Arquitetura de Sistemas Distribuídos

FABRÍCIO LUNA

Índice

Introdução

“Sistema Distribuído” – são frequentemente constituídos de partes complexas de software, componentes que estão dispersos por definição em múltiplas máquinas

...para dominar/controlar esta complexidade, é crucial que estes sistemas sejam devidamente organizados

... Esta organização é principalmente sobre os componentes de “software” que compões o sistema distribuído

Arquitetura de Software de Sistema Distribuído – nos informa como estes componentes de “Software estão organizados e como devem interagir entre si bem como outros componentes.

...concepção/implantação efetiva de um sistema distribuído requer a instanciação e associação de componentes de software em máquinas reais, o que pode ser feito de diferentes maneiras.

Introdução

... Instanciação final de uma **Arquitetura de Software** é comumente referenciada como **Arquitetura de Sistema.**

... Um dos objetivos de um sistema distribuído é separar aplicações da plataforma subjacente através da camada de “middleware”.

... Adoção de uma camada como esta é decisão arquitetural importante e o seu principal propósito é o de prover transparência de distribuição.

Estilos Arquiteturais

Estilo Arquitetural – define a forma como os componentes se **conectam** uns com os outros, como os dados são **trocados** entre os mesmos bem como são **configurados** conjuntamente com o sistema.

Componente – Unidade modular com interfaces bem definidas que são necessárias a sua operação e passíveis de substituição no seu ambiente.

Conector – Mecanismo responsável pela mediação da comunicação, coordenação e cooperação entre componentes.

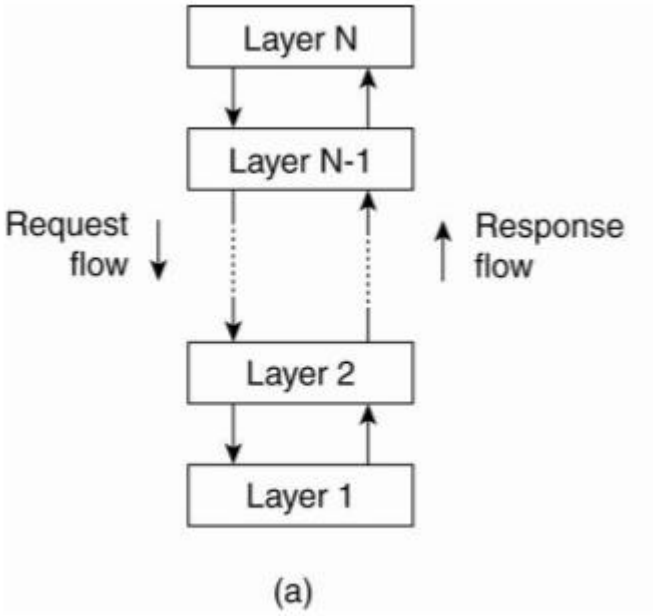
Ideia – Organizar o sistema em componentes logicamente diferentes e distribuir estes componentes em diversas máquinas.

Arquitetura em camadas

Componentes são organizados em camadas de modo que um componente da camada “N” invoque componentes da camada “N-1”, mas não outro fora dessa ordem.

Obs: O fluxo de controle passa de **camada em camada**, na ordem em que forma a estrutura hierárquica.

Largamente utilizado em redes de computadores e sistemas cliente-servidor



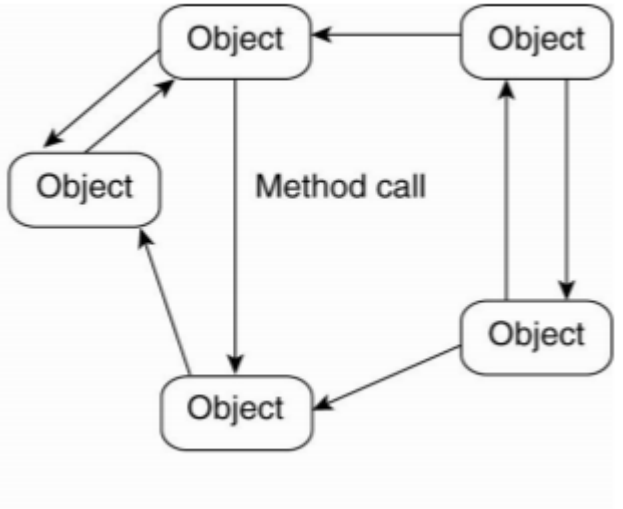
Arquitetura baseada em objetos

Cada componente é um **objeto** que se conectam uns aos outros através de mecanismos de **chamada de procedimento** (remoto).

Esta arquitetura combina com a arquitetura cliente-servidor já descrita.

Arquitetura baseada em objetos e camadas ainda formam os mais importantes estilos para a maioria dos sistemas de softwares.

Largamente utilizado em sistemas distribuídos



Arquitetura baseada em evento

Processos se comunicam essencialmente através da **programação de eventos**, que opcionalmente podem carregar dados.

Em sistemas distribuídos a programação de eventos vem geralmente associada com o que conhecemos por “publish/subscribe system”

Arquitetura baseada em evento

Principal vantagem do sistema baseado em eventos é o de os processos são **fracamente acoplados,** ou seja, a princípio não precisam referenciar um ao outro para se comunicarem.

Esta modalidade de troca de informações é referenciada como sendo **desacoplada no espaço** (“anonimato” entre objetos) ou “**Referentially desocoupled”.**

Arquiteturas baseadas em evento podem ser combinadas com Arquitetura Centrada em Dados – O que é conhecido por Espaço de dados compartilhado (desacoplamento no espaço e tempo)

Arquitetura Baseada em Evento

Essência – processos são desacoplados no espaço e também no tempo, ou seja, eles não precisam se encontrarem ativados quando a comunicação iniciar.

O que torna estas arquiteturas importantes para sistemas distribuídos é que elas tem também por objetivo alcançar a transparência de distribuição.

