Arquiteturas de Sistemas Distribuídos

Prof. Talles
talles@pucgoias.edu.br

Arquiteturas de Sistemas Distribuídos

Sumário da aula:

- Introdução
- Estilos Arquitetônicos
- Arquiteturas de Sistemas
  - Arquiteturas Centralizadas
  - Arquiteturas Descentralizadas
  - Arquiteturas Híbridas
- Arquiteturas *versus* Middleware
  - Interceptadores
  - Software Adaptativo
- Autogerenciamento em SD

Referências Principais:

- Tanenbaum A. S. e Van Steen. M. Sistemas Distribuídos, 2 ª edição, Prentice Hall, 2008. (cap. 2)
- Coulouris et al. Sistemas Distribuídos – Conceitos e Projeto, 4 ª edição, Bookman, 2007. (cap 2.)

Arquiteturas de Sistemas Distribuídos: Introdução

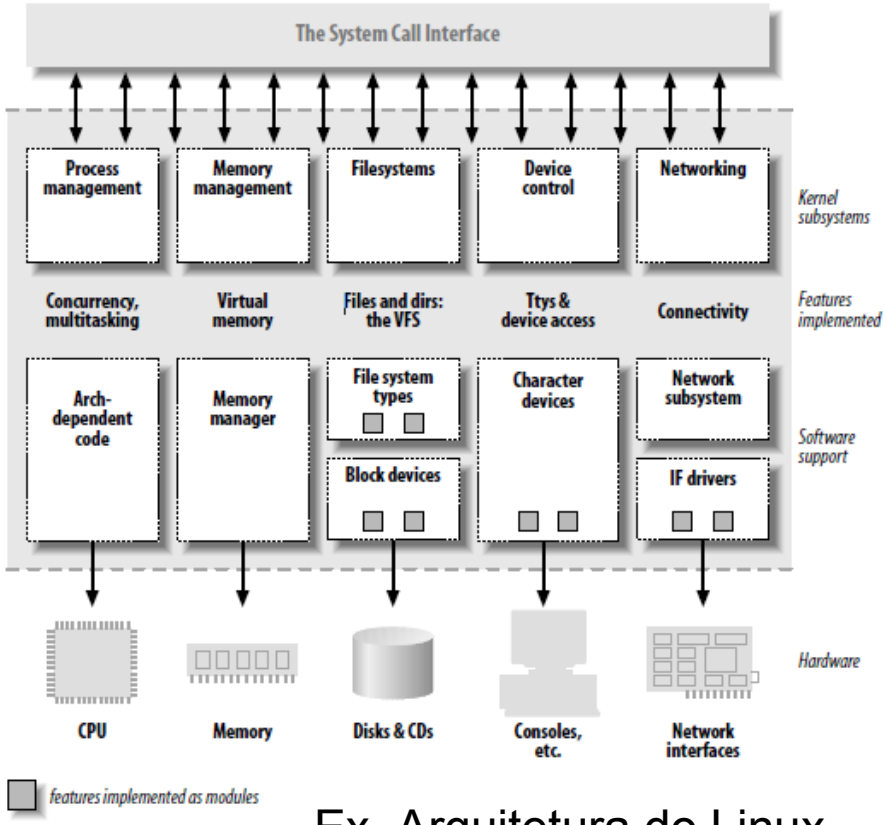
- Componentes Lógicos *versus* Componentes Físicos de um Sistema Distribuído

Conceitos Importantes:

Arquitetura de software de sistema: a estrutura dos componentes de um programa/sistema, seus interelacionamentos, princípios e diretrizes guiando o projeto e a evolução ao longo do tempo;

Componente: uma unidade modular com interfaces requeridas e fornecidas bem definidas que é substituível dentro de seu ambiente;

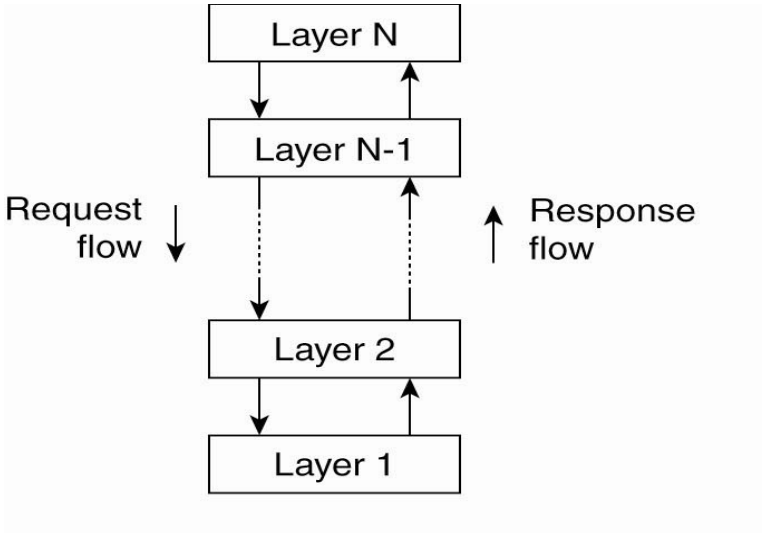
Conectores: mecanismos para mediar a comunicação ou a cooperação entre os componentes.



Ex. Arquitetura do Linux

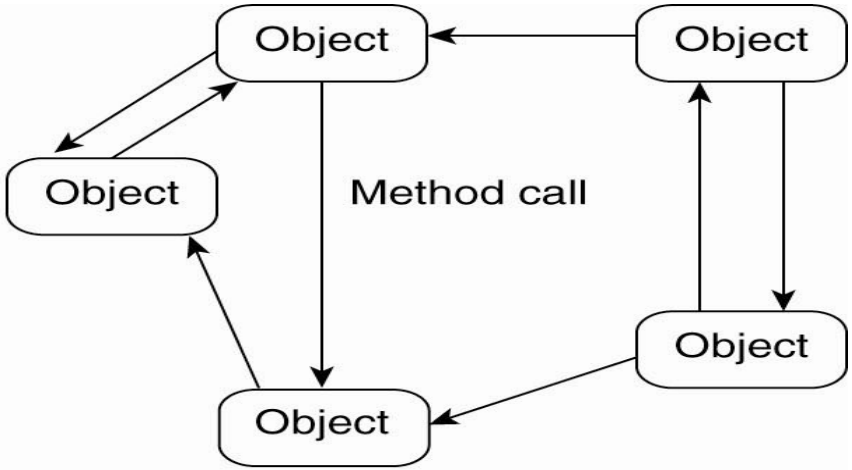
Arquiteturas em Camadas

- Um componente da camada Ni tem permissão para chamar um componente da camada Ni-1, mas não o contrário;
- A arquitetura em camadas não especifica a *granularidade* dos componentes. O número de camadas varia de acordo com a quantidade de funcionalidades;
- Manutenibilidade e Desempenho devem ser balanceados



Exs. Modelo OSI da ISO e Arquitetura TCP/IP

Arquiteturas Baseadas em Objetos



- Um conjunto de objetos distribuídos e comunicantes com estados associados
- Em geral, cada objeto é um componente. As conexões são associações ou agregações
- Facilidade para o mapeamento (modelagem): o mundo real é composto de objetos;
- Facilidade para a manutenção, o reuso, a distribuição e a execução concorrente e/ou paralela: independência dos objetos
- Transparência deve ser garantida. Ex. Incocação implícita de todos os objetos conectados ao objeto chamado

