Multiface refere-se a sistemas que oferecem múltiplas interfaces ou formas de interação para diferentes tipos de usuários ou dispositivos.

Estilos arquitetônicos

Um estilo é formulado em:

- Componentes (Substituíveis) com interfaces bem definidas.
- A maneira como os componentes estão conectados uns aos outros.
- Os dados trocados entre os componentes.
- Como esses componentes e conectores são configurados em conjunto em um sistema.

Conector

Um mecanismo que medeia a comunicação, coordenação ou cooperação entre componentes.

ex: recursos para chamada de procedimento (remoto), mensagens ou streaming.

Camada de aplicativos

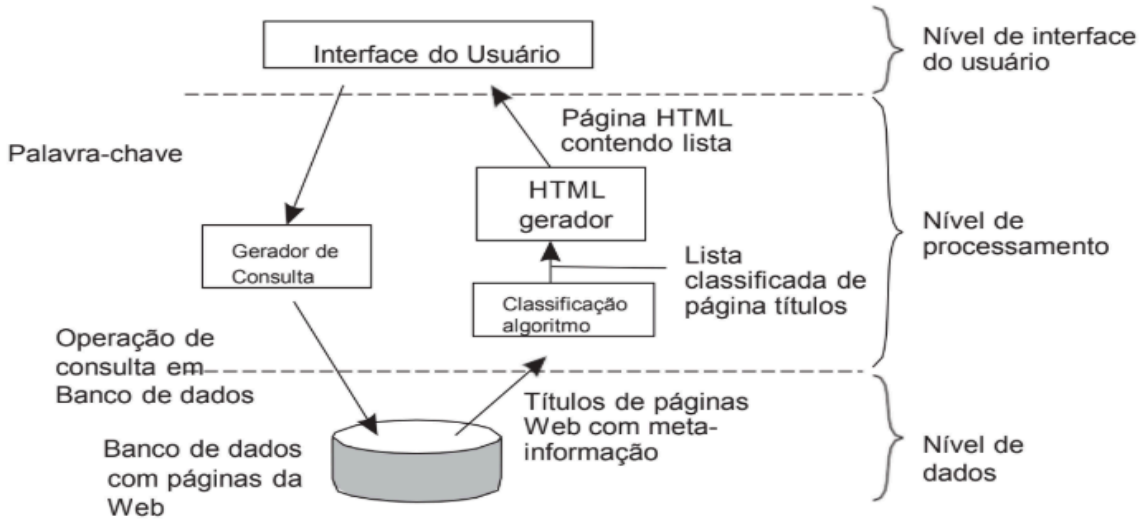
Camada de interface de aplicação - Contém unidades para interfaces com usuários ou aplicativos externos.

Processamento camada - Contém funções de um aplicativo, ou seja, sem dados específicos.

Camada de dados - Contém os dados que um cliente quer manipular através dos componentes da aplicação.

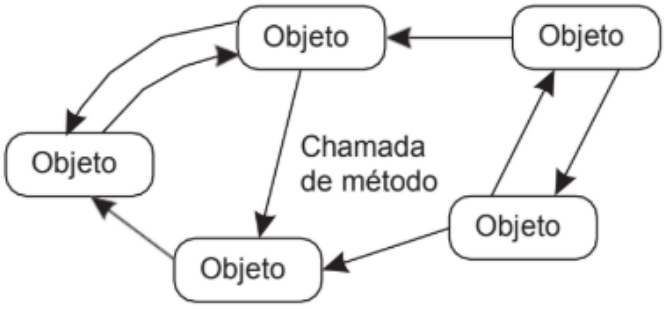
Observação

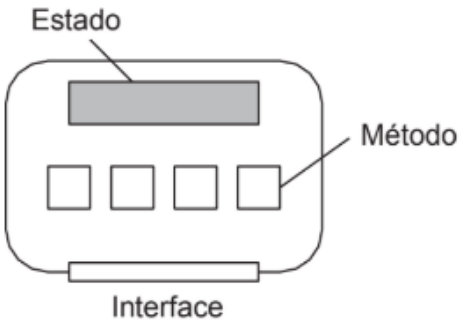
Esse modelo é encontrado em muitos sistemas de informação distribuídos, usando tecnologia de banco de dados tradicional e os aplicativos que a acompanham.