Arquitetura de Sistemas

Arquitetura de Software de Sistema Distribuído – nos informa como estes **componentes** de "software" estão **organizados** e como devem **interagir** entre si bem como outros componentes.

Instanciação final de uma Arquitetura de Software é comumente referenciada como Arquitetura de Sistema.

Arquitetura de Sistema – Decidir quais são os componentes de software, suas interações e localizações significa gerar uma instância de **Arquitetura de Software.**

Dentre as abordagens possíveis, destacam-se: Arquitetura Centralizada, Descentralizada e Híbrida

Arquiteturas Centralizadas

Modelo Cliente/Servidor: Processos em um sistema distribuído são divididos em 2 grupos - os que oferecem serviços (servidores) e os que usam serviços (clientes)

Clientes e servidores espalhados na rede interagem através de mensagens tipo “request/reply”

Arquiteturas Centralizadas

Considerando que mensagens “request/reply” não sejam perdias ou danificadas, o protocolo “request/reply” funciona bem mesmo sobre um protocolo orientado a conexão (LAN);

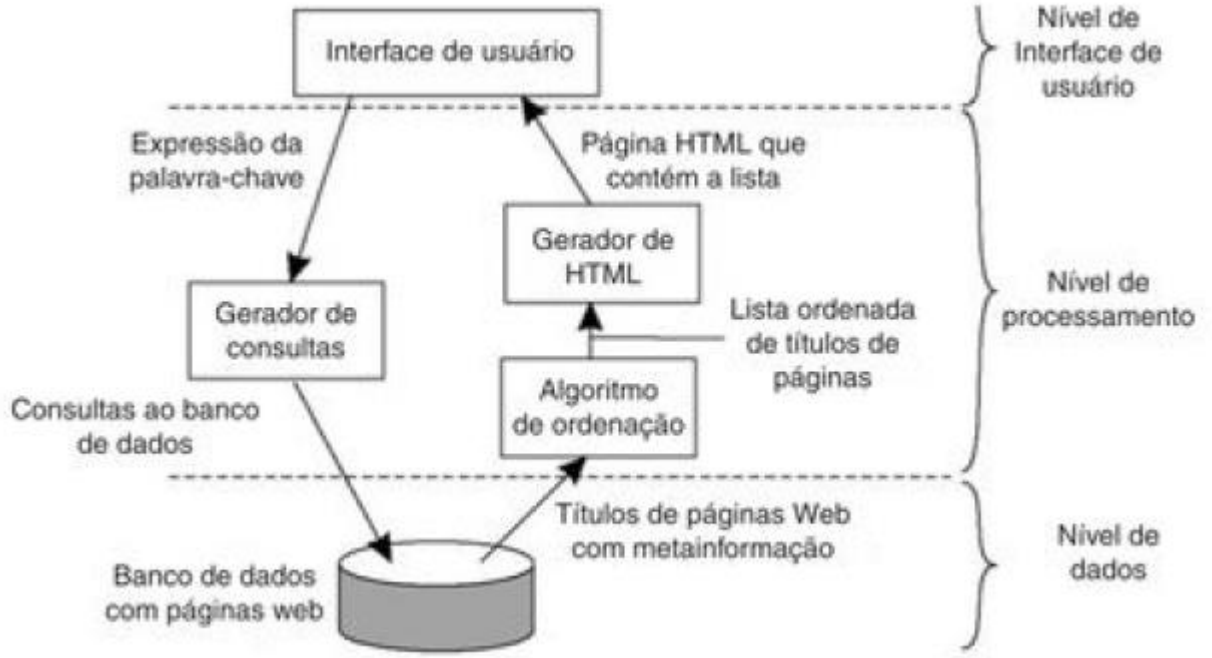
Infelizmente, tornar o protocolo resistente a falhas ocasionais de transmissão não é trivial – cliente não pode detectar qual das mensagens se perdeu (“Request” ou “reply”)

Operação **idempotente –** pode ser repetida inúmeras vezes sem prejuízos, mas por outro lado, nem todas as mensagens do tipo “request” são idempotentes => não há solução simples para tratar a perda de mensagens.

Alternativa – Utilização de protocolo orientado a conexão confiável, o que por outro lado pode comprometer a performance.

Aplicação em Camadas

Considerando que muitas das Aplicações Cliente/Servidor estão direcionadas para acesso a banco de dados, há um consenso de que o estilo arquitetural é o de camadas.



Aplicação em Camadas

Nível Interface do usuário – contém o que é necessário para interagir com o usuário, ou seja, define a interface de comunicação;

Nível de processamento – contém a aplicação (sem os dados)

Nível de Dados – contém os dados que o cliente quer acessar.

Observação: estas camadas são encontradas em muitos sistemas distribuídos de informação e utilizam tecnologia tradicional de banco de dados e de aplicações.

Arquitetura em Multicamadas

Conclusão – distinção em 3 níveis lógicos, sugere inúmeras possibilidades para distribuir a aplicação ou partes do cliente/servidor sobre diferentes máquinas.

