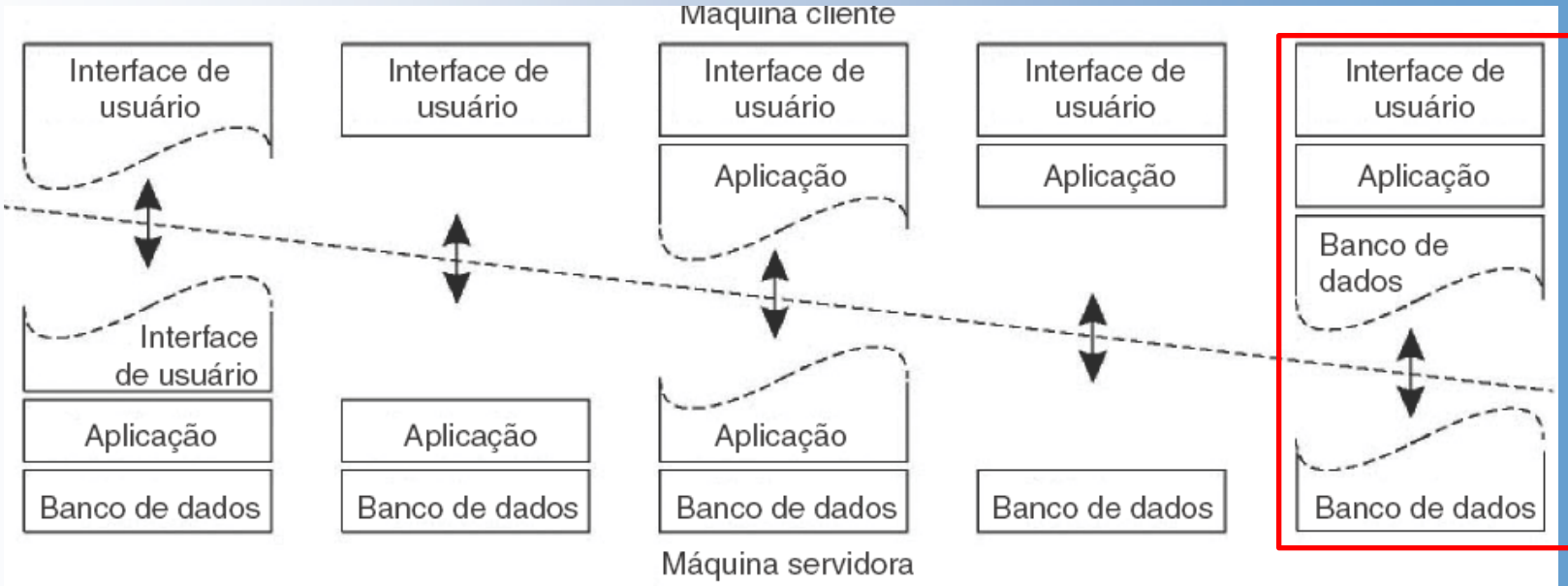


Figura 2.5 Alternativas de organizações cliente–servidor (a)—(e).

Divisão do Banco de Dados

1. Situação em que o disco local do cliente contém parte dos dados.
2. Ao consultar páginas web por exemplo, o cliente pode gradativamente construir uma enorme cache em disco local com páginas web mais recentemente consultadas.

Perpectiva do Gerenciamento de Sistemas

- Clientes gordos (fat clients)
  - 2.5(d) e 2.5(e)
  - Não é ótimo, pois máquinas clientes são difíceis de gerenciar
  - Máquinas clientes são mais problemáticas.
  - Torna o software mais propenso a erros.
  - Mais dependente da plataforma subjacente.
- Clientes magros (thin clients)
  - De 2.5(a) a 2.5(c) são mais fáceis de gerenciar.

2.2.1 Arquiteturas Centralizadas Arquiteturas multidivididas

- Arquitetura de três divisões (físicas)
  - Servidor as vezes pode agir como cliente.