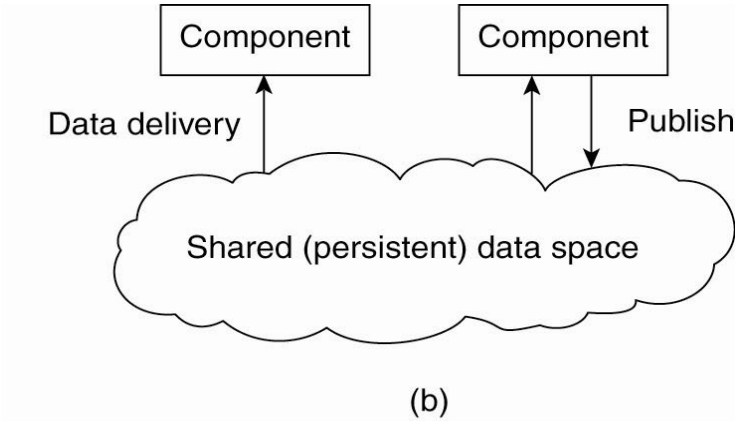
Arquiteturas baseadas em Eventos (a)

- Componentes (processos) se comunicam pela propagação de eventos que, opcionalmente, transportam dados. Não precisão se referir explicitamente uns aos outros, são **Referencialmente Desacoplados**
- Modelo de interação **Publicar/Subscrever**: apenas os componentes inscritos (que subscreveram) para determinados eventos os receberão

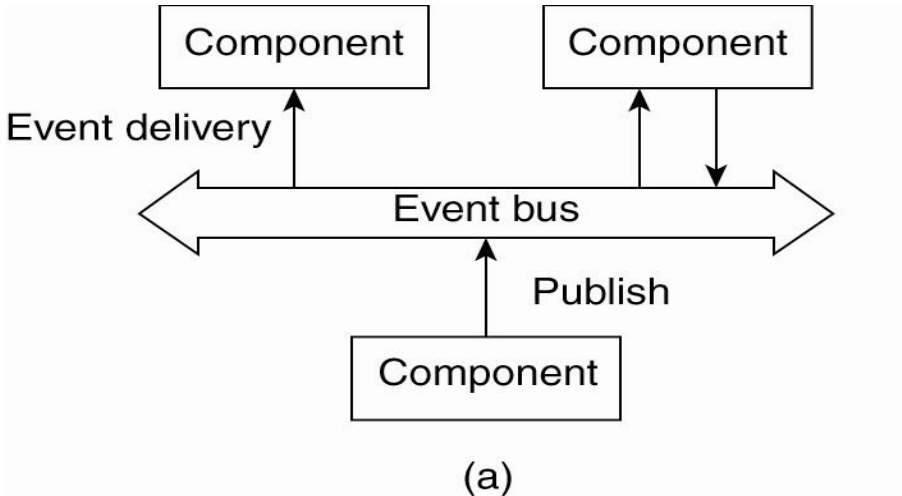
Arquiteturas centradas em dados (b)

- Processos se comunicam por meio de um repositório comum, ativo ou passivo.

Ponto de vista atual: adoção de modelos híbridos. Nenhuma solução até o momento é capaz de abarcar todos os requisitos não funcionais requeridos para todos os tipos de aplicações

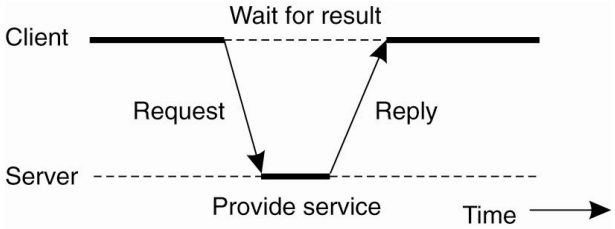


(b)

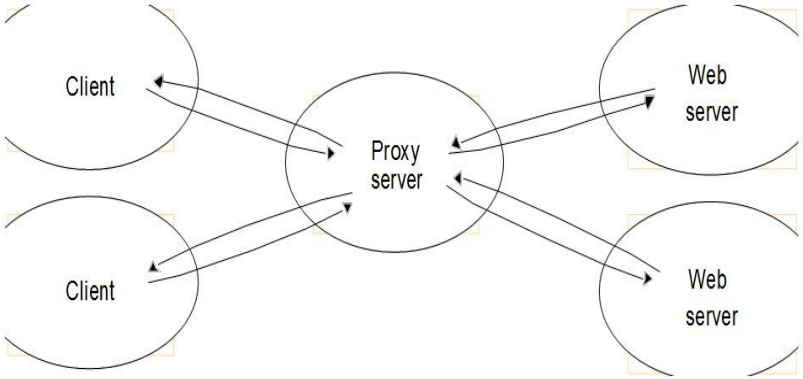


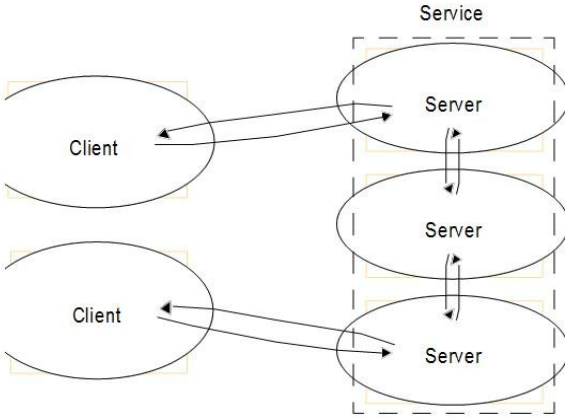
(a)

Arquiteturas Centralizadas: Modelo Cliente/Servidor



- Processos e Serviços
- ! máquinas (estações)
- Comportamento Requisição/Resposta
- Síncrona (bloqueante) x Assíncrona (não bloqueante)
- Espera deferida (*get response()*)
- Operação Idempontente x não idempotente
- Principais Classificações de Servidores: concorrentes x Servidores iterativos; Servidores com Conexão x Sem Conexão; Servidores Ativos x Servidores Passivos; Servidores *Statefull* x Servidores *Stateless; Servidores inband x outband;* Múltiplos servidores e Servidores intermediários ou *Proxy*