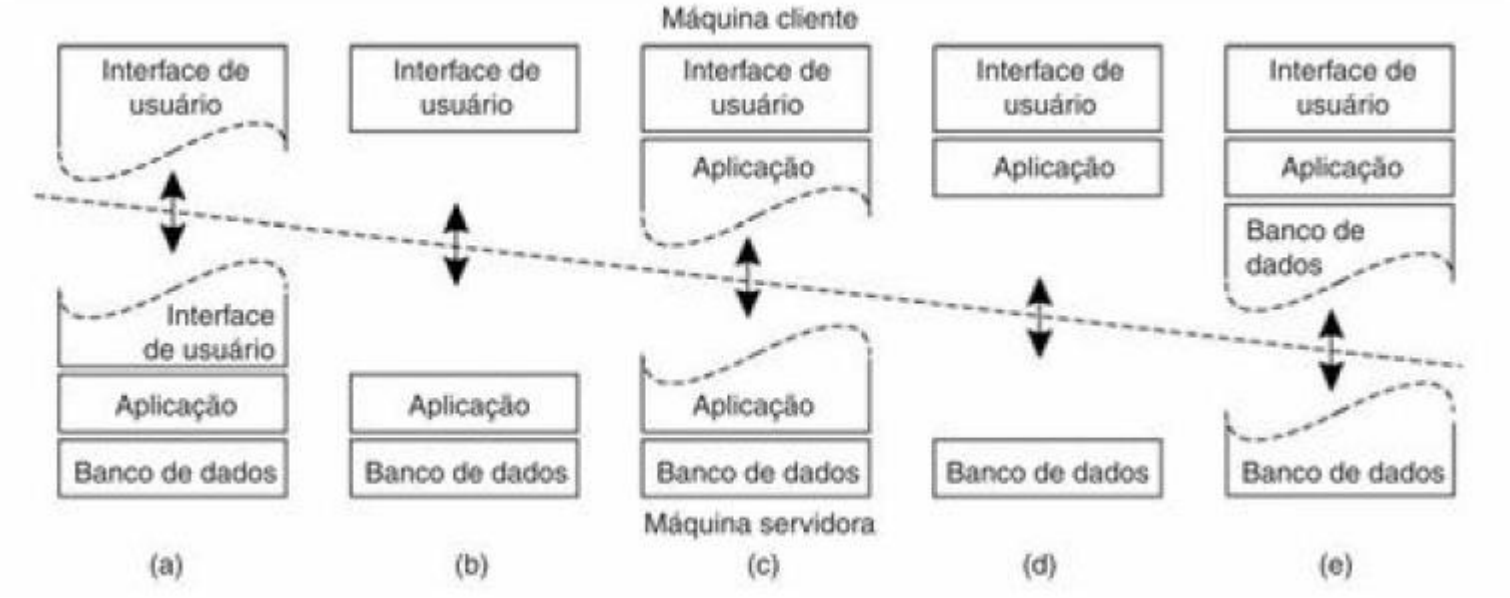
Organização mais simples é a que temos 2 tipos:

> **Máquina cliente**: contém somente programas implementando o nível de interface do usuário ou parte desse nível
>
> **Máquina servidor –** contém programas que implementam os níveis de processamento e de dados.

Nesta abordagem, praticamente tudo é manipulado pelo servidor enquanto o cliente é praticamente um terminal “burro” com uma interface gráfica provavelmente.

Arquitetura em Multicamadas

Uma outra abordagem consiste em distribuir os programas na camada de aplicação em diferentes máquinas como mostrado na figura abaixo:



Arquitetura em multicamadas

Da perspectiva de gerenciamento de sistemas “clientes gordos” não são a melhor escolha, ao invés disso, “clientes magros” são mais fáceis.

Distinção entre 2 tipos de máquinas (máquina cliente e máquina servidor) é conhecido como “**Arquitetura de duas divisões**.

Um servidor as vezes pode agir como cliente, o que resulta em uma “**Arquitetura de três divisões”**

Arquitetura Descentralizadas

Arquiteturas Multicamadas são uma consequência direta da divisão de aplicações em interface do usuário, componente de processamento e nível de dados

Distribuir o processamento é equivalente a organizar a aplicação cliente/servidor em arquitetura multicamada – o que é conhecido por **Distribuição vertical.**

Quando o cliente ou o servidor são divididos em partes logicamente equivalentes e cada parte opera no seu próprio conjunto de dados, temos a **Distribuição horizontal.**

Nesse contexto, destacamos as arquiteturas modernas de sistemas, também conhecidas por "**peer-to-peer systems"**

Arquiteturas descentralizadas

Sistemas P2P podem ser classificados em>

- **Arquitetura P2P Estruturada** – nós organizados seguindo uma estrutura de distribuição de dados específica, ou seja, procedimento determinístico,
- **Arquitetura P2P não Estruturada -** nós tem vizinhos escolhidos de forma aleatória, ou seja, algoritmos randômicos são utilizados.

Observação: Virtualmente em todos os casos estamos lidando com "**Rede de sobreposição"** , ou seja, redes em que os nós são formados por processos e os "links" representam os possíveis canais de comunicação (ex: conexões TCP)

Arquitetura Peer-to-Peer Estruturada

Ideia Básica – nós são organizados em uma rede de sobreposição estruturada com um anel lógico permitindo a distribuição de serviços no nós bem como a sua identificação

Tabela de hash distribuída (DHT) – itens de dados são identificados por uma chave aleatória de um espaço amplo, tal como 128 ou 160bits.

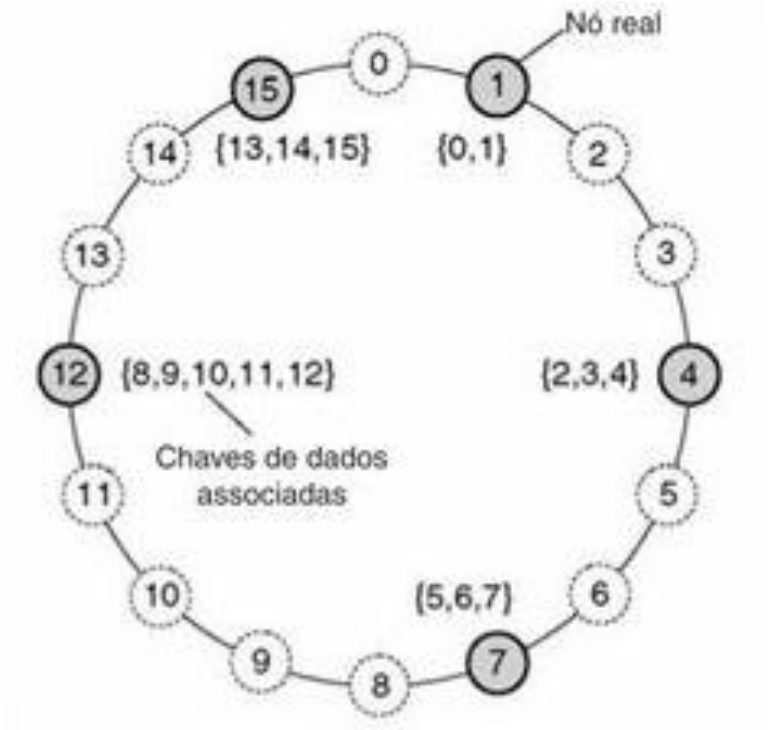
Adicionalmente, nós dos sistema são identificados por nós aleatórios dentro do mesmo espaço de identificadores;