Baseado em objeto estilo

Essência - Componentes são objetos, conectados para cada outro através de procedimentos chamados. Os objetos podem ser colocados em diferentes máquinas chamadas podem por isso executar através de uma rede.





Encapsulamento - Os objetos encapsulam dados e oferecem métodos sobre esses dados sem revelar a implementação interna.

Arquiteturas RESTful

Exibir um sistema distribuído como uma coleção de recursos, gerenciada individualmente por componentes. Os recursos podem ser adicionados, removidos, recuperados e modificados por aplicativos (remotos).

1. Os recursos são identificados por meio de um único esquema de nomenclatura.
2. Todos os serviços oferecem a mesma interface.
3. As mensagens enviadas de ou para um serviço são totalmente descritas por ele mesmo.
4. Depois de executar uma operação em um serviço, esse componente é tudo sobre o chamador(stateless).

Operação	Descrição
PUT	Criar um novo recurso
GET	Recuperar o estado de um recurso em alguma representação
DELETE	Excluir um recurso
POST	Modificar um recurso transferindo novo estado

Exemplo: Amazon simples armazenar serviço

Objetos (ou seja, arquivos) são colocados em buckets (ou seja, diretórios). Os baldes não podem ser colocados em baldes. As operações em ObjectName no bucket BucketName exigem o seguinte identificador:

http://BucketName.s3.amazonaws.com/ObjectName

Operações comuns

Todas as operações são realizadas enviando solicitações HTTP.

- Crie um bucket/objeto: PUT, junto com o URI
- Listando objetos: GET em um nome de bucket.
- Lendo um objeto: GET em um URI completo.

Nas Interfaces

A abordagem RESTful para um serviço é muito simples visto que o problema precisa ser feito no espaço de parâmetros.

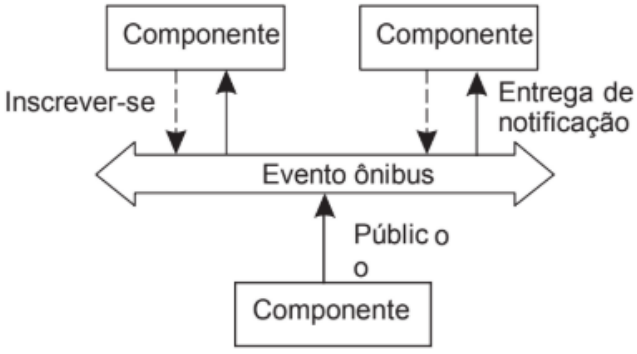
ex: Suponha que um bucket de interface ofereça uma operação crate, exigindo uma string de entrada como mybucket, para criar um bucket.

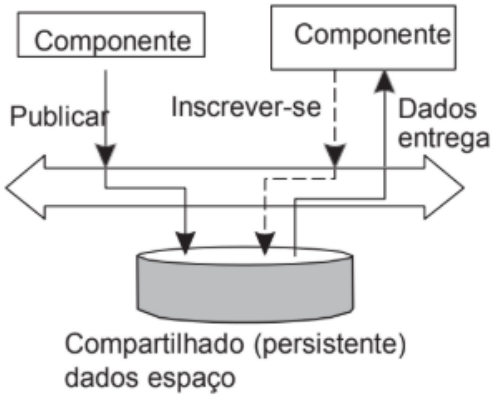
Coordenação

Acoplamento temporal e referencial

	Temporalmente acoplado	**Temporalmente desacoplado**
Referencialmente acoplado	Direto	Caixa de correio
Referencialmente desacoplado	Evento-baseado	Compartilhado dados espaço

Espaço de dados compartilhado e baseado em eventos





Exemplo: Espaço Linda Tupley

- int(t): Remova um modelo de correspondência de tupla T.
- rd(t): Obter cópia de um modelo de tupla T.
- out(t): Adicionar tupla t para o espaço da tupla.

Mais detalhes

- Chamar (t) duas vezes seguidas leva ao armazenamento de duas cópias da tupla.
- T -> um espaço de tuplas é modelado como um multiconjunto. Ambos em E rd são operações bloqueadas: o chamador vai ser bloqueado até que uma tupla correspondente seja encontrada , ou se torne disponível.