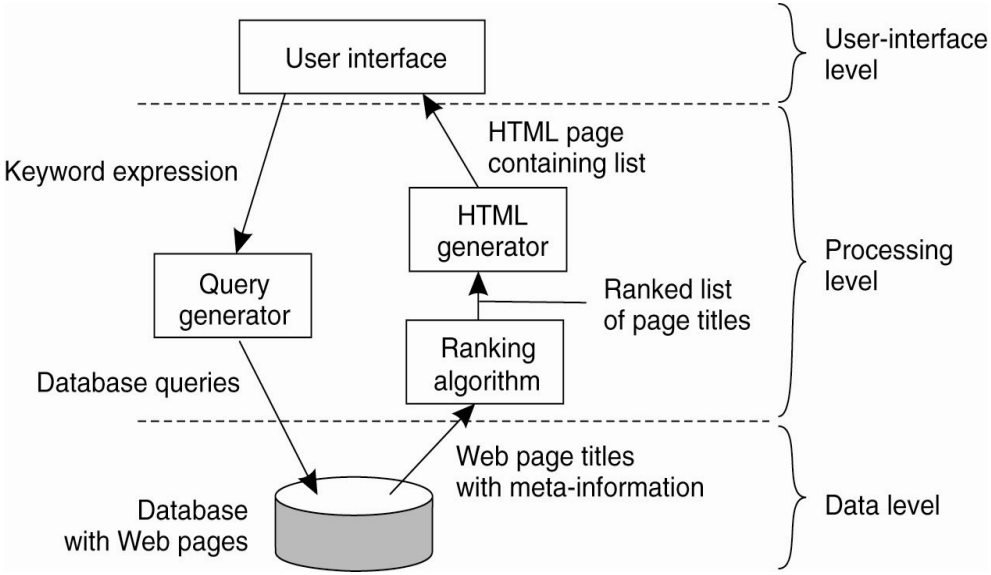


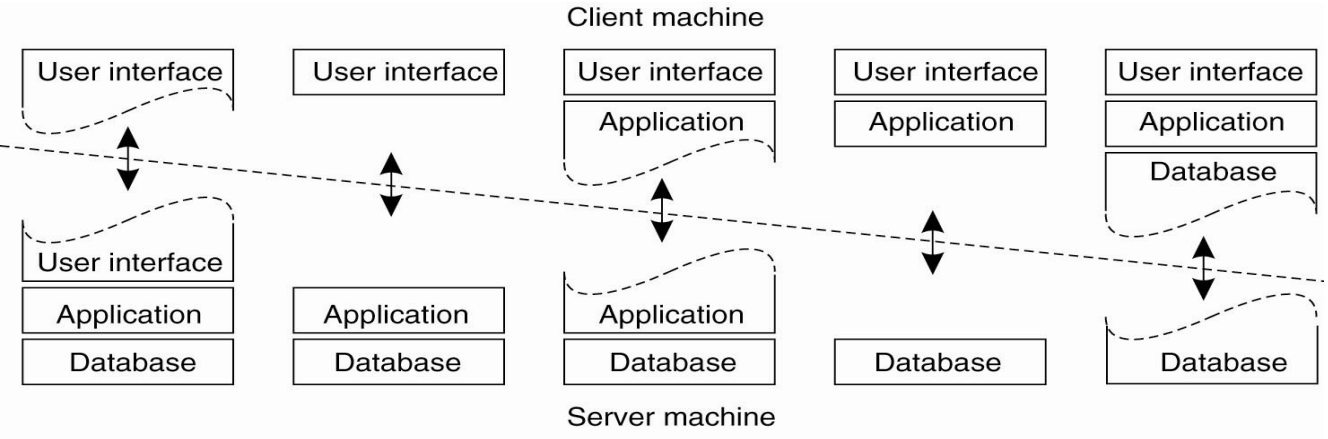
Múltiplos Servidores

Arquiteturas de Sistemas Distribuídos

Arquiteturas Centralizadas: Modelo Cliente/Servidor

- Camadas de aplicação
  - Interface de usuário
  - Processamento
  - Dados
- Arquiteturas multidivididas
  - Clientes gordos (*fat clients*)
  - Clientes magros (*thin clients)*





Organização simplificada de um mecanismo de busca da Internet

Distribuição vertical

Origem: **Fragmentação vertica**l, inerente aos BD Distribuídos: tabelas subdivididas em colunas e distribuídas em várias máquinas

Arquiteturas Descentralizadas

Principal característica: Distribuição horizontal

- Interação simétrica entre pares (*peer-to-peer)*

Rede de Sobreposição (*Overlay Network*): nós são formados por processos e os enlaces representam os possíveis canais de comunicação!

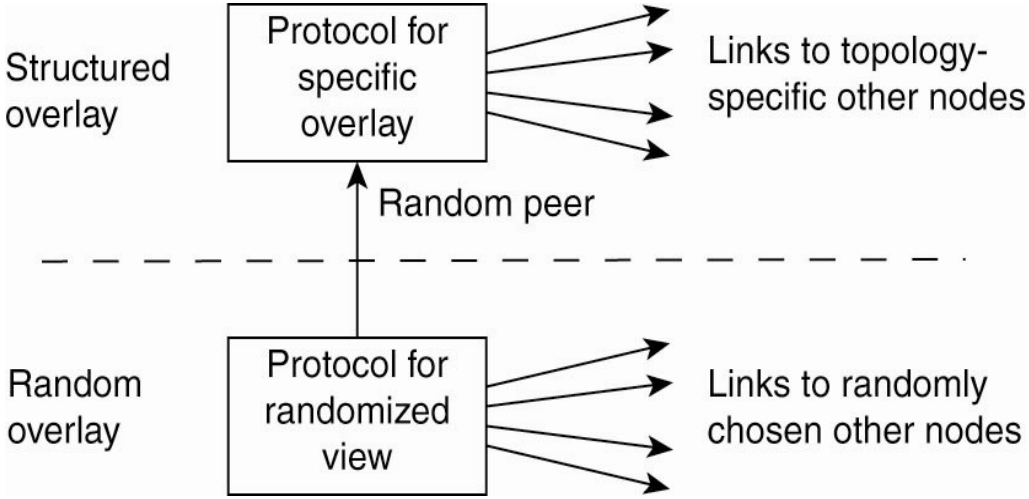
- Arquiteturas *peer-to-peer* estruturadas (*Structured Ovelay*): contruídas por procedimentos determinísiticos
  - *The Lookup Problem*
  - Tabela de Hash Distribuída (*Distributed Hash Table* – DHT): mapeia uma chave em um item de dado armazenado em um nó

Para casa: Ler e fazer uma resenha do texto “Looking up Data in P2P Systems” (Balakrishnan *et al*., 2003). Apoio: Seção 5.2.3 (do livro de Tanenbaum e Van Steen) e/ou cap. 10 (do livro de Coulouris *et al*.)

Arquiteturas Descentralizadas

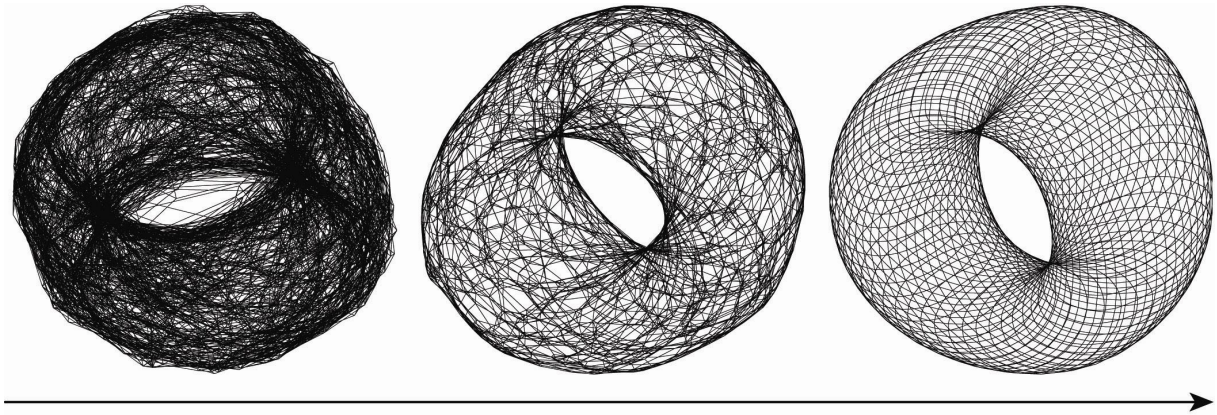
- Arquiteturas *peer-to-peer* não estruturadas (*Random Overlay*): utiliza algoritmos aleatórios para construir a rede de sobreposição.
  - Organização aleatória dos nós e associação dos itens de dados aos nós
  - *Flooding* para consulta de busca
  - Grafo aleatório
    - Visão Parcial: cada nó mantém uma lista de vizinhos, composta pelos nós ativos escolhidos aleatoriamente
    - Nós trocam entradas regularmente de sua visão parcial
    - Atualização pelo descarte de entradas velhas
    - Risco de um nó se tornar popular: nós ativos a mais tempo tendem a ser mais referenciados (alto grau interno)
    - Desequilíbrio em relação à carga de trabalho
    - Solução apontada por Tanenbaum e Van Steen: adoção de uma abordagem de gerenciamento de duas camadas para balanceamento da rede de sobreposição
    - Resultado: aumento da eficiência dos algoritmos de busca