Problemas em DHT – implementar um esquema eficiente e determinístico para mapear a chave do item de dado ao identificador do nó baseado em alguma métrica de distância

Arquitetura Peer-to-Peer Estruturada

Sistema provê operação de “LOOKUP(key)”

Requisições são roteadas para o nó associado



Arquitetura Peer-to-Peer Estruturada

Chord System – nós são organizados como um anel de modo que o item de dado com chave “k” esteja mapeado para o nó cujo menor identificador “id” seja maior ou igual a k (>=k)

Quando um nó deseja juntar-se ao sistema, ele inicia o processo gerando um identificador aleatório “id”

Considerando que o espaço de identificadores é grande o bastante, a probabilidade de gerar um identificador atribuído a algum nó atual é próximo de zero

Nó referenciado como sucessor da chave k e denotado por SUCC(k) pode ser obtido através da função LOOKUP(k)

Naturalmente que este esquema possibilita a cada nó o armazenamento de seus predecessores.

Arquitetura Peer-to-Peer Estruturada

Outras abordagens também são baseadas no DHT – “**Rede de conteúdo endereçável” (CAN)**

CAN – dispões os nós em um espaço de dimensão “d” de tal forma que cada nó seja responsável por dados em uma região específica.

Todo item de dado na rede endereçada por conteúdo é identificado por um único ponto de espaço bem como pelo nó responsável por aquele dado.

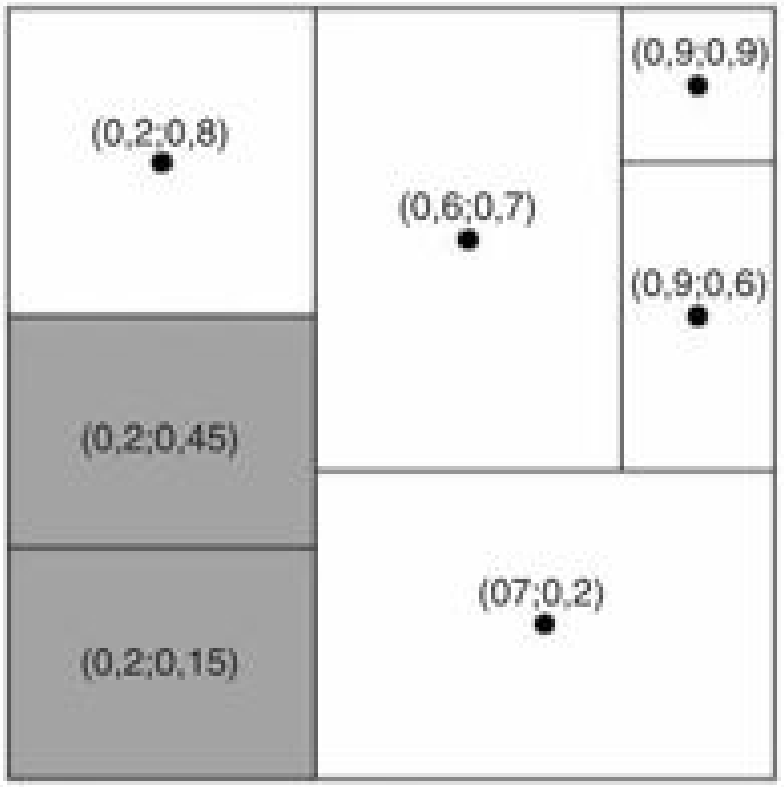
Quando um nó é adicionado => região é dividida

Arquitetura Peer-to-Peer Estruturada

Mapeamento de Itens de dados em nós em uma rede

Arquitetura Peer-to-Peer Estruturada

Divisão de região quando da adesão de um nó



Arquitetura Peer-to-Peer Não estruturada

Arquitetura P2P não estruturada - cada nó mantém uma lista de “c” vizinhos, onde idealmente cada um dos vizinhos representa uma escolha aleatória do conjunto de nós ativos

Se assumirmos que cada nó troca regulamente entradas da sua visão parcial, cada entrada identifica um outro nó na rede bem como quão velha/antiga é a referência para aquele nó

Assim há a necessidade de ao menos 2 “threads” como detalhado nos 2 algoritmos a seguir:

- Ações para “thread” ativa
- Ações para “thread” passiva

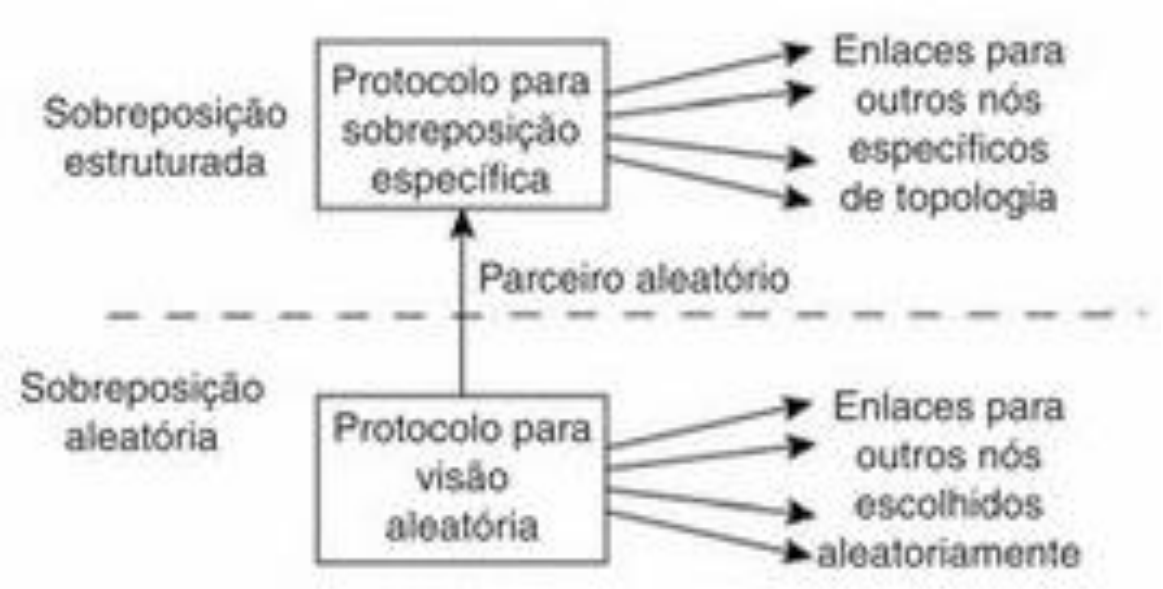
Thread Ativa x Passiva

É com vocês!

Topologia de Gerenciamento em Redes de Sobreposição

Embora possa parecer que Sistemas Peer-to-Peer Estruturados e não estruturados formem classes independentes, não é o caso.

Com a seleção ou troca cuidadosa de visões parciais é possível construir e manter uma topologia específica



Topologia de Gerenciamento em Redes de Sobreposição

É perceptível que a localização de itens de dados pode se tornar um problema a medida que as redes não estruturadas crescem

pois não há um meio determinístico de pesquisa, o que por outro lado pressupõe que a técnica a ser utilizada é a técnica de **inundação de requisições.**

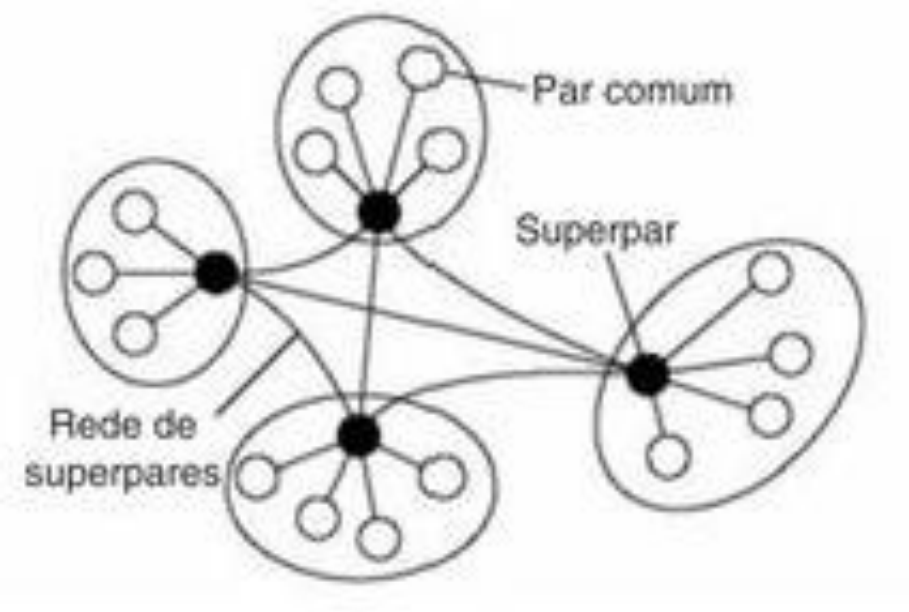
Há várias maneiras nas quais a Inundação de requisições pode ser representada, mas uma alternativa é a utilização de nós especiais que mantém um índice de itens de dados.

tais estruturas que mantém um índice ou agem como um interceptador são geralmente referenciadas como “super nós”

Topologia de Gerenciamento em Redes de Sobreposição

Organização hierárquica de nós dentro da rede de super nós, onde um nó regular está conectado como cliente a um super nó.

Em muitos casos, a relação cliente – super nó é fixa, ou seja, quando um nó regular se junta a rede, ele se conectar a um super nó e permanece conectado até deixar a rede.



Arquiteturas Hibridas

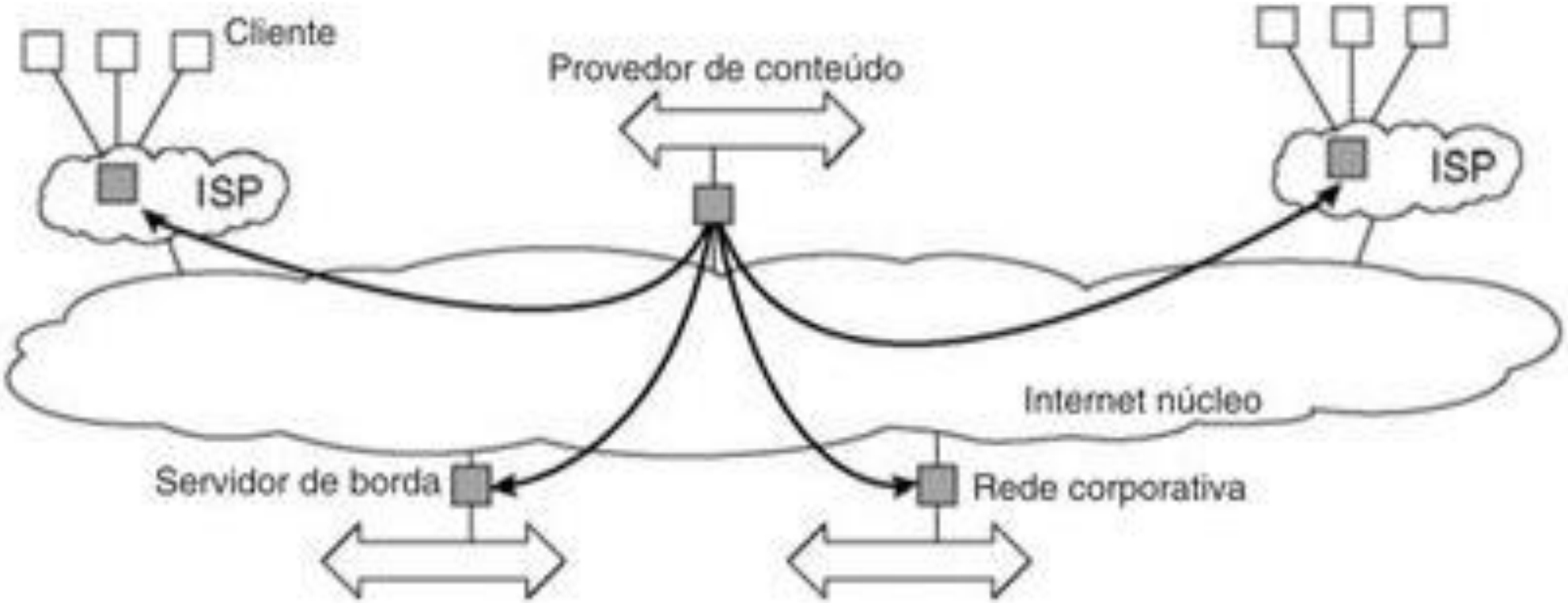
Muitos sistemas distribuídos combinam aspectos arquiteturais e são assim referenciados como sistemas de arquiteturas híbridas