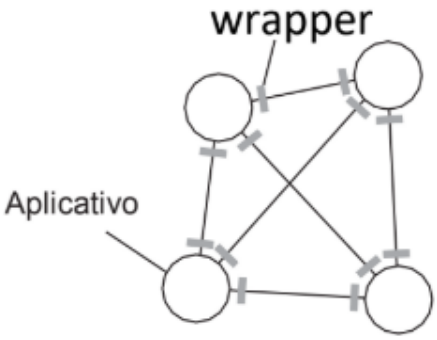
Middleware construído de forma legada

Problema - As interfaces oferecidas por um componente legado são a maioria. Provavelmente não é adequado para todas as aplicações.

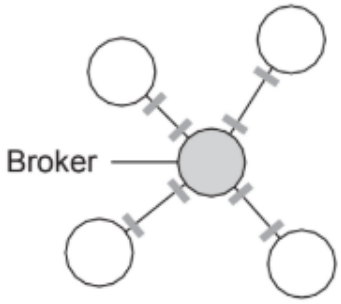
Solução - Um wrapper ou adaptador oferece uma interface aceitável para um cliente aplicativo. Isso é funções são transformadas em aqueles disponíveis no componente.

Organizando wrappers

Duas soluções: 1-a-1 ou através de um broker



Wrappers em sistemas distribuídos são componentes ou camadas de software que encapsulam a lógica de uma funcionalidade ou serviço para facilitar sua integração, reutilização, ou abstração. Em um sistema distribuído, eles são especialmente úteis para lidar com a complexidade de comunicação e coordenação entre diferentes componentes que podem estar em máquinas distintas.



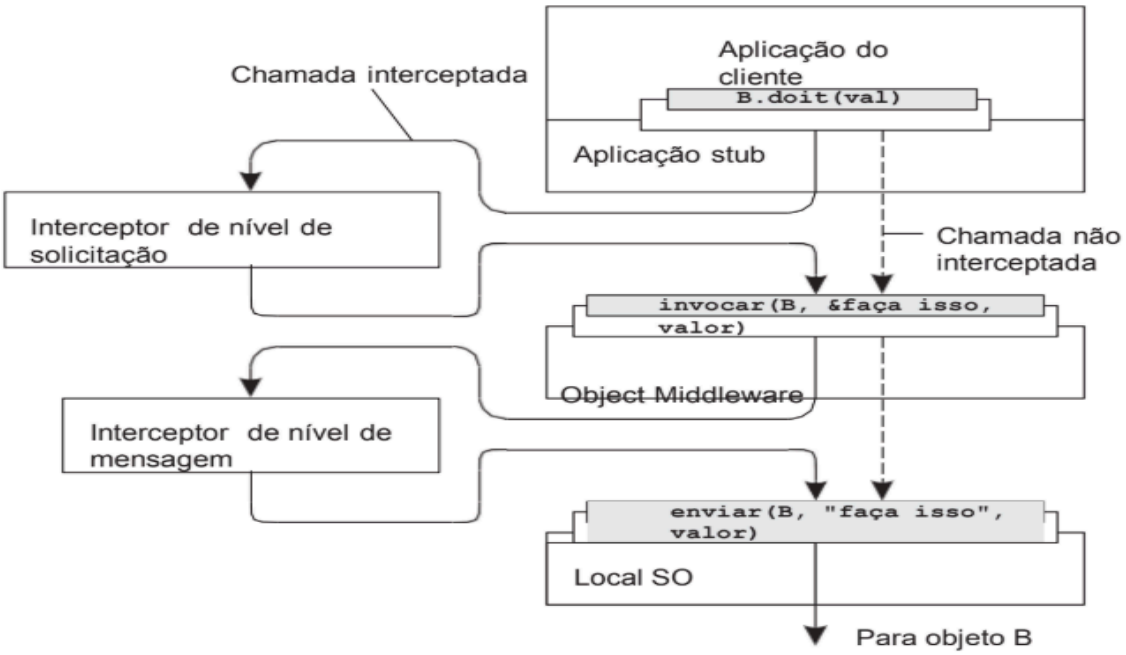
Complexidade com N aplicações

- **1-para-1:** Requer N X(N -1) = O (N^2) wrappers
- **Corretor**: Requer 2 N = O (N) wrapper.

Desenvolvimento adaptável no middleware

Middleware contém soluções que são boas para maioria das aplicações => Necessita-se adaptar seu comportamento para aplicações específicas.