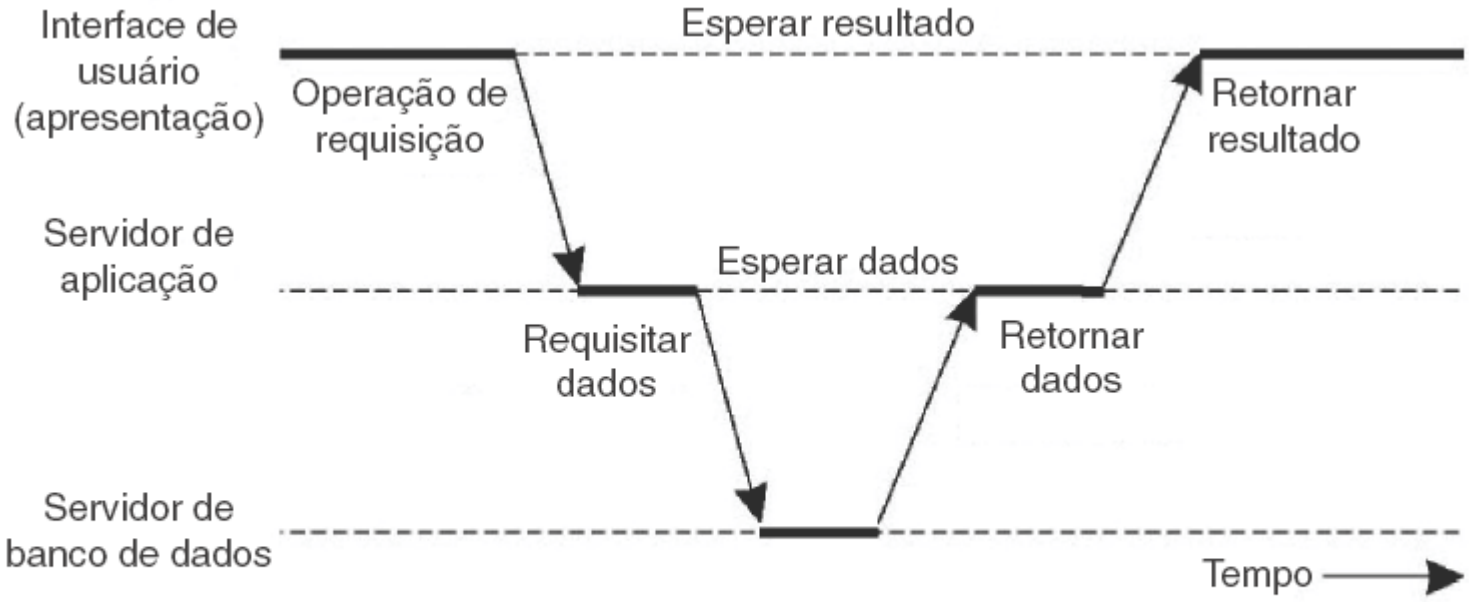


Figura 2.6 Exemplo de um servidor que age como cliente.

Questão

- 4. Considere-se uma cadeia de processos de P1, P2, ..., Pn implementando uma arquitetura cliente-servidor multicamada. O processo Pi é cliente de um processo Pi+1, e Pi retornará uma resposta para Pi-1, somente após receber uma resposta de Pi+1. Quais são os principais problemas com esta organização quando se examina o desempenho de requisição-resposta no processo P1?

- 4. Espera-se um desempenho ruim para um n grande. O problema é que cada comunicação entre duas camadas sucessivas é, em princípio, entre duas máquinas diferentes. Outro problema é que, se uma máquina na cadeia funciona mal ou mesmo está temporariamente inacessível, então esta irá imediatamente degradar o desempenho num nível mais alto.

2.2.2 Arquiteturas descentralizadas

- Distribuição vertical
  - divide componentes *logicamente* diferentes em máquinas diferentes;
- Distribuição horizontal
  - Um cliente ou servidor pode ser subdividido em partes logicamente equivalentes, mas cada parte está operando em sua própria porção do conjunto de dados, equilibrando a carga.
  - Ex.: **Peer to Peer**
    - Grande parte da interação entre processos é simétrica
    - servidor e cliente ao mesmo tempo, também chamada "servente"

Questão

- 3. Qual é a diferença entre uma distribuição vertical e uma distribuição horizontal?

- 3. A distribuição vertical refere-se à distribuição das diferentes camadas de uma arquitetura multicamadas através de múltiplas máquinas. Em princípio, cada camada é implementada em uma máquina diferente. A distribuição horizontal lida com a distribuição de uma única camada através de múltiplas máquinas, como a distribuição de um único banco de dados.

2.2.2 Arquiteturas descentralizadas Peer to Peer

- Devido ao comportamento simétrico arquiteturas P2P se dividem pela forma de organizar os processos em uma rede de sobreposição.
  - Rede no qual os nós são formados pelos processo e os enlaces representam os canais de comunicação possíveis
  - Usualmente são usadas conexões TCP/IP.
  - Processos não se comunicação diretamente e sim por meio dos canais de comunicação disponíveis.

2.2.2 Arquiteturas descentralizadas Peer to Peer

- Redes de sobreposição estruturadas
  - Rede de sobreposição é construída com a utilização de um procedimento determinístico.
- Redes de sobreposição não-estruturadas
  - Dependem de algoritmos aleatórios para a construção da rede de sobreposição

2.2.2 Arquiteturas descentralizadas Arquiteturas Peer to Peer estruturadas

- Para criação de arquiteturas P2P estruturadas normalmente organiza-se os processos por meio de uma DHT (Distributed Hash Table).
  - Itens de Dados e Nós recebem uma chave aleatória de 128 ou 160 bits como identificador.
  - Ponto crucial de sistemas baseados em DHT é mapear exclusivamente a chave de um Item de Dado para um Identificador de Nó, tendo somente alguma distância métrica.

2.2.2 Arquiteturas descentralizadas Arquiteturas Peer to Peer estruturadas

- Sistema Chord – Nós ligados logicamente em anel, onde um Item de Dado com chave k é mapeado para o Nó que tenha o menor Id ≥ k.
  - Esse nó é denominado sucessor da chave k e denotado como succ(k).
  - Uma aplicação executando em um nó arbitrário, para consultar o item de dado teria de chamar LOOKUP(K), que retornaria o endereço de rede de succ(k).