Uma importante classe de sistemas distribuídos organizados de acordo com uma arquitetura híbrida é formada pelos sistemas servidores de borda

Sistema Servidor de Borda: Sistemas são implantados na rede onde servidores estão localizados na borda da rede

- Esta borda é formada pelo limite da rede corporativa e a rede internet. EX: Como previsto pelo ISP (internet Service Provider)
- Também quando usuários finais se conectam a internet através do seu ISP, ou seja, o ISP pode ser considerado como residente de borda de rede

Arquiteturas Hibridas



Arquiteturas Híbridas

Coleção de servidores de borda podem ser usados para otimizar a distribuição de aplicações e conteúdo.

Ou seja o modelo básico é o de que para uma organização específica, cada servidor de borda se comporte como um servidor origem do qual todo o conteúdo se origina.

EX: soluções distribuídas baseadas na WEB

Arquiteturas Híbridas

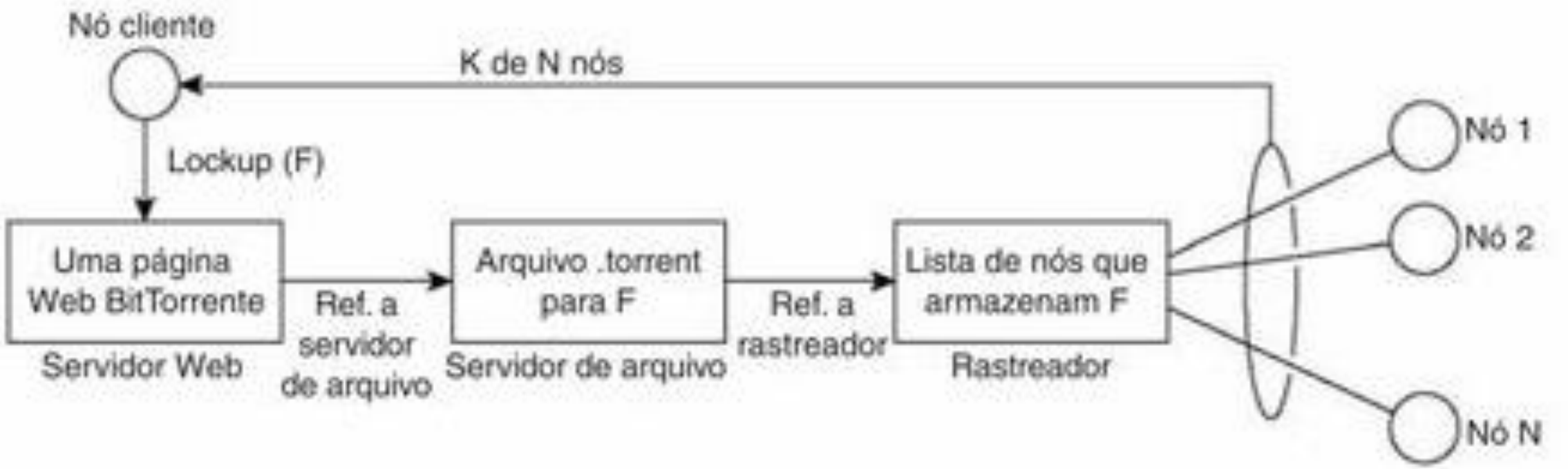
Sistemas Colaborativos Distribuídos – uma vez que o nó tenha se juntado ao sistema, ele pode utilizar um esquema totalmente descentralizado para colaboração.

EX: para maior percepção, considere o BitTorrent – Sistema de download de arquivo p2p que combina soluções centralizadas e descentralizadas

Ideia Básica: Quando um usuário **busca um arquivo**, ele **descarrega pedaços** – “chunks” do arquivo de um outro usuário até que os pedaços descarregados possam ser juntados para produzir o arquivo original

Um importante aspecto de projeto presente nesta proposta é a **garantia de colaboração** durante todo o processo.

Arquiteturas Híbridas



Arquiteturas Híbridas

Para efetuar o “download” de um arquivo, usuário acessa um **diretório global**, que é um dos “web sites” bem conhecidos

Este diretório contém **referências** para o que são denominados arquivos “.torrent” - que contém informações necessários para o “download” do referido arquivo – arquivo específico;

“Torrent” referencia o que é conhecido por **“tracker”,** ou seja, um servidor que mantém precisamente quais nós ativos contém pedaços - **“chunks”** do arquivo requisitado;

Um nó ativo é por exemplo que está efetuando o “**downloading**” um outro arquivo.