Interceptar o fluxo comum de controle



Arquitetura de sistema centralizado

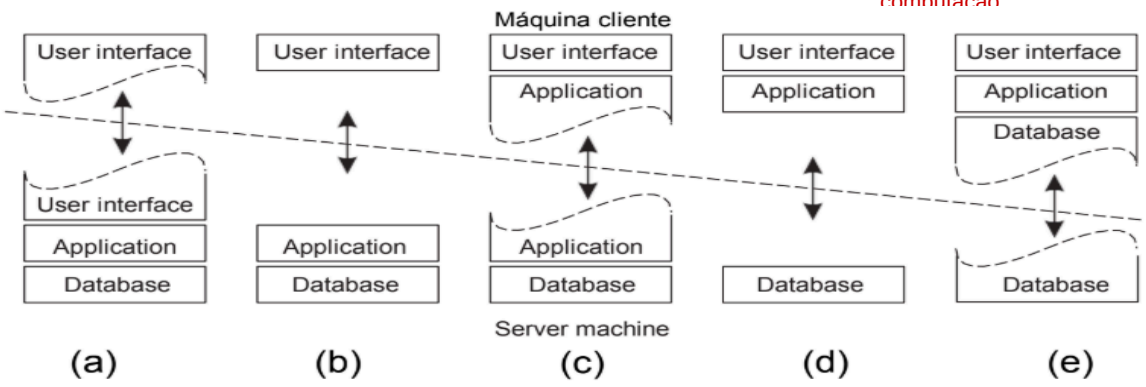
Básico cliente-servidor modelo:

- Existem processos que oferecem serviços (servidores).
- Existem processos que usam serviços (clientes).
- Clientes e os servidores podem estar em máquinas diferentes.
- Clientes seguem o modelo de solicitação/resposta ao uso de serviços.

Arquitetura centralizadas de sistemas multicamadas

- Camada única: Terminal burro/configuração mainframe.
- Duas camadas: Cliente/simples configuração do servidor.
- Três níveis: Cada camada em máquina separada.

Um terminal burro (em inglês, dumb terminal) é um dispositivo que serve apenas como uma interface de entrada e saída para um computador central (mainframe ou servidor). Ele não possui capacidade de processamento ou armazenamento local e depende inteiramente do sistema central para realizar qualquer tipo de computação



Organizações alternativas

- **Distribuição vertical -** Originou da divisão de aplicativos distribuídos em três camadas lógicas e da execução dos componentes de cada camada em um servidor (máquina) diferente.
- **Distribuição horizontal -** Um cliente ou servidor pode ser fisicamente devido em partes logicamente equivalentes, mas cada parte está operando em seu próprio compartilhamento do conjunto de dados completo.
- **Arquitetura peer-to-peer -** Os processos são todos iguais: As funções que precisam ser executadas são representadas por todos os processos => cada processo atuará como cliente e servidor ao mesmo tempo (ou seja, atuando como servidor).

Peer-to-peer estruturado

Faz o uso de um índice semântico livre: cada item de dados é associado exclusivamente a uma chave, por sua vez usada como um índice. Prática comum: use uma função hash.

Key(data item) = hash(data item´s value).

O sistema p2p agora armazena pares.

Exemplo: Coro

- Os nós são organizados logicamente em um anel. Cada nó tem um identificador de m-bit.
- Cada item de dados é hash para uma chave m-bit.
- O item de dados com chave k é armazenado no nó com a menor indetificação de id > k, chamada de sucessora de chave k.
- O anel é estendido com vários links de atalho para outros nós.

P2P não estruturado

Cada nó mantém uma lista ad hoc de vizinhos. A sobreposição resultante se assemelha a um grafo aleatório: Uma aresta (u, v) existe apenas com uma certa probabilidade P[(u, v]).

Tipos de estratégia de busca

- **Inundação -** O nó emissor U passa a solicitação de d para todos os vizinhos. A solicitação é ignorada quando o nó de recebimento é visto antes. Caso contario, v procura localmente por d (recursivamente). Pode ser limitado por um Time-To_Live: Um número máximo de saltos.
- **Caminhada aleatória -** O nó emissor U passa a solicitação de D para o vizinho escolhido aleatoriamente, v. Se v não ter d, ele encaminha a solicitação para um de seus vizinhos aleatoriamente e assim por diante.

Redes superpares

As vezes, é sensato quebrar a simetria em redes peer-to-peer puras:

- Ao pesquisar em sistemas P2P não estruturados, ter servidores de indexação melhora o desempenho.
- A decisão de onde armazenar dados geralmente pode ser feita com mais eficiência por meio de brokers.