Gerenciamento de topologia de redes de sobreposição



Nós são ordenados em relação a algum critério em relação a determindo nó (*random peer*)

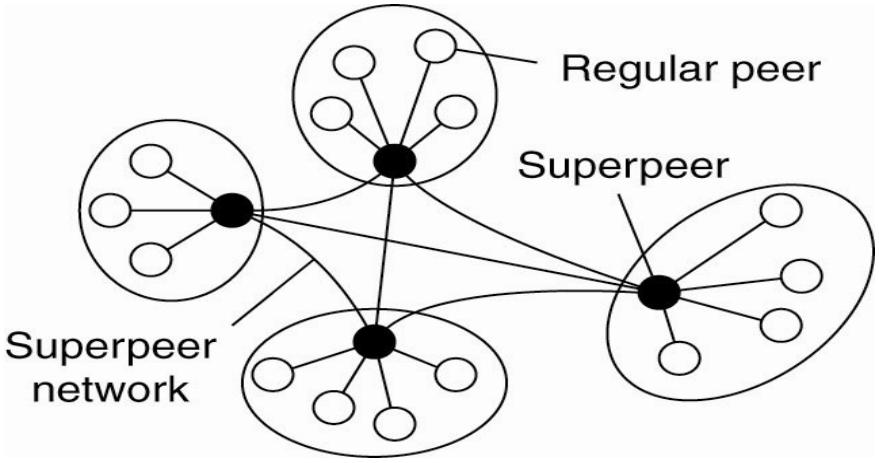
Nós periodicamente trocam entradas de suas visões parciais com objetivo de manter um grafo aleatório preciso. Precisão: visão parcial preenchida com entradas referentes a nós ativos selecionados aleatoriamente



Melhor balanceamento da carga ao longo do tempo

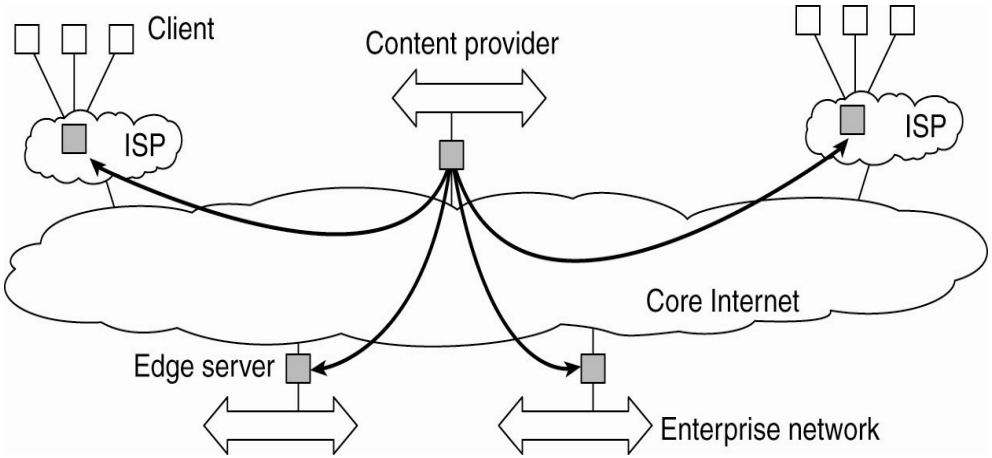
Redes de sobreposição

- Superpares (*superpeers*): mantém um índice de itens de dados ou agem como intermediários
- Estabelecem uma organização hierárquica: toda comunicação de e para um par comum ocorre por meio de um super par associado ao par
- Minimizam o problema de escalabilidade de redes de sobreposição não estruturadas pela falta de um mecanismo determinístico para rotear as buscas
- Riscos pela centralização
- Falhas e perdas de desempenho afetam mais do que apenas um *peer*
- Necessidade de escolha dos *superpeers*



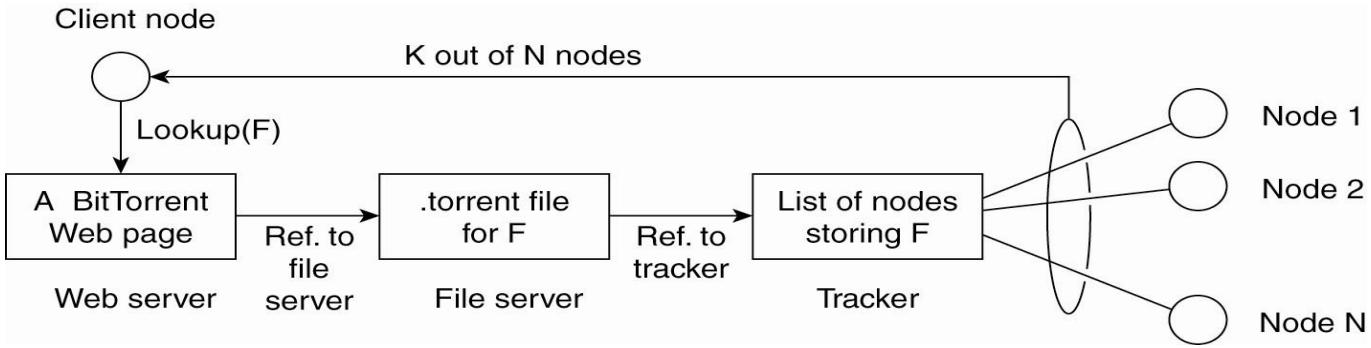
Arquiteturas Híbridas

- Sistemas de Servidor de Borda
- Sistemas Distribuídos Colaborativos



Borda é a fronteira entre a rede corporativa e a Internet.
Principais funções do Servidor de borda: **servir conteúdo** – aumentar desempenho pelo *caching* e/ou balanceamento da carga e/ou transcodificação; suporte para maior **tolerância a falhas** (replicação); **segurança**: autenticação, criptografia e, principalmente, filtragem.

Rastreador: servidor que mantém a lista dos nós ativos que têm o arquivo requisitado.



Arquiteturas Híbridas: Código Móvel e Agentes

Agente: entidade autônoma, capaz de executar uma ação sem a interferência de um sistema ou de outro agente. Um agente interage com o ambiente à sua volta e pode interagir com outros agentes.