2.2.2 Arquiteturas descentralizadas Sistema Chord

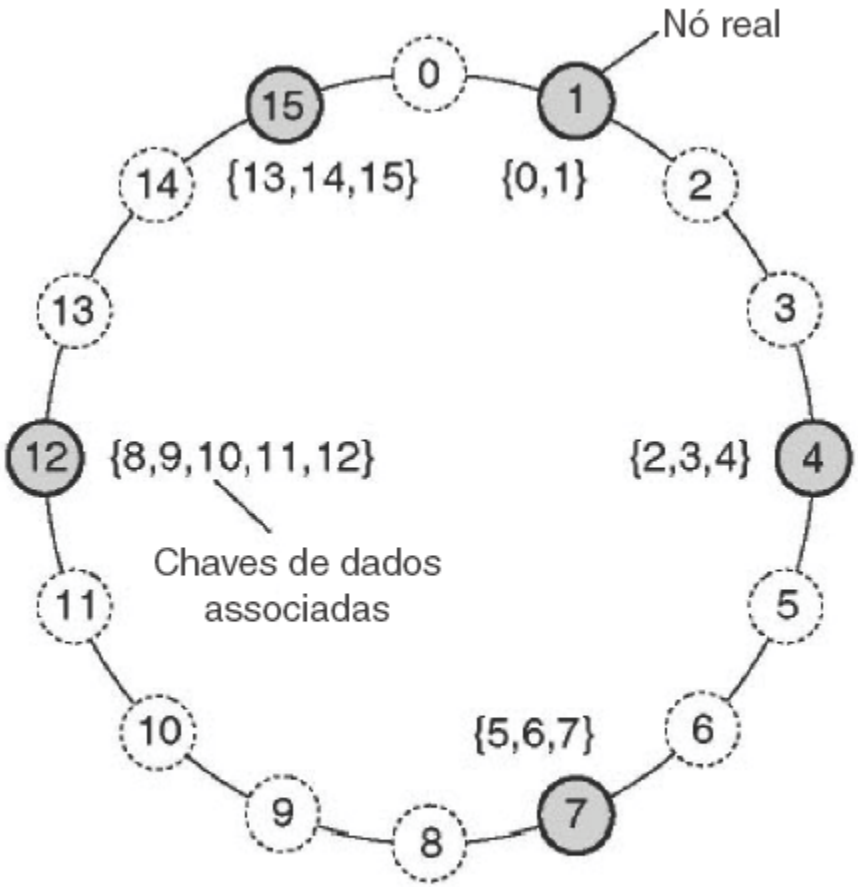


Figura 2.7 Mapeamento de itens de dados para nós em Chord.

2.2.2 Arquiteturas descentralizadas Arquiteturas Peer to Peer estruturadas

- Novo Nó no Sistema Chord
  - Gera id aleatório (com um bom gerador aleatório a chance de gerar um id em uso é quase zero).
  - Nó faz pesquisa por id, retornando succ(id).
  - Nó contacta succ(id), seu predecessor e se insere no anel.
  - Cada item de dados em succ(id) que está associado ao id deve ser transferido para o novo nó.

2.2.2 Arquiteturas descentralizadas Arquiteturas Peer to Peer estruturadas

- Saída do Nó no Sistema Chord
  - Nó informa sua partida ao seu predecessor e sucessor
  - Transfere seus itens de dados para succ(id).

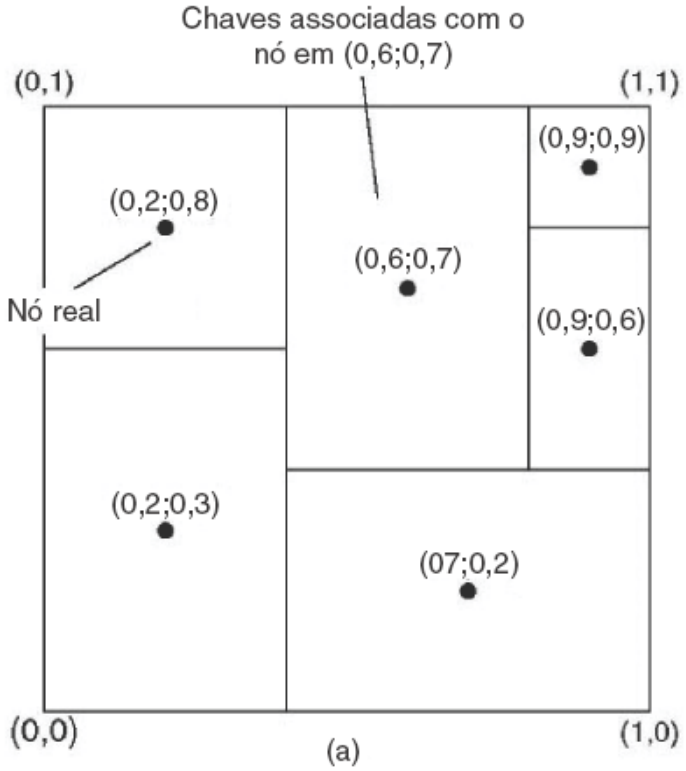
2.2.2 Arquiteturas descentralizadas Arquiteturas Peer to Peer estruturadas

- CAN (Content Addressable Network) – Usa espaço de coordenadas cartesianas de *d* dimensões particionado entre todos os nós participantes do sistema.

2.2.2 Arquiteturas descentralizadas Arquiteturas Peer to Peer estruturadas

- A figura a seguir mostra como um espaço bidimensional de [0,1] x [0,1] é dividido entre 6 nós.
  - Cada nó tem uma região associada
  - Todo Item de Dados no CAN será atribuído um único ponto desse espaço.

2.2.2 Arquiteturas descentralizadas
Rede de Conteúdo Endereçável (CAN)



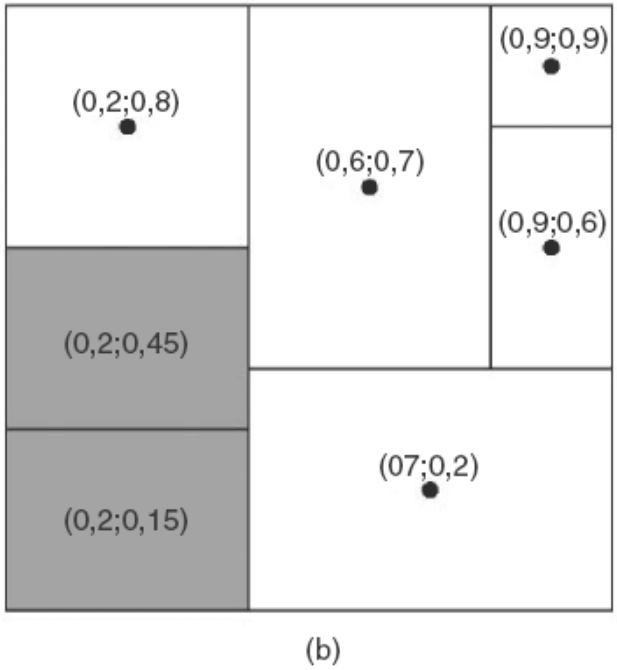


Figura 2.8 (a) Mapeamento de itens de dados para nós em CAN. (b) Subdivisão de uma região quando um nó se junta ao grupo.

