Padrões Arquiteturais

Padrões arquiteturais são formas conhecidas de se estruturar os componentes de um sistema. São semelhantes aos padrões de projeto, mas seu escopo é o sistema como um todo, ou um subsistema.Também chamados de *estilos* arquiteturais

Padrões diferem uns dos outros: no tipo de componentes, no tipo de conectores, na forma de conexão.

1-Padrões para sistemas de fluxo de dados

A disponibilidade dos dados controla a computação. A estrutura baseia-se na transferência ordenada de dados entre componentes. Não há outra forma de interação entre componentes.

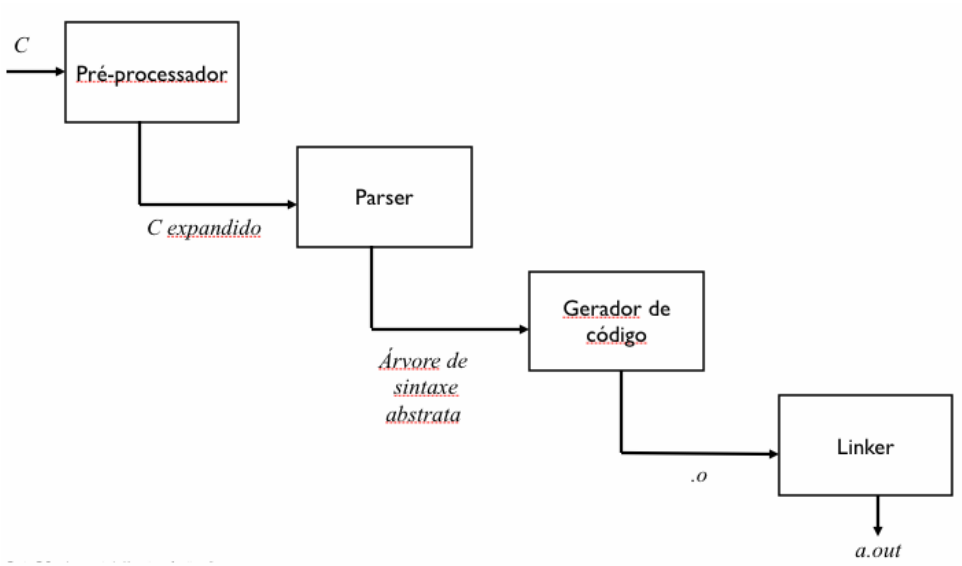
Variações:

Batch sequencial

Esse padrão representa uma das maneiras mais antigas de se executar um sistema. Consiste da execução de uma sequência de programas, com os dados sendo transmitidos de um programa para o outro.

Cada programa executa completamente antes do programa seguinte começar.

Exemplo: Compilador



Limitações:

No padrão batch sequencial, os programas não são interativos. O dado entra em uma ponta do processamento e sai na outra sem qualquer interação com o usuário.

Camadas

- Cada componente (camada) se comunica apenas com componentes adjacentes.
- Camadas implementam abstrações em diversos níveis.
- Equivalente a pipes-e-filtros com comunicação bidirecional onde os pipes não existem.

Exemplo 1: Padrão ISO/OSI de redes

Exemplo2 : Arquitetura em 3 camadas

Camada de apresentação

Composta de classes que constituem a funcionalidade para visualização dos dados pelos usuários e interface com outros sistemas (classes de fronteira)

Camada da lógica do negócio (ou da aplicação)

Composta de classes que implementam as regras do negócio no qual o sistema está para ser implantado (classes de controle)

Camada de acesso

Contém as classes que se representam as entidades que são persistentes no banco de dados (classes de entidade persistente)

Características:

- No sentido mais estrito, cada camada só provê serviços para a camada imediatamente superior e usa os serviços da camada imediatamente inferior através de uma interface bem definida.
- Em um sentido mais amplo, camadas distantes podem se comunicar através de uma ponte. Isso reduz a portabilidade e a flexibilidade.
- Uma camada representa um todo coerente e não precisa saber como as outras camadas estão implementadas. Ela depende apenas das interface das camadas adjacentes.