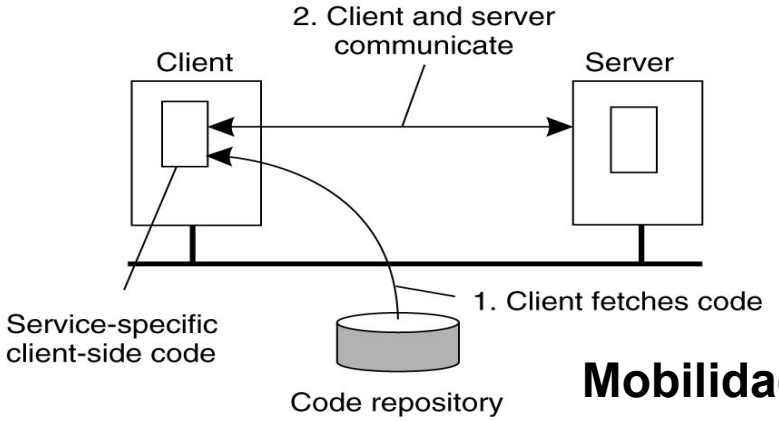
- **Agente inteligente:** apresenta comportamento racional
- **Agente móvel:** suporta a migração de dados, de processamento e, até mesmo, de processos (**mobilidade forte**)

Futuro de muitas aplicação da Internet (Ex. o comércio eletrônico)

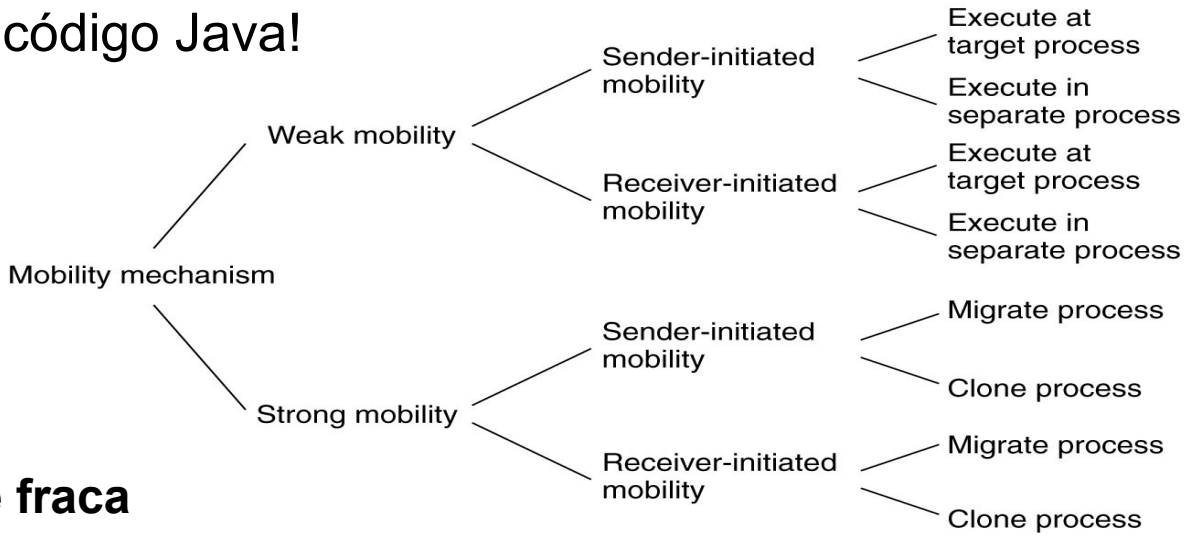
- Plataforma Aglets (http://sourceforge.net/projects/aglets/files/)

Um contra-exemplo: Um applet para autenticar uma senha bancaria

- Apenas Mobibilidade do código Java!



Mobilidade fraca



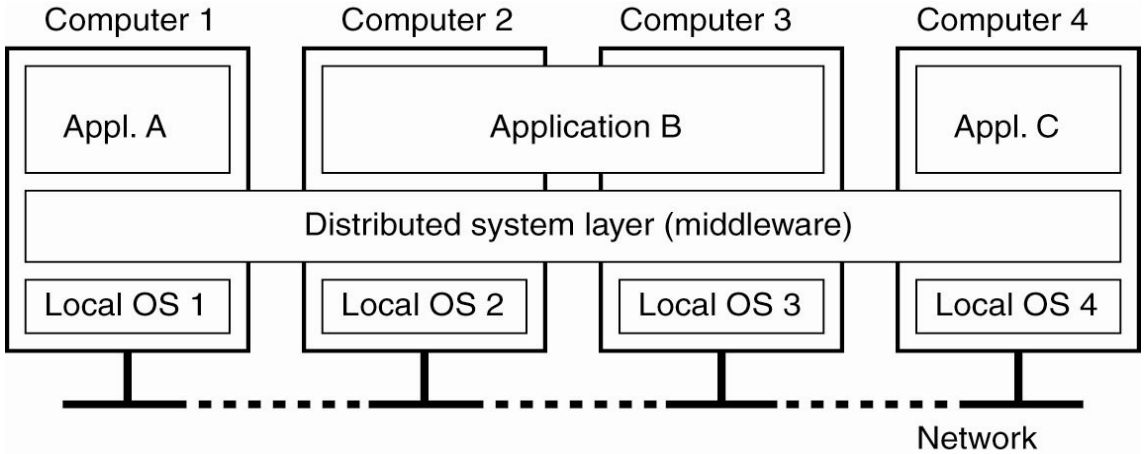
Middleware

- Definição mais usada em SD: *camada de software intermediária entre a plataforma de suporte e a aplicação*
  - Plataforma de suporte: HW + Kernel

Principais finalidades:

- Transparência de distribuição: ocultar das aplicações, até certo ponto, a distribuição dos dados, do processamento e do controle (gerência)
- Prover facilidades para as aplicações: acesso remoto, nomeação, controle de concorrência, transações,...
- Preocupação atual: fazer sistemas de *middleware* que sejam mais simples configurar, adaptar e personalizar para um conjunto de aplicações

Clara separação entre mecanismos e políticas!