2.2.2 Arquiteturas descentralizadas Arquiteturas Peer to Peer estruturadas

- Novo nó no CAN
  - Nó P escolhe um ponto arbitário em [0,1] x [0,1]
  - Em seguida, pesquisa o nó Q, em cuja região o ponto cai
  - O Nó Q subdivide sua região em 2 metades.
  - Uma metade é designada ao nó P
  - P questina a Q quem são seus vizinhos
    - Os nós monitoram os seus vizinhos

2.2.2 Arquiteturas descentralizadas Arquiteturas Peer to Peer estruturadas

- Sair do CAN
  - Pouco mais complicado
  - Considere a saída do nó (0.6, 0.7)
    - Difícil fundi-la a outra região e obter um retângulo
  - Leva a partições menos simétricas do espaço de coordenadas

Questão

- 5. Em uma rede de sobreposição estruturada, as mensagens são encaminhadas de acordo com a topologia da sobreposição. Cite uma importante desvantagem desta abordagem?

- 5. O problema é que estamos lidando apenas com caminhos lógicos. Pode muito bem ser o caso de que dois nós A e B, que são vizinhos na sobreposição rede, serem colocados fisicamente distantes. Como consequência, o caminho logicamente curto entre A e B podem requerer encaminhamento de uma mensagem por um caminho muito longo na rede física subjacente.

Questão

- 6. Considere a rede CAN da figura 2.8. Como você rotearia uma mensagem do nó de coordenadas (0.2,0.3), para o nó que tem as coordenadas (0.9,0.6)?

- 6. Existem várias possibilidades, mas se queremos o caminho mais curto de acordo com uma distância euclidiana, devemos seguir a rota (0.2,0.3) -> (0.6,0.7) -> (0.9,0.6), que tem uma distância de 0,882. A via alternativa (0.2,0.3) -> (0.7,0.2) -> (0.9,0.6) tem uma distância de 0,957.

Questão

- 7. Considerando-se que um nó em CAN conheça as coordenadas de seus vizinhos imediatos, uma política de roteamento razoável seria a de transmitir uma mensagem ao nó mais próximo em direção ao destino. Quão boa é esta política?

- 7. No nosso exemplo da questão anterior, pode ser visto que não precisa-se lidar com a melhor rota. Se nó (0.2,0.3) segue esta política para o mensagem destinada para o nó (0.9,0.6), seria enviá-lo para o nó (0.7,0.2).

2.2.2 Arquiteturas descentralizadas Arquiteturas P2P não-estruturadas

- Cada nó mantém uma lista de vizinhos construída de forma aleatória.
- Itens de dados são colocados aleatoriamente nos nós.
  - Consulta através de inundação da rede
- A meta é construir uma rede de sobreposição parecida com um grafo aleatório.

2.2.2 Arquiteturas descentralizadas
Arquiteturas P2P não-estruturadas

- Cada nó mantém uma lista de c vizinhos, onde cada nó representa um nó vivo escolhido aleatoriamente no momento.
- A lista de vizinhos é também chamada de visão parcial.
- Para construção dessas visões os nós trocam regularmente suas visões parciais, associado a uma idade da informação.
- Deve-se descartar o maior número possível de entradas velhas.

2.2.2 Arquiteturas descentralizadas Arquiteturas P2P não-estruturadas

- Para um nó se juntar ao grupo
  - Contacta um outro nó arbitrário, de uma lista de pontos de acesso bem conhecidos.
  - Esse pontos de acesso são membros comuns, exceto que podemos considerar que ele tem alta disponibilidade.

2.2.2 Arquiteturas descentralizadas Arquiteturas P2P não-estruturadas

- Para abandonar a rede
  - Como os nós trocam visões parciais frequentemente, basta sair sem informar nada.
  - Se um nó contactar um nó que não está respondendo, ele vai apenas remover a entrada na sua visão parcial e contactar outro nó.
  - Se o nó for muito popular pode-se causar um desequilíbrio da carga de trabalho, pois há entrada para esse nó em muitas visões parciais.

Gerenciamento de topologia de redes de sobreposição

- Pode parecer que sistemas P2P estruturados e não estruturados formam classes independentes, podem ser combinados.
- Abordagem de 2 camadas
  - Sobreposição aleatória mantém um gráfico aleatório preciso dos nós vivos.
  - Sobreposição estruturada utilizada da camada inferior que seleciona vizinhos segundo uma topologia desejada.