Benefícios:

- Uma camada pode ser substituída por outra que implemente a mesma interface sem nenhum prejuízo para o sistema.
- Este padrão não possui as limitações presentes no pipes e filtros relativas ao mínimo denominador comum do fluxo de dados.
- Camadas suportam implementação e teste incrementais, permitindo a localização das mudanças.
- Também suportam arquiteturas dirigidas por responsabilidades, que dividem as tarefas em grupos de responsabilidades relacionadas.

Limitações:

- Acoplamento entre as camadas adjacentes é prejudicial à manutenção.
- Cada camada precisa gerenciar os dados de entrada e saída armazenando-os em um buffer interno e empacotando e desempacotando adequadamente
- Em geral é um padrão que oferece pouca eficiência na execução.
- Pode ser difícil estabelecer a granularidade das camadas (melhor ter 3 ou 7 camadas?).

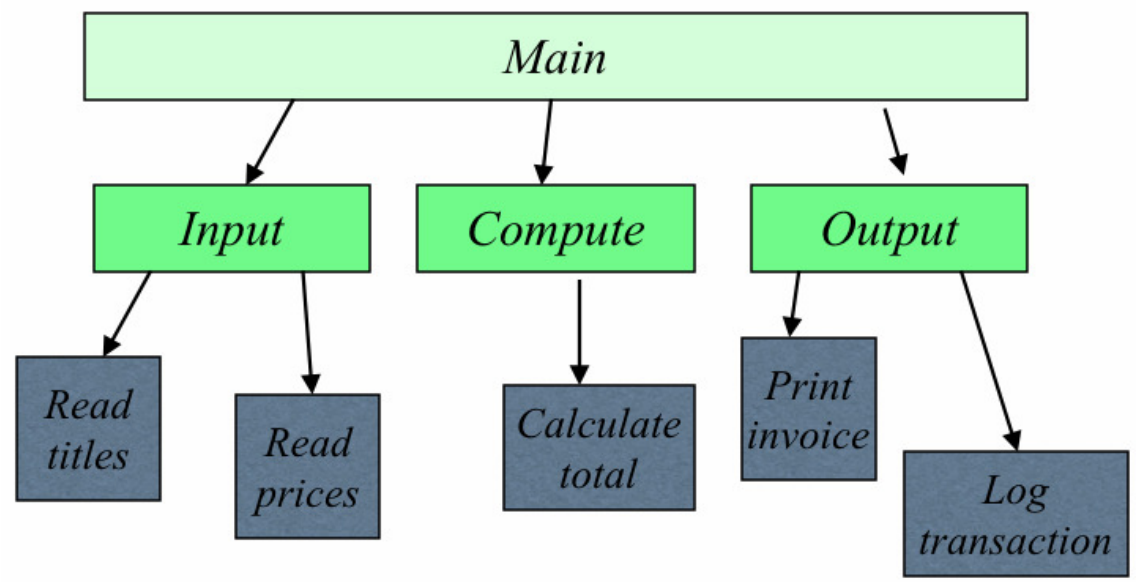
2- Sistemas de chamada-e-retorno

O fator predominante nesse padrão é a forma em que o controle é transferido entre os componentes. A transferência de controle pode ou não ser acompanhada de dados.

Variações:

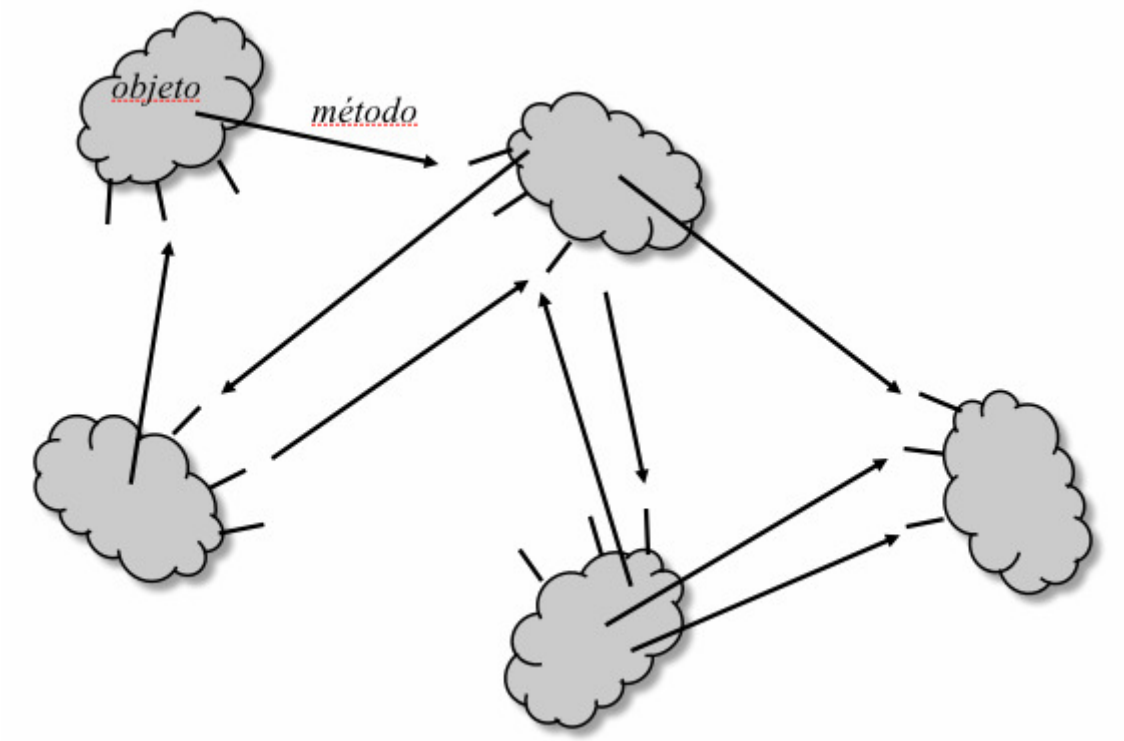
Programa principal e subrotinas

Consiste da decomposição hierárquica por funcionalidade (top-down). Tem suporte direto em linguagens de programação estruturada - a chamada de funções.



Orientação a objetos

- Decomposição por tipo de dados.
- Suporte direto em linguagens de programação
- Chamada de métodos
- Encapsulamento, herança, polimorfismo



3-Sistemas de redes

Os componentes executam em máquinas distintas de uma rede.

Os conectores são tecnologias e/ou protocolos de rede:

HTTP

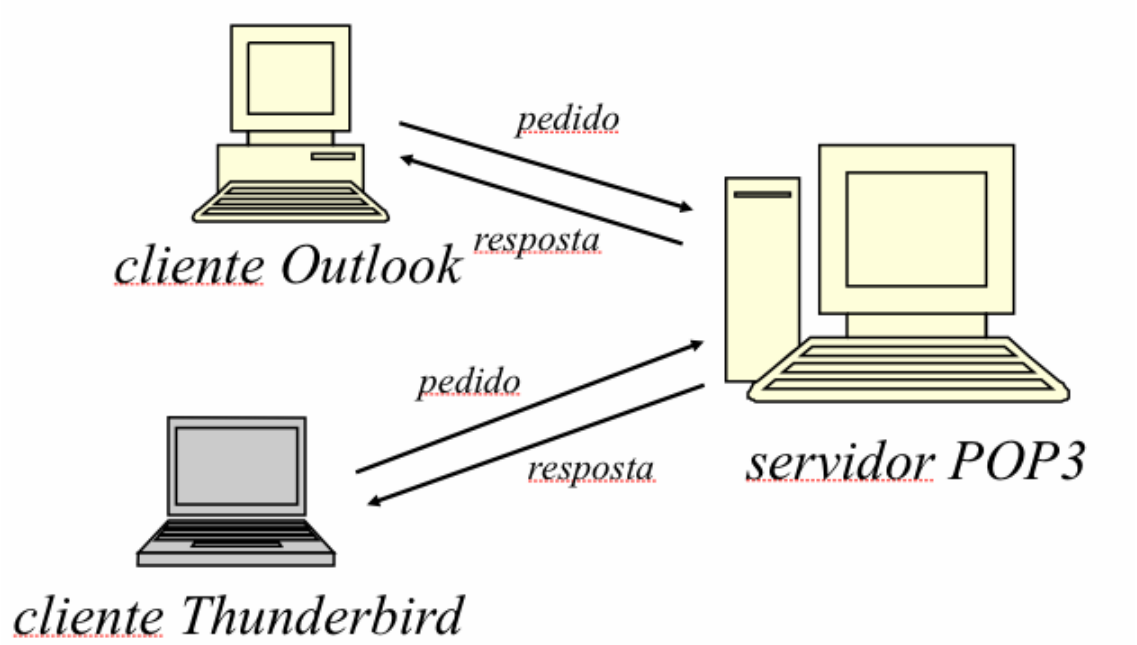
TCP/IP

Cliente-servidor

Em geral, clientes iniciam as computações e interagem com usuários.

Servidores, por sua vez, executam as computações e provêm recursos.

Exemplo: e-mail



Variações:

Cliente magro

Clientes "magros" são aqueles que praticamente só contem a interface com o usuário. Quase todo o processamento é feito no servidor.

Cliente grosso

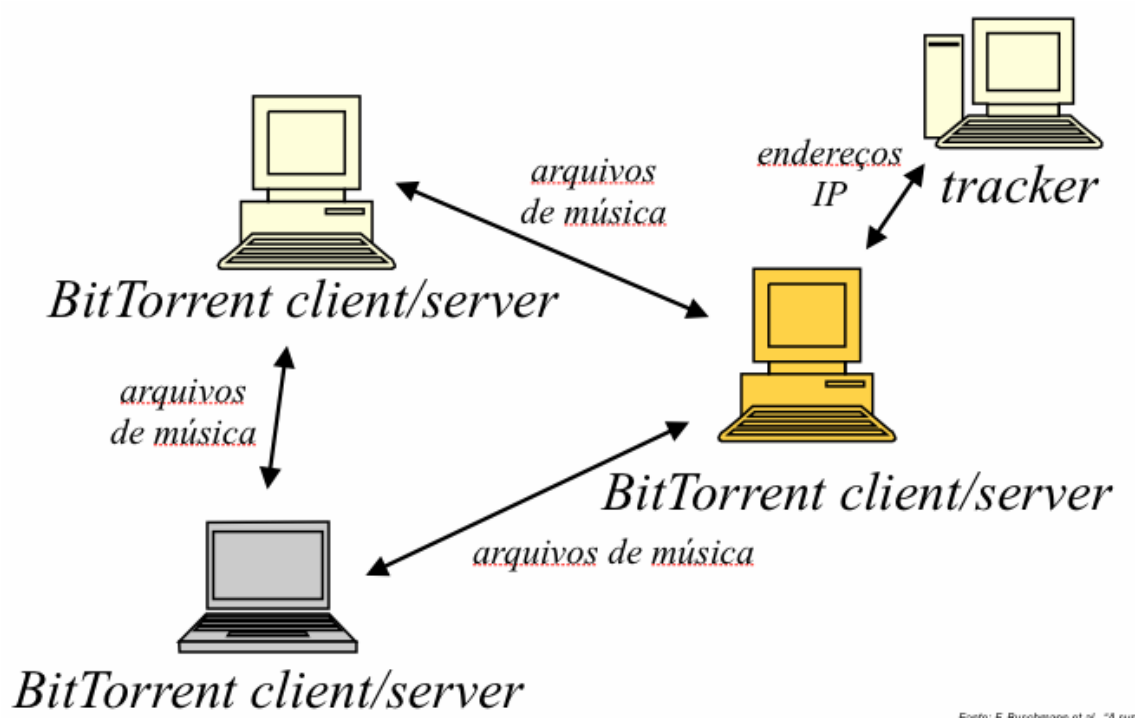
No cliente "grosso", o cliente faz o processamento e o servidor só centraliza recursos.

É uma abordagem vantajosa em determinadas situações:

- jogos eletrônicos se beneficiam da alta capacidade de processamento e de hardware gráfico específico para propiciar uma experiência multimídia mais interessante aos seus usuários
- aplicações na Web que buscam minimizar os contatos com o servidor web apenas aos momentos em que não há como produzir o mesmo efeito, obter alguma informação ou validá-la. Tecnologias relacionadas a este paradigma são AJAX, Adobe Air e Silverlight.
- A evolução do navegador como plataforma permitiu que os clientes ficassem grossos.