Obs. - geralmente são diferentes “**trackers**”, embora tenha-se um “**tracker**” por arquivo ou por uma coleção de arquivos

Arquiteturas Híbridas

Um vez identificado de onde os “**chunks**” podem ser **descarregados**, o nó que solicitou o “download” é efetivamente ativado;

Neste ponto ele será forçado a **ajudar** outros, p.ex., **oferecendo** “**chunks**” do arquivo que já tenham sido descarregados para outros usuários que ainda não tenham estes pedaços.

Este comportamento vem de uma **regra** muito simples: se nó P percebe que o nó Q está recebendo mais do que está fornecendo, P pode decidir decrementar a taxa na qual envia dados para Q;

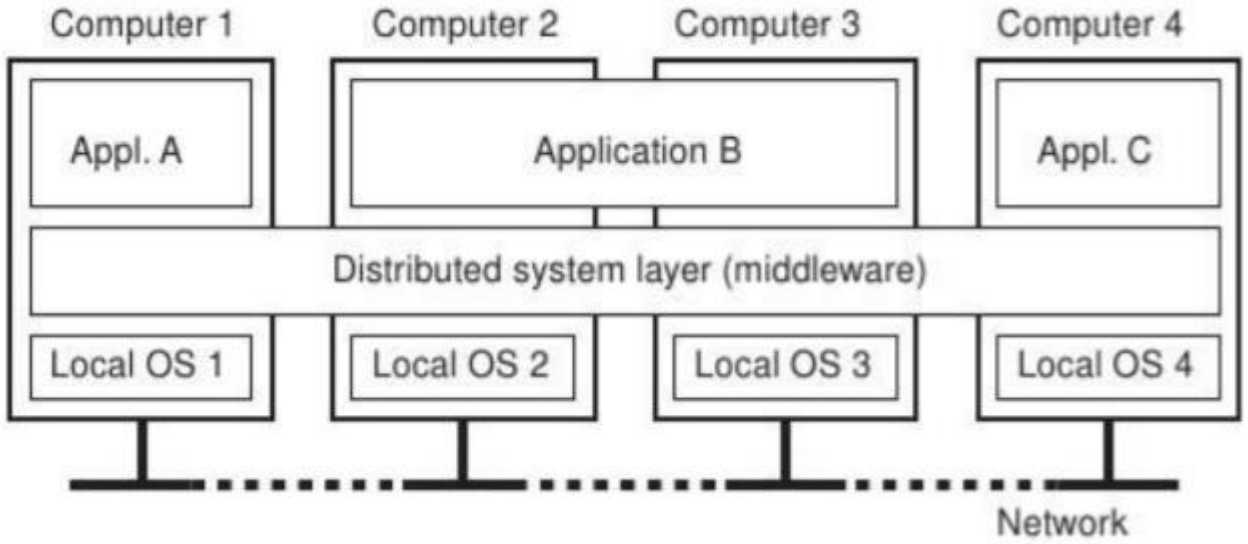
Este esquema funciona bem desde que P tenha algo para descarregar do nó Q.

Por esta razão, nós são referenciados por **muitos** outros **nós**, permitindo que tenham condições para negociar dados.

Arquitetura X Middleware

Middleware – “Software” que oferece serviços para as aplicações tendo por base serviços disponíveis no sistema operacional.

Middleware NÃO é parte do sistema operacional ou do gerenciamento do sistema, nem mesmo das aplicações.



Arquitetura vs Middleware

Desenvolver “middleware” segundo uma padrão arquitetural traz o **benefício** de tornar as **aplicações distribuídas simples**, mas por outro lado, o **desempenho** pode ser **comprometido** em relação ao que o desenvolvedor tinha em mente.

Embora o “middleware” seja o meio para prover transparência de distribuição, percebe-se que **soluções** específicas devem ser **adaptáveis** aos requisitos da aplicação;

Arquitetura vs Middleware

Solução – Disponibilizar várias versões do middleware de modo que cada versão seja adaptada para uma classe específica

Alternativa – Uma abordagem mais interessante é permitir que o middleware seja fácil de ser configurado, adaptado e personalizado como exigido por uma aplicação.4

Conclusão – Sistemas vem sendo desenvolvidos com separação mais rigorosa entre políticas e mecanismos que possibilitem que o comportamento do middleware seja modificado.

Interceptadores

Interceptador – construção em software que **intercepta** um fluxo usual de controle e aciona um outro código para ser executado normalmente uma aplicação específica

Decisão de implementar um interceptador – Que exige esforço substancial - não é clara quando comparada com a simplicidade e aplicabilidade restrita de uma versão de “middleware”

Em muitos casos, ter funcionalidade de interceptação ainda que limitadas irá melhorar o gerenciamento do software bem como do sistema distribuído como um todo.

Para tornar o problema mais concreto, considere a invocação de um método do objeto “B” por um objeto “A”

Interceptadores

EX: objeto A pode invocar um método do objeto B ainda que B esteja em outra máquina, ou seja, a invocação do objeto se dá em 3 passos

Objeto A disponibiliza uma interface local que é exatamente a mesma que a interface disponibilizada pelo objeto “B”, assim “A” simplesmente invoca o método disponível naquela interface;

chamada de “A” é transformada em um invocação genérica de objeto, possivelmente através de uma interface de invocação de objeto genérica oferecida pelo “middleware” na máquina onde “A” reside;

Finalmente, a invocação genérica de objeto é transformada em uma mensagem que é enviada pela interface da camada de transporte do sistema operacional de rede da máquina onde “A” reside

Interceptadores

Após o primeiro passo, a chamada “B.do_something(value)” é transformada em uma chamada genérica tal como “invoke(B, &do_something, value)” com a referência do método B bem como os parâmetros da chamada do método;

Considere o cenário que o objeto “B” tenha sido replicado, neste caso cada réplica deveria ser invocada.