Gerenciamento de topologia de redes de sobreposição

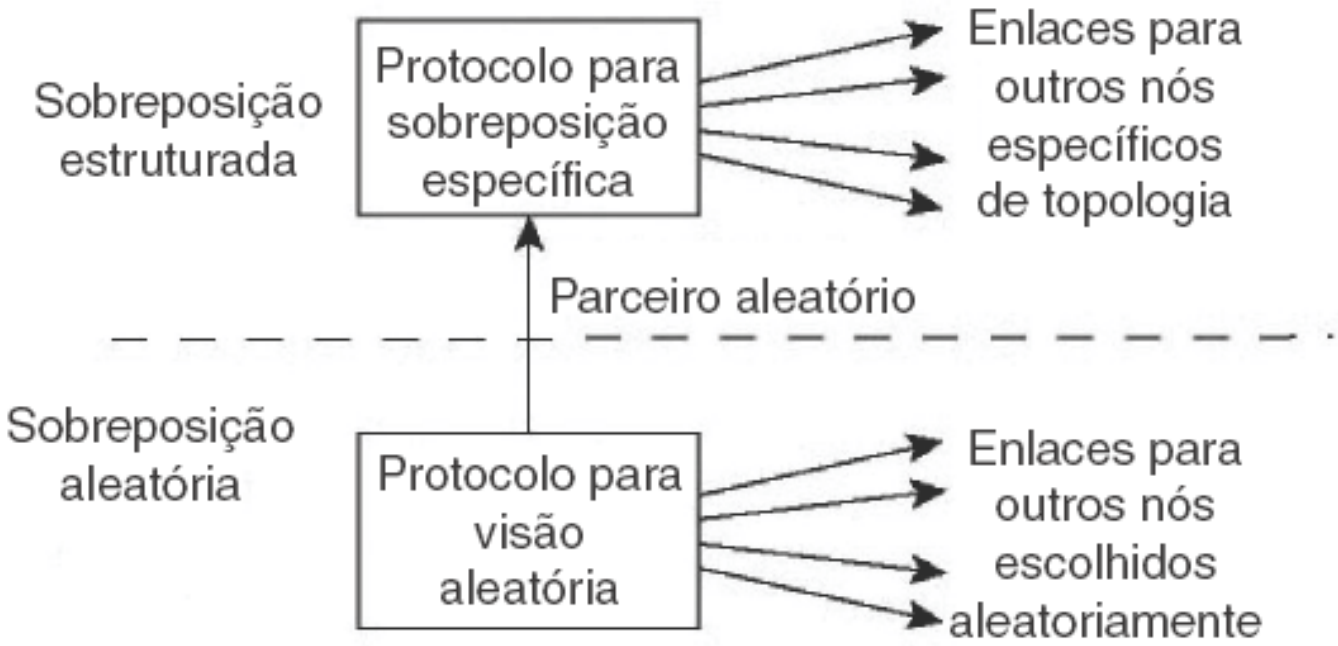


Figura 2.10 Abordagem de duas camadas para construir e manter topologias específicas de sobreposição usando técnicas de sistemas peer-to-peer não estruturadas.

Super-pares (superpeers)

- Usam nós especiais que mantêm um índice de itens de dados, a fim de facilitar a localização de dados relevantes em sistemas P2P não estruturados.
  - Buscados por inundação
- Definem, em muitos casos, uma relação cliente-superpar fixa, onde sempre que um par se junta à rede, se liga a um dos superpares e continua ligado até sair da rede
  - Nem sempre é uma boa solução!

Super-pares (superpeers)

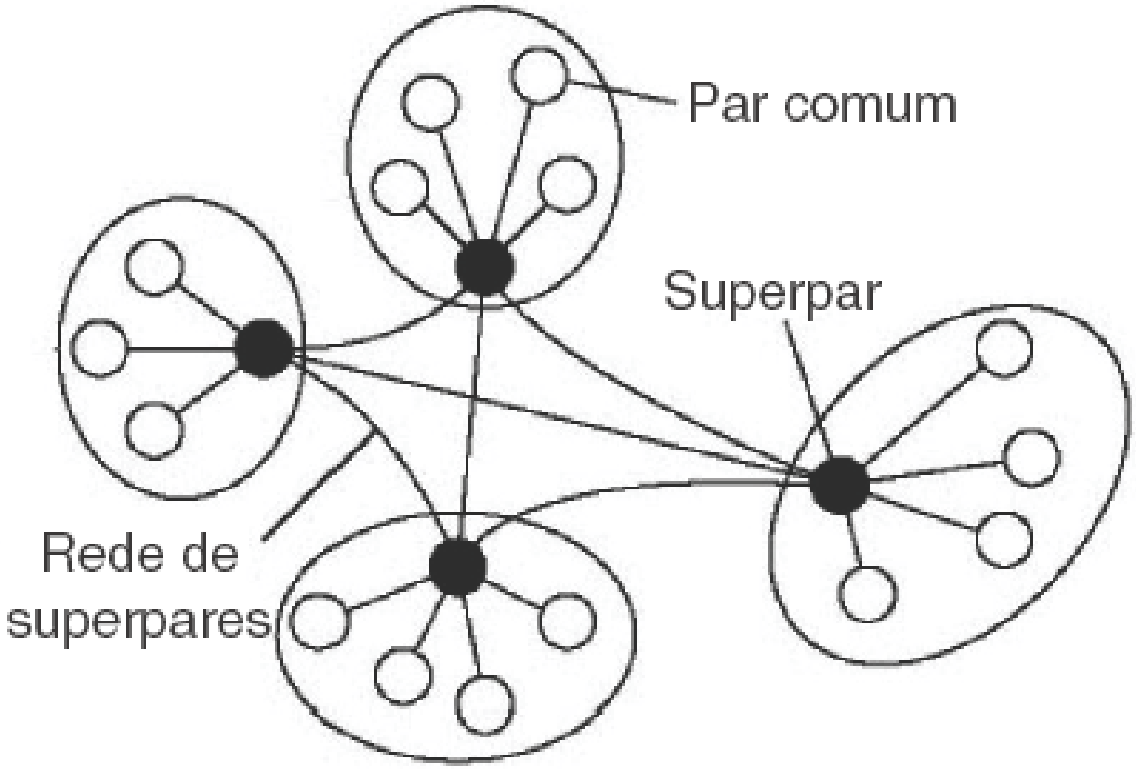


Figura 2.12 Organização hierárquica de nós em uma rede de superpares.

Questão

- 10. Nem todo nó de uma rede P2P deve tornar-se superpar. Quais são os requisitos razoáveis que um superpar deve ter?

- 10. Em primeiro lugar, o nó deve ser altamente disponível, uma vez que muitos outros nós contarão com ele. Além disso, deve ter capacidade suficiente para processar as requisições. O mais importante talvez seja o fato de que ele pode ser confiável para fazer bem o seu trabalho.