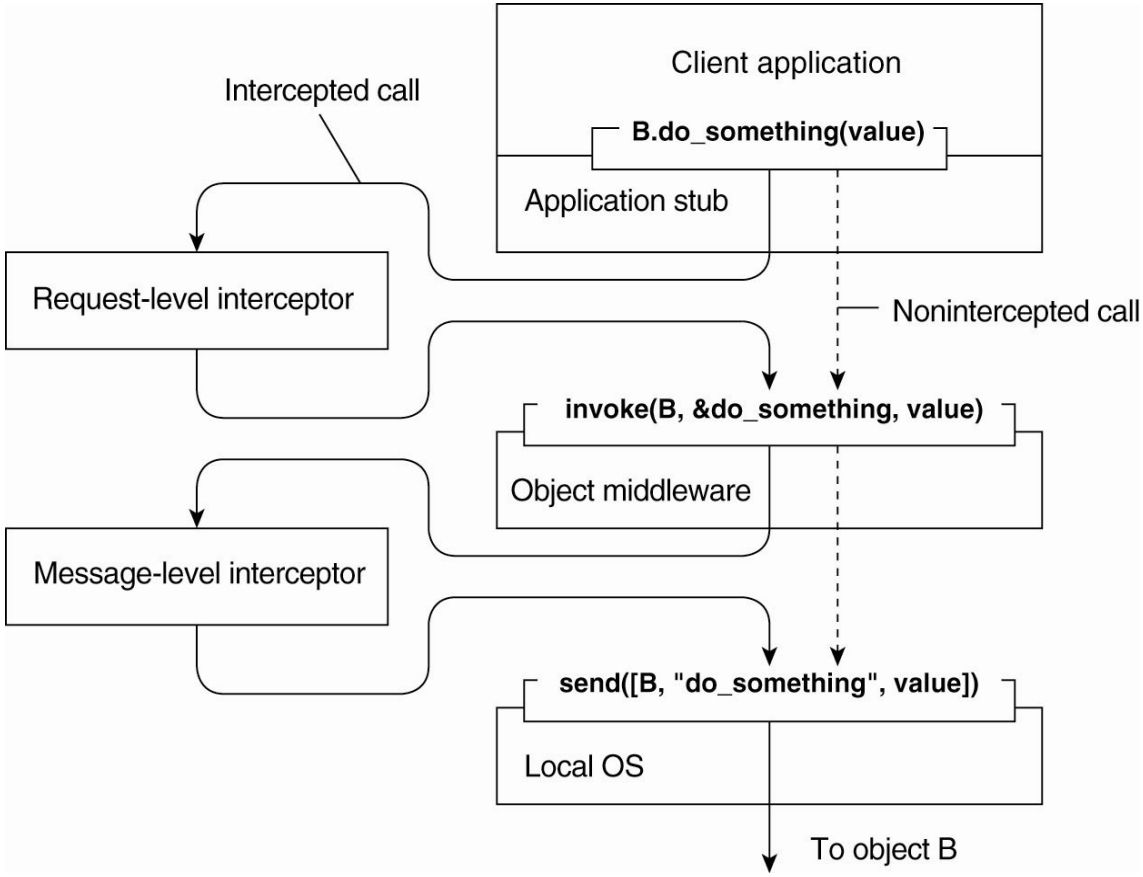


Middleware: Interceptadores

- Um mecanismo que interrompe o fluxo de controle usual e permite que seja executado um outro código (específico da aplicação). Um "adaptador" para o *middleware*
- Suportados em muitos sistemas distribuídos baseados em objetos

Nível de requisição: transparência quanto ao tratamento de objetos. Replicação, tipo de invocação, ...de acordo com o interesse da aplicação

Nível de mensagem: transparência quanto ao tratamento da mensagem. Por ex., *Parsing*, fragmentação,... de acordo com o interesse da aplicação



Software Adaptativo:

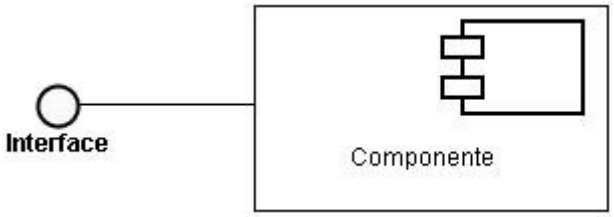
Motivações:

- Requisitos conflitantes → software com tamanho exagerado e complexidade inerente
- Possibilitar que o software mude à medida que o ambiente muda → necessidade de flexibilidade
- Muitos SD não podem ser desligados

Objetivo final: implementar comportamento reativo sem a intervenção humana → Computação Autonômica (discutiremos ao final...)

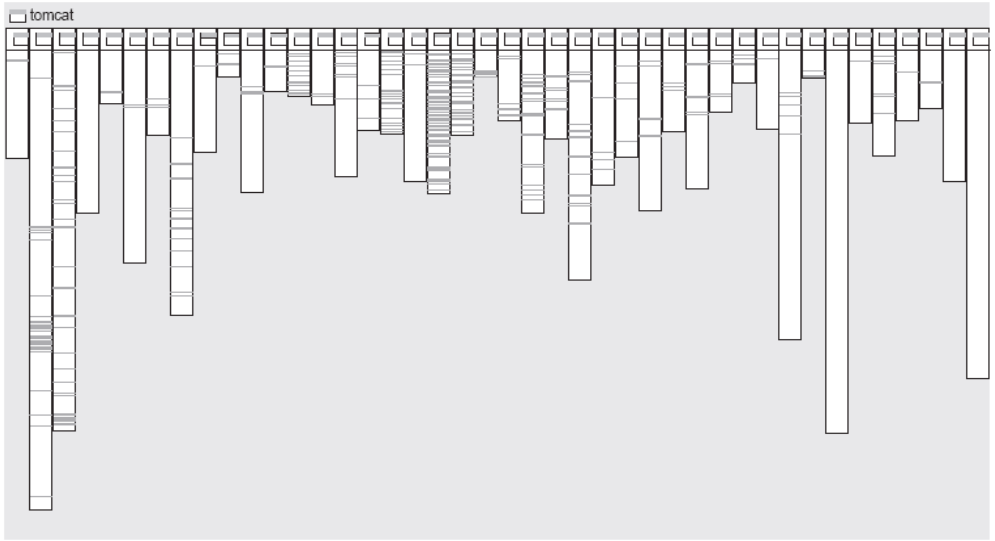
Principais Abordagens de Software Adaptativo para SD:

- Projeto baseado em componentes (premissa básica)
- Separação de Interesses
- Metaprogramação
  - Em tempo de Compilação
  - Em tempo de Execução

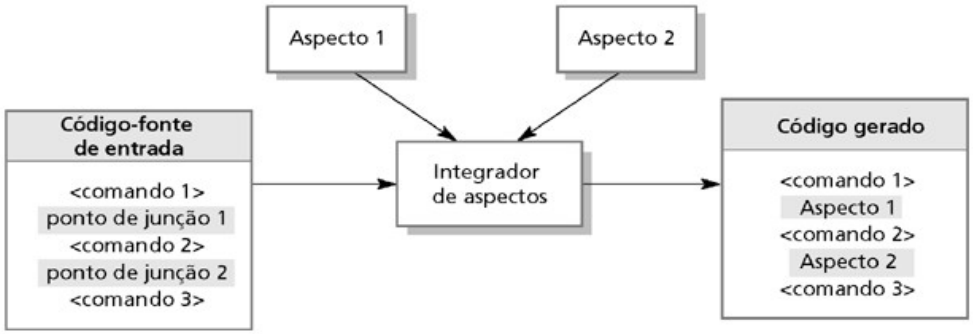


Software Adaptativo: separação de interesses

- Foco na modularização do sistema
  - Separar as partes que implementam funcionalidades das que cuidam de outras coisas. Essas partes estão espalhadas em todos os programas. Ex. Controle de Exceções em Java
  - Facilita o desenvolvimento e a adaptação às novas demandas. Entretanto, separa-se e, na sequência, reúne os chamados **interesses cruzados**
  - Desenvolvimento de software **Orientado a Aspectos**
  - Ainda, poucos projetos de SD orientados a aspectos



Ocorrência de código emaranhado
(implementação do Tomcat)



Software Adaptativo: metaprogramação

"Metaprogramas são programas que representam ou manipulam outros programas ou eles mesmos. Pode ser em tempo de compilação ou em tempo de execução. São escritos em metalinguagens"

Generative Programming Paradigm

- Programas que escrevem outros programas
- Exemplos:
  - Um compilador:

linguagem de alto nível → linguagem de baixo nível

  - Programação baseada em *scripting*

Este programa gera um outro programa (chamado *program*). Quando executado imprime 993 mensagens (*strings*). Cada mensagem é um número de 1 a 992.

```
# !/bin/bash
# metaprogram
echo  '#! /bin/bash'  >program
for ((I=1; I<=992; I++)) do
    echo "echo $I"  >>program
done
chmod +x program
```

Qual a utilidade deste paradigma no desenvolvimento de SD?