Peer-to-peer (P2P)

- Componentes executam em máquinas distintas
- Cada componente age como ambos cliente e servidor
- interage com o usuário
- efetua o processamento
- Em geral há um servidor onde os componentes se registram
- Exemplo: compartilhamento de música



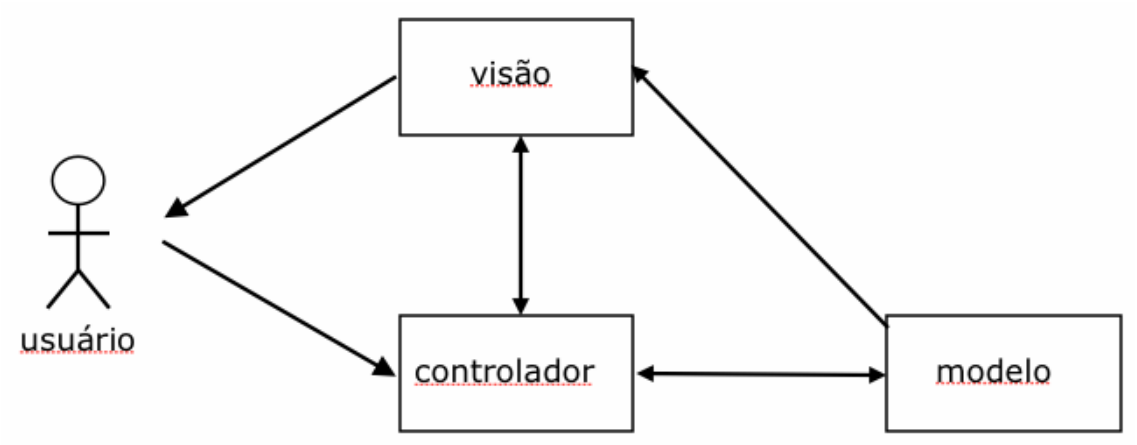
Características:

- Estende o estilo arquitetural cliente/servidor permitindo:
- múltiplas respostas à uma única requisição
- compartilhamento de dados entre os componentes (peers)
- descoberta de recursos
- resiliência à remoção de componentes (peers)

Model-view-controller (MVC)

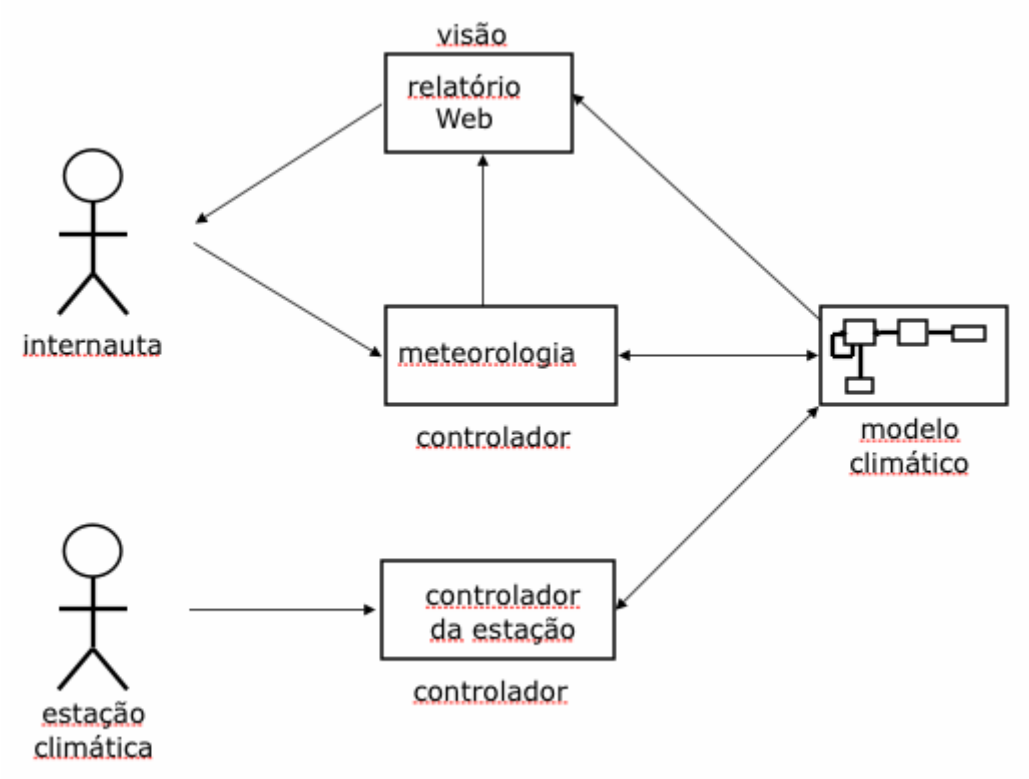
Divisão dos componentes em três tipos:

- o **modelo** faz o acesso aos dados,
- a **visão** mostra os resultados,
- o **controlador** recebe as entradas do usuário e toma as atitudes necessárias.



O MVC separa a interface com o usuário da apresentação e da lógica de negócios. Originado no ambiente Smalltalk, é um dos padrões arquiteturais mais usados. Há frameworks que implementam MVC, como o Struts , o Rails etc.

Exemplo: Previsão do tempo na Web



Modelo(Model)

- O modelo encapsula e manipula os dados do domínio que são de interesse. Ele não sabe como mostrar a informação que contem, não interage com o usuário e não recebe nenhuma entrada do usuário.
- O modelo encapsula a funcionalidade necessária para manipular, obter e entregar dados para as outras partes. É independente de interface com o usuário ou com outros dispositivos de entrada.
- Cada modelo tem uma lista dos seus dependentes, mas ele não sabe nada sobre estes dependentes e não se interessa por eles (precisa manter a lista apenas para notificar quando as mudanças acontecerem)

Visão(view)

- A **visão** é uma apresentação específica da informação contida no modelo.
- Pode ser gráfica ou baseada apenas em texto. Cada visão é dependente de um modelo; quando o modelo muda, todas as visões dependentes são atualizadas.

Controlador(Controller)

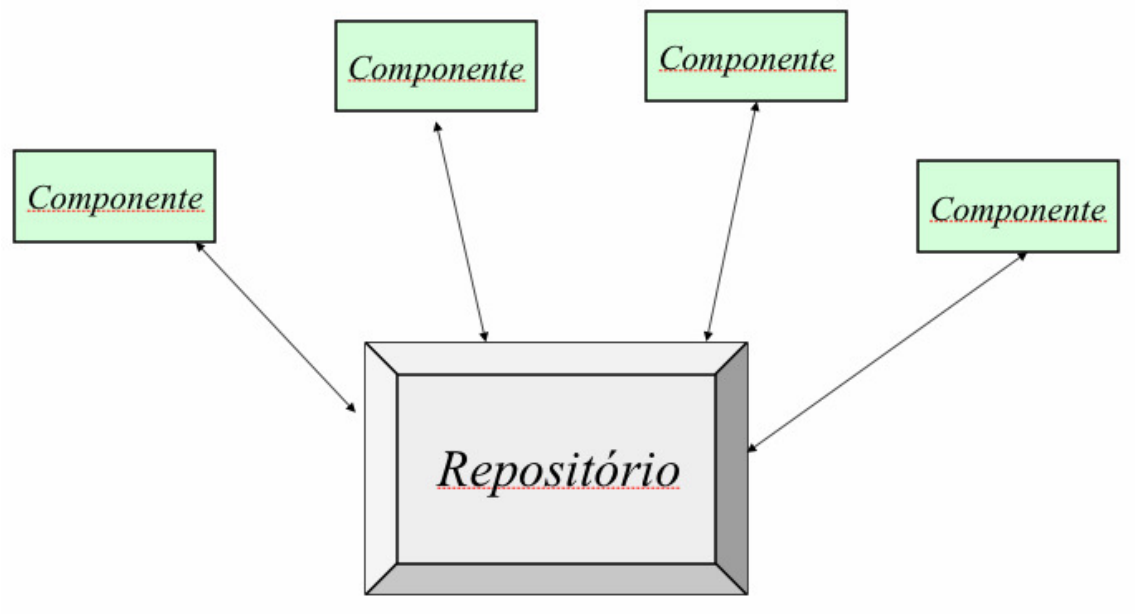
- **Controladores** lidam com a entrada do usuário (eventos) e tratam as requisições usando o modelo e as visões.
- O controlador possibilita o desacoplamento entre o modelo e as visões. Seu comportamento é dependente do estado do modelo.