2.2.3 Arquiteturas híbridas

- Combinação de soluções cliente-servidor com arquiteturas descentralizadas
  - Sistemas de servidor de borda (Internet)
  - Sistemas colaborativos (Torrent)

2.2.3 Arquiteturas híbridas - Internet

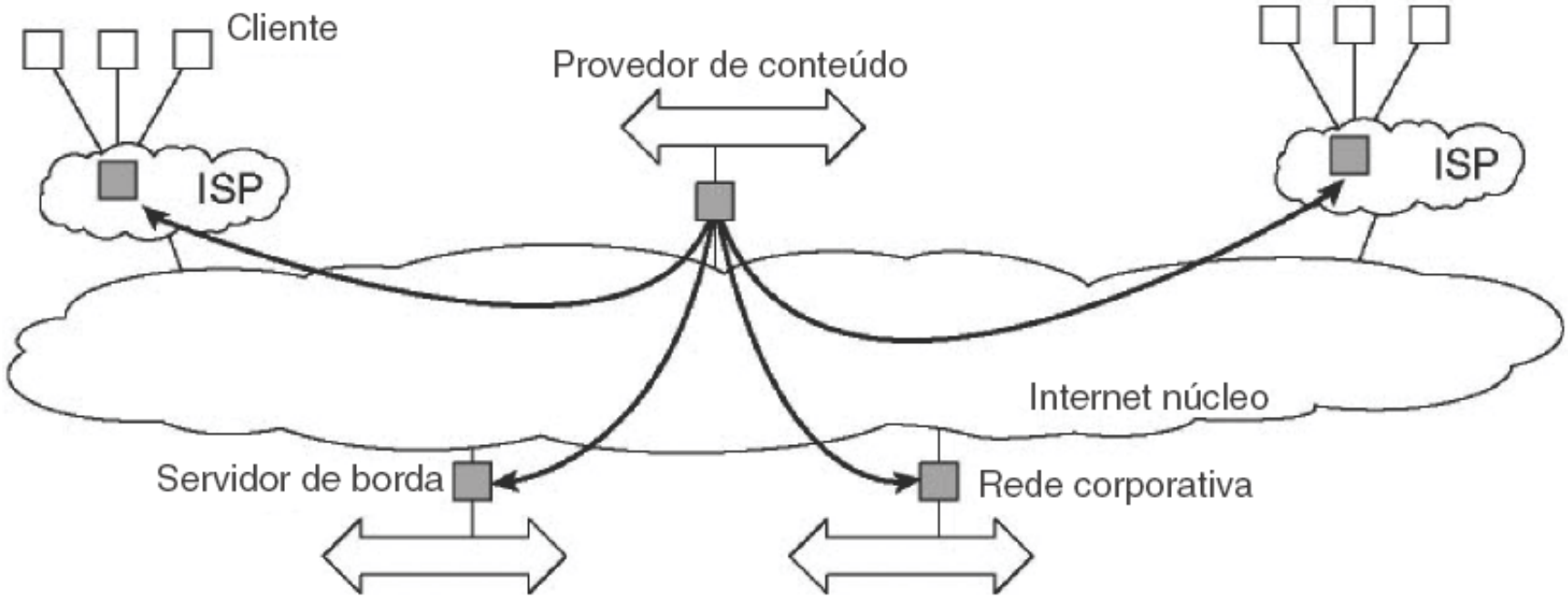


Figura 2.13 Visão da Internet como rede composta por um conjunto de servidores de borda.

2.2.3 Arquiteturas híbridas - BitTorrent

- Um usuário acessa um diretório global que contém um arquivo *.torrent*, o qual contém as informações necessárias para transferir um ou mais arquivos específicos
- O arquivo se refere a algo conhecido como rastreador.
  - Rastreador é o servidor que mantém uma contabilização precisa de nós ativos que tem o arquivo requisitado, ou porções dele.

2.2.3 Arquiteturas híbridas - BitTorrent

- Após a identificação dos nós e suas porções de arquivos, o nó que está transferindo se torna efetivamente ativo, sendo forçado a auxiliar os outros, garantindo a colaboração.

Funcionamento do BitTorrent

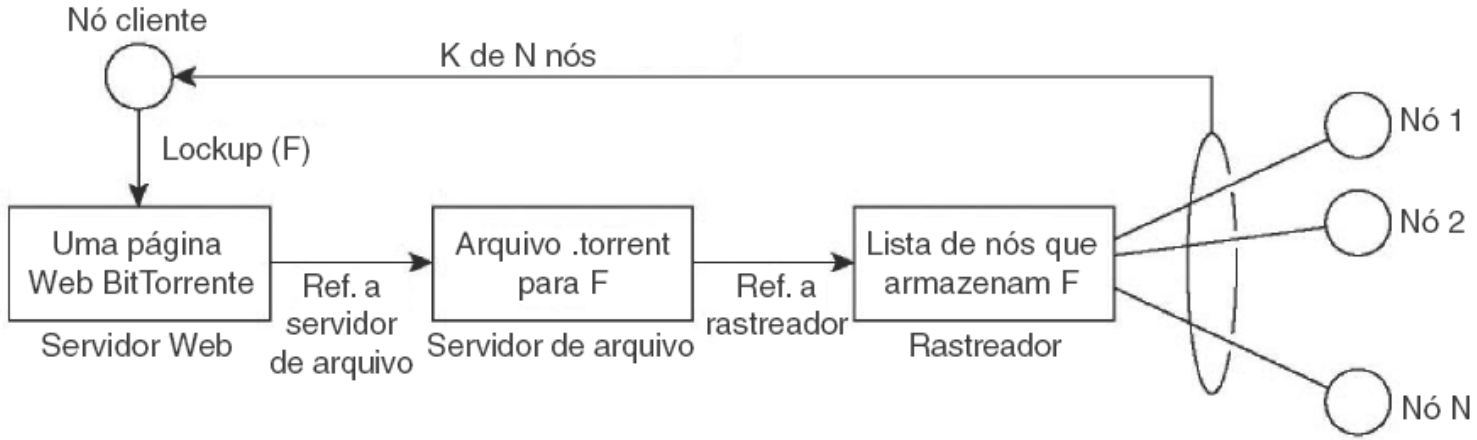


Figura 2.14 Funcionamento principal do BitTorrent [adaptado com permissão de Pouwelse et al. (2004)].

Questão

- 12. Dê um argumento (técnico) convincente porque a política toma-lá-dá-cá, usado no BitTorrent está longe de ser ideal para o compartilhamento de arquivos na Internet.