Software Adaptativo: metaprogramação

Generic Programming Paradigm

Possibilita que programas possam ser escritos numa gramática que extende a linguagem original por meio de tipos parametrizados, também chamados de “generics” ou “templates”

```
Template <typename T>
Void Swap (T&a, T&b)
{
T temp = b;
b = a;
a = temp;
…
}

String hello = “world”;
String world = “hello”;
Swap (world, hello);
cout  <<hello<<world<< endl;
```

Componente implementado

Alterações sem a necessidade de modificar o componente

Qual a utilidade deste paradigma no desenvolvimento de SD?

Software Adaptativo: metaprogramação

- *Reflective Programming Paradigm*

  Possibilita que um programa possa inspecionar e até modificar a sua estrutura interna, em tempo de execução.

Paradigma tradicional:

- Dados são processados
- Instruções são executadas

Para suportar a Reflexão as instruções também devem ser processadas. Como implementar?

Preservando a estrutura interna (meta-informação) que é descartada após a compilação

Estruturas geradas pelos analisadores sintático e semântico possibilitam rastrear e modificar informações sobre métodos, nomes de classes, estados,...

Qual a utilidade deste paradigma no desenvolvimento de SD?

Software Adaptativo: metaprogramação

Vantagem das metaprogramação?

- Independência da plataforma de suporte
  - Portabilidade
- Modificabilidade
  - Reconfiguração e/ou reprogramação
- **Exemplo 1: Componentes metaprogramados de um Sistema operacional**
  - Reuso por parametrização: ajustes dependem da definição de alguns parâmetros, externos ao componente (em tempo de compilação)
  - Não necessita de mecanismos para "search-and-replace", por exemplo, um pré-processador para tradução de Macros. Todo custo das abstrações (memória) é

Software Adaptativo: metaprogramação

```
class AVR8_GPIO_Port:
        protected GPIO_Port_Common {
public:
 enum {
   PORTA = 0x39,
   PORTB = 0x36,
   PORTC = 0x33,
   PORTD = 0x30
 }; // ...
 void operator=(unsigned char value) {
   _ddr = (unsigned char)0xff;
   _port = value;
 }
 operator unsigned char() {
   _ddr = (unsigned char)0x00;
   return _pin;
 } // ...
private:
 IO_Register<unsigned char> _pin;
 IO_Register<unsigned char> _ddr;
 IO_Register<unsigned char> _port;
};
```

Usado no Sistema EPOS para implementação de mediador de *hardware* (Fröhlich *et al,* 2005), componente da Camada de Abstração do Hardware (HAL)

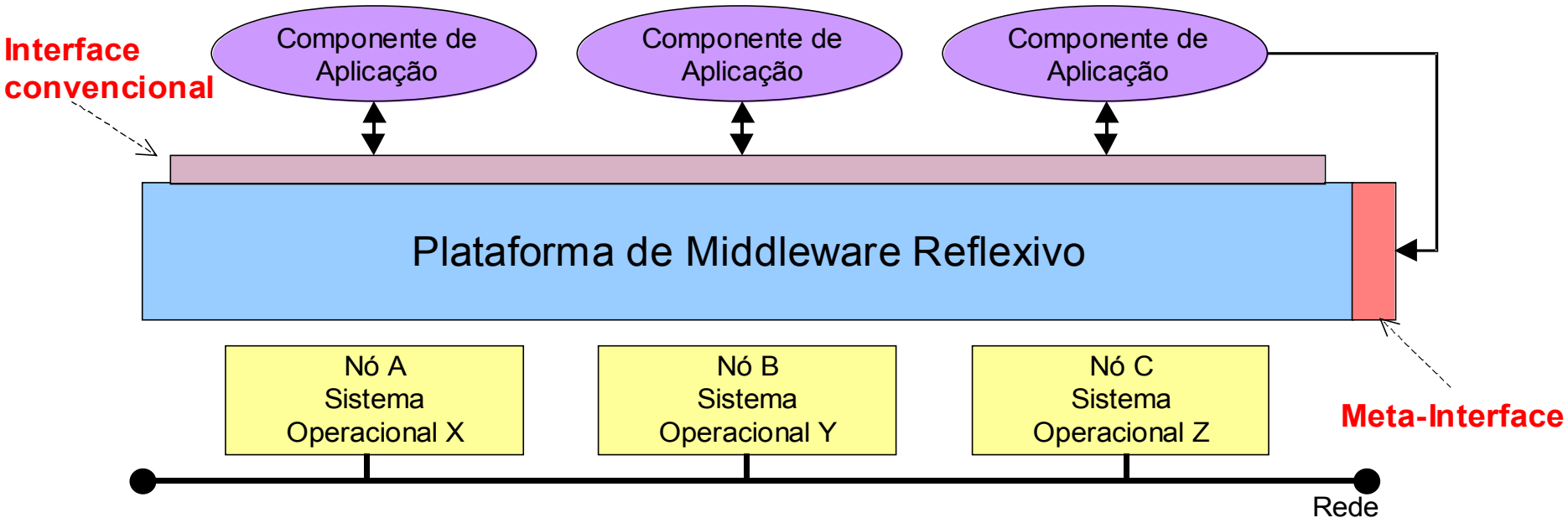
Neste exemplo, o mediador não necessita alterar o componete para modificar o acesso ao barramento GPIO (*General Purpose Input/Output*) de um microcontrolador

Software Adaptativo: metaprogramação

- **Exemplo 2: Middleware Reflexivo OO: conceitos básicos**

  Transparência para as aplicações que não estão interessadas nos detalhes da plataforma

  Translucidez para aplicações cujo desempenho pode melhorar pela sintonia fina do middleware



Costa F. Notas de aula, 2008.