Este é um ponto onde o interceptador pode ajuda !!

“Intercptor” irá chamar “invoke(B, &do_something, value)” para cada uma das réplicas sem que “A” saiba da existência das réplicas de “B” ou mesmo tenha que gerenciá-las;

Somente o “interceptor” no nível de requisição que foi adicionado ao “middleware” necessita saber de “B” e “ B' ”

Interceptadores

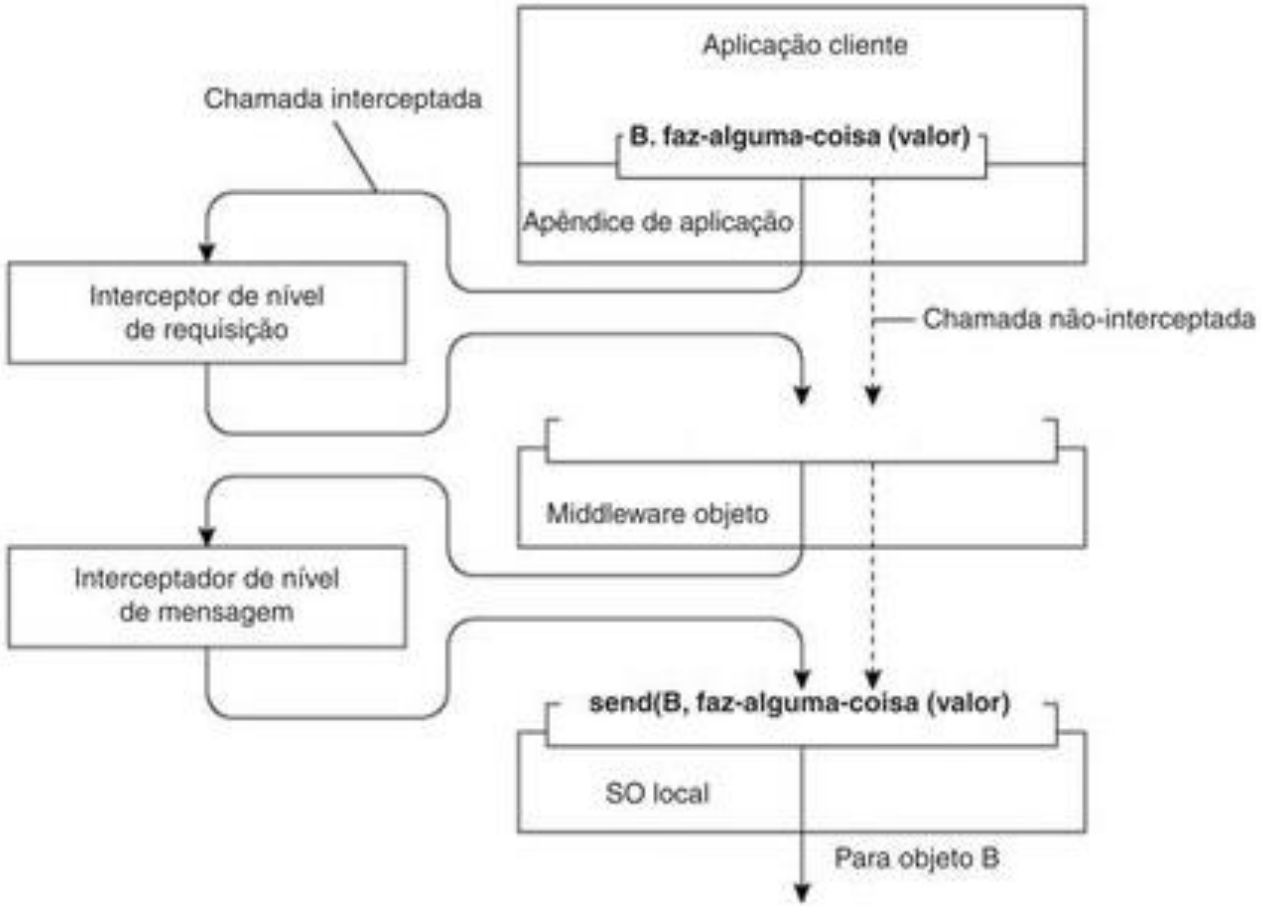
Ao sair do “middleware” a chamada para o objeto remoto deve ser enviada para a rede, ou seja, interface de mensagem do sistema operacional de rede local deve ser invocada;

Neste nível, o “interceptor” no nível de mensagem pode auxiliar a transferência de invocação para o objeto alvo.

EX: parâmetro “value” corresponde a um array grande de dados, assim, é mais inteligente fragmentá-lo em partes menores e remontá-las no destino.

Middleware não está a par desta fragmentação, mas o “interceptor” no nível de mensagem se encarrega de forma transparente tratar a parte final da comunicação com o SO

Interceptador



Software Adaptativo

Interceptador – Constituem-se no meio para adaptar o middleware

Esta necessidade vem do fato de que o ambiente no qual aplicações distribuídas são executadas mudam continuamente

Assim, muitos projetistas de middleware passaram a considerar a construção de softwares adaptativos.

MicKinley et al. (2004) propões três técnicas básicas para adaptação de software

- Separação de responsabilidades
- Computação Reflexiva
- Projeto baseado em componentes

Software Adaptativo

Separação de Responsabilidades - Está relacionado ao modo tradicional de modularização de sistemas.

- Ou seja, separa as partes que implementam funcionalidades de outras partes que são especializadas em outras funcionalidades tais como: confiabilidade, desempenho, segurança etc.

Computação Refletiva – Habilidade de um programa inspecionar a si mesmo e, se necessário, adaptar-se.

- Normalmente contemplada na linguagem de programação

Projeto baseado em componentes – Oferece suporte à adaptação através da composição, permitindo que um sistema seja configurado estaticamente em tempo de concepção do projeto ou dinamicamente em tempo de execução.