- 12. O raciocínio é relativamente simples. A maioria dos clientes BitTorrent operam por meio de ligações assimétricas, como fornecidas pelo ADSL ou modems à cabo. Em geral, é oferecida aos clientes alta largura de banda de capacidade de entrada, mas ninguém realmente espera que os clientes tenham serviços a oferecer. BitTorrent não faz essa suposição, e transforma clientes em servidores colaborativos. Ter então conexões simétricas é muito melhor para a política toma-lá-dá-cá.

2.3 Arquiteturas *versus* Middleware

- Onde o Middleware se encaixa?
  - Entre aplicações e plataformas distribuídas, com finalidade de proporcionar um grau de transparência à distribuição de dados, processamento e controle.
- Middlewares normalmente seguem estilos arquitetônicos específicos:
  - CORBA (baseado em objetos);
  - TIB/Rendezvouz (baseado em eventos);

2.3.1 Interceptadores

- É um software que interromperá o fluxo de controle usual e permitirá que seja executado um outro código (específico da aplicação).
- Funcionamento em O.O.

1. Oferecida a A uma interface local igual à oferecida pelo objeto B.
2. A chamada por A é transformada em uma invocação a objeto genérico, possibilitada por meio de uma interface geral de invocação de objeto oferecida pelo middleware na máquina em que A reside.
3. Por fim, a invocação a objeto genérico é transformada em uma mensagem que é enviada por meio de uma interface de rede de nível de transporte como oferecida pelo sistema operacional de A.

Funcionamento de um Interceptador

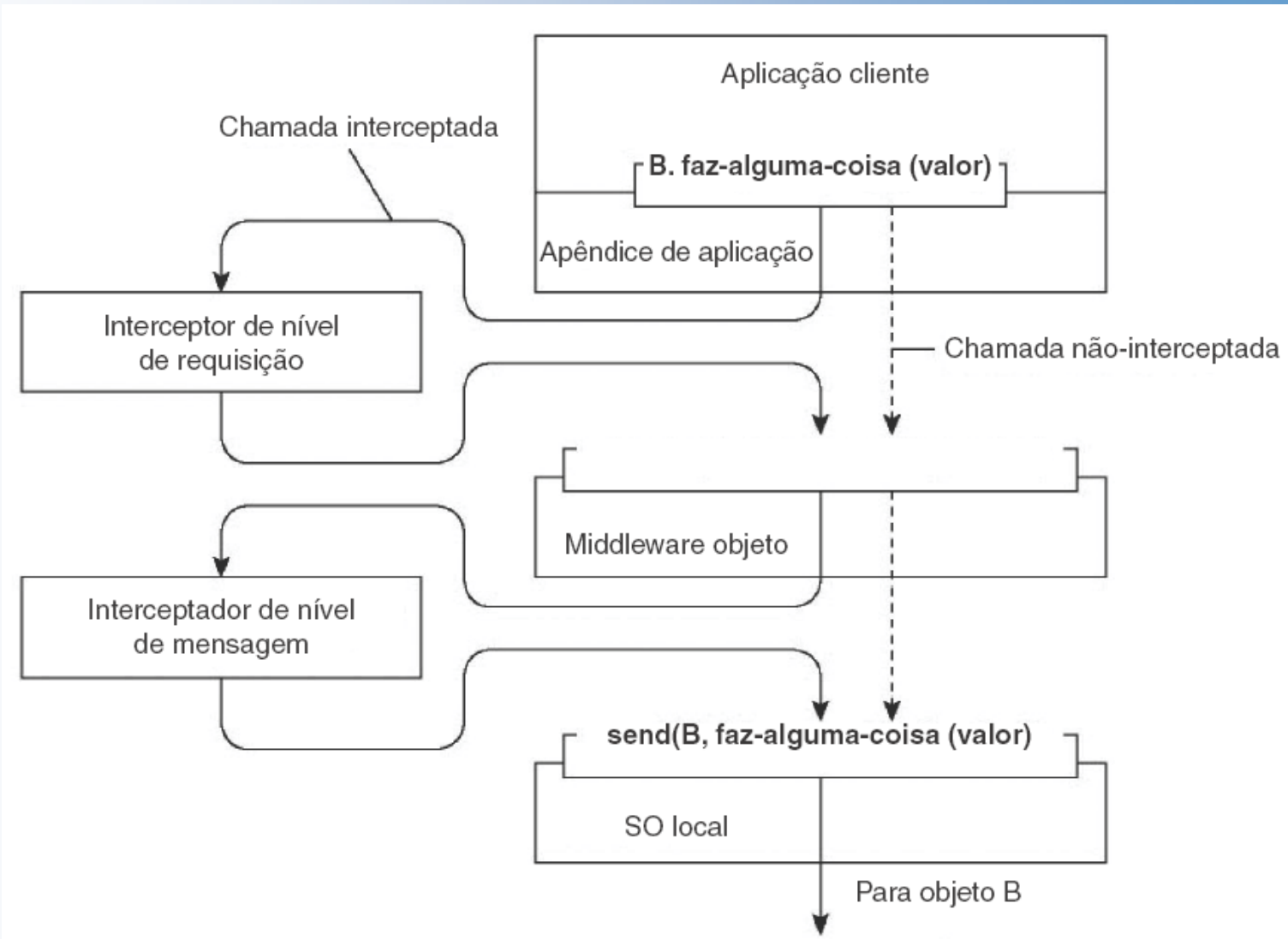


Figura 2.15 Utilização de interceptadores para manipular invocações de objeto remoto.

2.3.1 Interceptadores - Exemplo

- Replicação transparente
  - Não seria necessário componentes específicos para lidar com isso.
  - Não é necessário tratar cada réplica especificamente
  - Interceptador cuida disso
- Fragmentação de Dados
  - Fragmentação de grande volume de dados em partes menores e remontagem no receptor
    - Melhora desempenho e confiabilidade
  - Interceptador pode fazer sem ciência do middleware.

Questão

- 13. Nós demos dois exemplos de uso de interceptores em middleware adaptativo. Quais outros exemplos vêm à mente?