Software Adaptativo: metaprogramação

- **Exemplo 2: Middleware Reflexivo OO: conceitos básicos**

Um **sistema reflexivo** mantém sua auto-representação interconectada de maneira causal (MAES, 1987)

- A plataforma mantém uma representação explícita de sua própria estrutura e comportamento internos
- Causalmente conectada:mudanças na representação geram mudanças correspondentes no middleware,e vice-versa

Reificação

ação de expor, de maneira programática, a representação interna de um sistema

- criação (ou atualização) da auto-representação do sistema

Absorção (reflexão)

ação de refletir mudanças realizadas na auto-representação nas respectivas entidades reificadas do sistema real

Software Adaptativo: metaprogramação

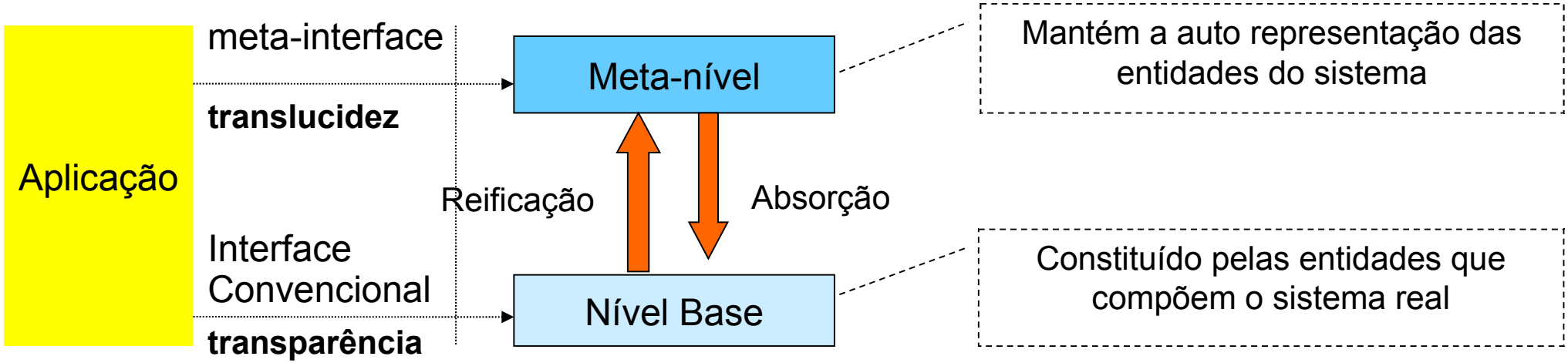
- **Exemplo 2: Middleware Reflexivo OO: arquitetura de meta-níveis**

Estrutura explícita em termos de:

- **nível base** – processamento a respeito do domínio de aplicação do sistema
  - representa o sistema em si
- **meta-nível** – processamento reflexivo, a respeito do próprio sistema

"O meta-nível pode ser visto como a máquina que executa o nível base".

Costa F. Notas de aula, 2008.



Software Adaptativo: metaprogramação

- **Exemplo 2: Middleware Reflexivo OO**

Reflexão estrutural: reificação completa da estrutura interna da plataforma, métodos e estados → mudanças funcionais

Ou seja, possibilita inspecionar e mudar a funcionalidade do middleware pelas mudanças na malha de componentes do meta-nível.

Reflexão comportamental: mudar a forma como requisições são processadas (interceptadores) → requisitos não funcionais e gerenciamento de recursos

Autogerenciamento em Sistemas Distribuídos:

Computação Autonômica: iniciativa da IBM, em 2001. O objetivo é projetar sistemas auto-gerenciáveis, ou seja, o projetista/usuário deve definir as políticas e o sistema deve ser capar de se auto-gerenciar

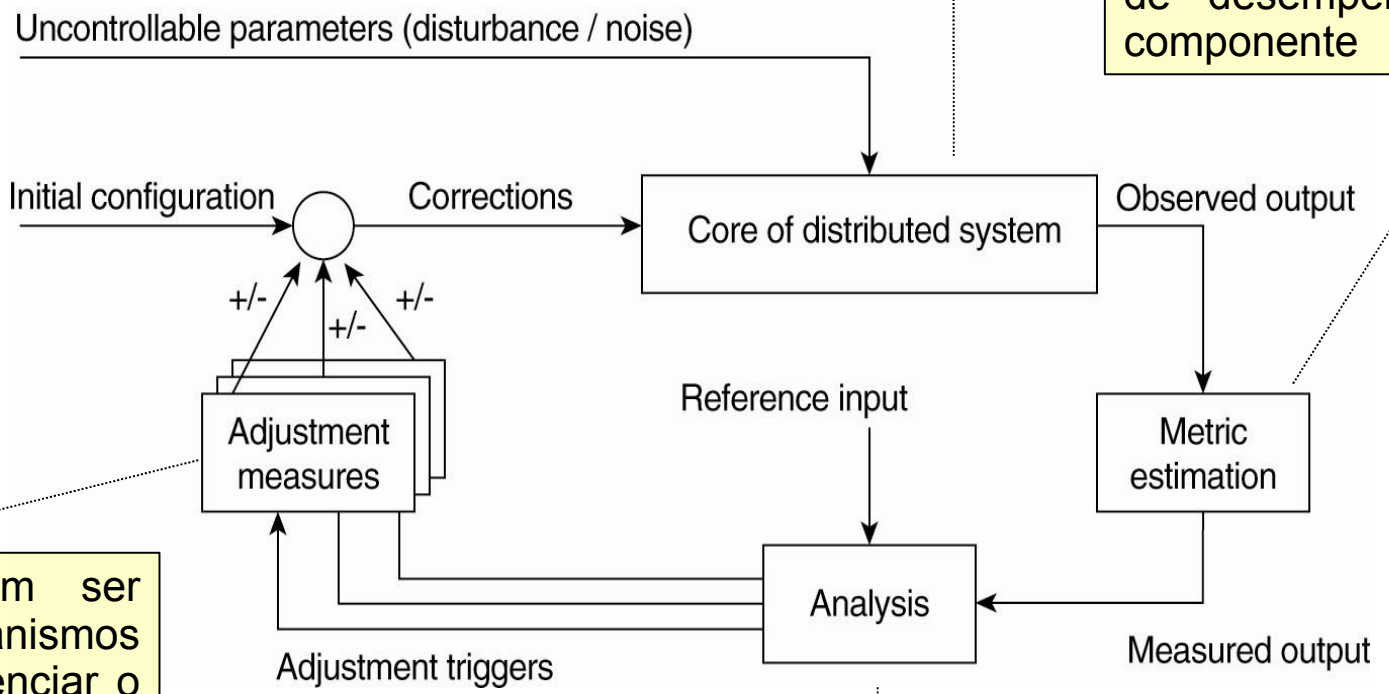
	Today	**The Autonomic Future**
Self-configure	Corporate data centers are multi-vendor, multi-platform. Installing, configuring, integrating systems is time-consuming, error-prone.	Automated configuration of components, systems according to high-level policies; rest of system adjusts automatically. Seamless, like adding new cell to body or new individual to population.
Self-heal	Problem determination in large, complex systems can take a team of programmers weeks	Automated detection, diagnosis, and repair of localized software/hardware problems.
Self-optimize	E.g.: WebSphere and DB2 have hundreds of nonlinear tuning parameters; many new ones with each release.	Components and systems will continually seek opportunities to improve their own performance and efficiency.
Self-protect	Manual detection and recovery from attacks and cascading failures.	Automated defense against malicious attacks or cascading failures; use early warning to anticipate and prevent system-wide failures.

Autogerenciamento em Sistemas Distribuídos:

O Modelo de Realimentação de Controle

Formado por componentes que precisam ser gerenciados

Associado a cada componente do *core*, fornece estimativas de desempenho do componente



Aplica o ajuste. Podem ser acionados vários mecanismos (componentes) para influenciar o comportamento do sistema: instalação de réplicas, prioridade de escalonamento, migração de dados, redirecionamento de requisições, ...

Implementa as políticas que decidem possíveis adaptações

Arquiteturas de Sistemas Distribuídos

Exercícios:

- Do livro de Tanenbaum e Van Steen: Problemas 1-5.
- Do livro de Couluris *et al.*: 2.1-2.8