- 13. Há vários. Por exemplo, poderíamos usar um interceptor para suportar mobilidade. Nesse caso, um interceptor de nível de requisição primeiramente faria a procura pelo local atual de um objeto referenciado antes da chamada ser encaminhada. Da mesma forma, um interceptor pode ser usado para transparentemente criptografar mensagens quando a segurança está em jogo. Outro exemplo é quando o log é necessário. Em vez de deixar isso ser manipulado pelo aplicativo, poderíamos simplesmente inserir um interceptor de método específico que iria gravar eventos específicos antes de passar a chamada para o objeto referenciado.

Questão

- 14. Até que ponto os interceptores são dependente do middleware em que são disponibilizados?

- 14. Em geral, interceptores são altamente dependentes do middleware. Se considerarmos a figura 2.8, é fácil ver o por que: o stub cliente provavelmente será fortemente acoplado às interfaces de nível mais baixo oferecidas pelo middleware, assim como interceptores de nível de mensagem serão altamente dependentes da interação entre middleware e o sistema operacional local. No entanto, é possível padronizar essas interfaces, abrindo caminho para o desenvolvimento de interceptores portáteis, embora muitas vezes por uma classe específica de middleware. Esta última abordagem tem sido seguido por CORBA.

2.3.2 Abordagens gerais para software adaptativo

- É necessário adaptar sistemas distribuídos por vários motivos, como hardware defeituoso ou esgotamento de bateria. Isso levou aos softwares adaptativos, que são responsáveis por reagir a essas mudanças.

1. Separação de Interesses – Separa as partes que implementam funcionalidades das que cuidam de funcionalidades extras. Além disso, entrelaça os interesses cruzados em um sistema distribuído (software orientado a aspecto).
2. Reflexão computacional – Programa inspeciona a si mesmo e, se necessário, adapta-se ao ambiente.
3. Projeto baseado em componente – adaptação por meio de composição.

Autogerenciamento em SDs
Modelo de realimentação e controle

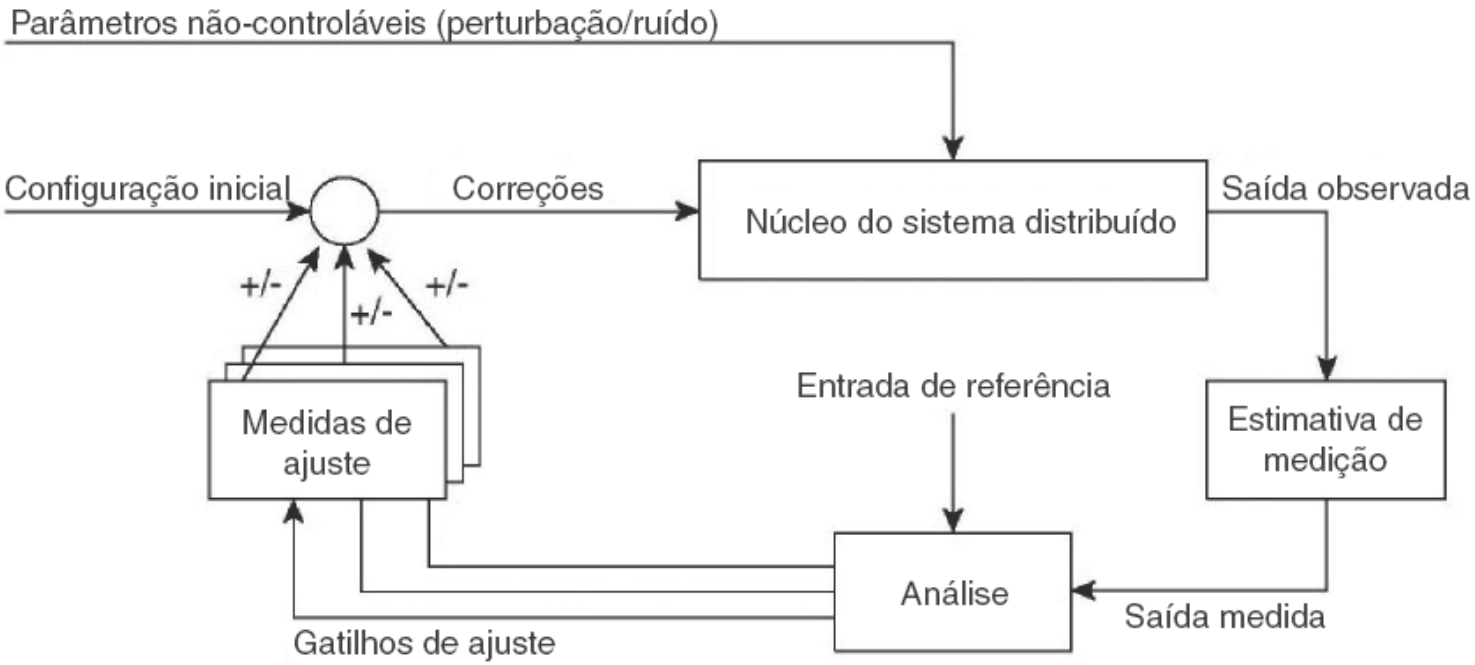


Figura 2.16 Organização lógica de um sistema de realimentação de controle.

Questão

- 15. Os carros modernos são recheados de dispositivos eletrônicos. Dê alguns exemplos de sistemas de realimentação de controles em carros.

- 15. Uma resposta óbvia é o controle de cruzeiro. Por um lado, este subsistema mede a velocidade atual, e quando muda a partir da configuração desejada, o carro é retardado ou acelerado. O sistema de freios anti-bloqueio (ABS) é outro exemplo. Pulsa os freios de um carro, enquanto que, ao mesmo tempo regula a pressão que cada roda está a exercendo, sendo assim é possível continuar a direção sem perder o controle caso as rodas estivessem bloqueadas. Um último exemplo é formado por um circuito fechado de sensores que monitoram as condições do motor. Assim quando um estado perigoso é alcançado, um carro pode parar automaticamente para evitar o pior.