Funcionamento do MVC

- O modelo tem uma lista das visões às quais ele dá suporte.
- Cada visão tem uma referencia para o seu modelo e para o seu controlador.
- Cada controlador tem uma referencia para a visão que ele controla e para o modelo no qual a visão se baseia. O modelo não sabe nada sobre o controlador.
- A cada evento gerado pelo usuário, o controlador notifica o modelo e este notifica suas visões sobre a mudança
- Alterar o controle de uma visão vai modificar a experiência de uso (feel).
- Alterar a visão de um controle vai modificar a aparência (look).
- Desta forma, o modelo está desacoplado da sua visão em termos de gerenciamento.

Repositórios

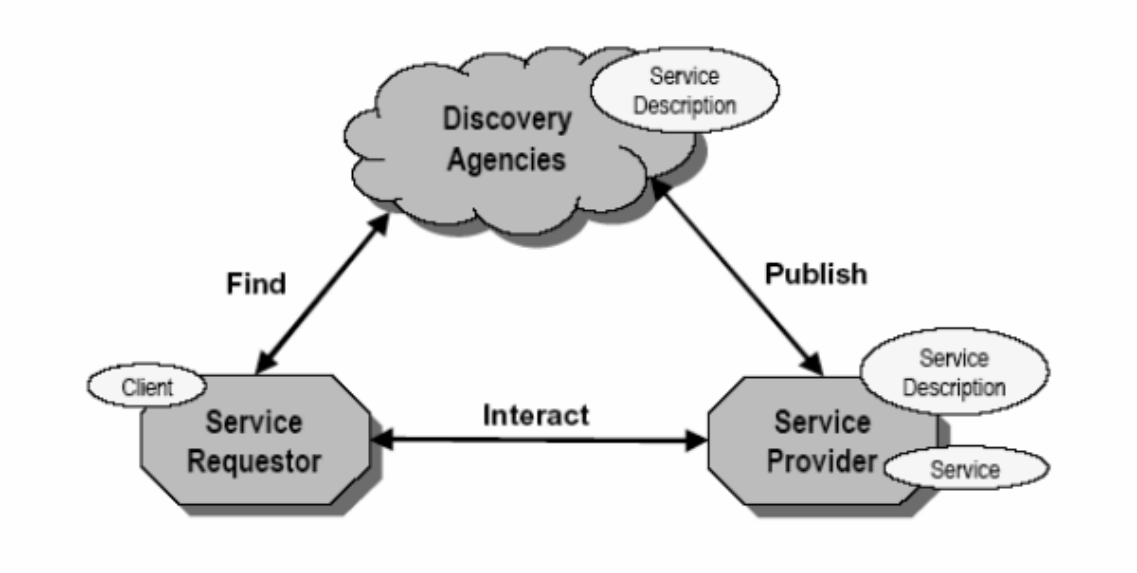
- Dois tipos de componentes:
- um repositório de dados centralizado
- uma coleção de componentes que operam sobre o repositório
- Componentes não se comunicam entre si.
- Dois tipos:
- Banco de dados centralizado: os componentes iniciam as ações (o acesso ao repositório)
- Quadro-negro: o repositório inicia as ações e notifica os componentes no caso de mudanças internas de seu interesse. Usado em sistemas de inteligência artificial



4-Sistemas orientados a serviços

- Componentes são serviços de organizações possivelmente diferentes (provedores).
- Conectores são tecnologias baseadas em trocas de dados via XML.
- As aplicações são requisitantes (consumidores) dos serviços.
- Existe um diretório de serviços onde eles podem ser descobertos.

Exemplo: serviço de conversão de moedas

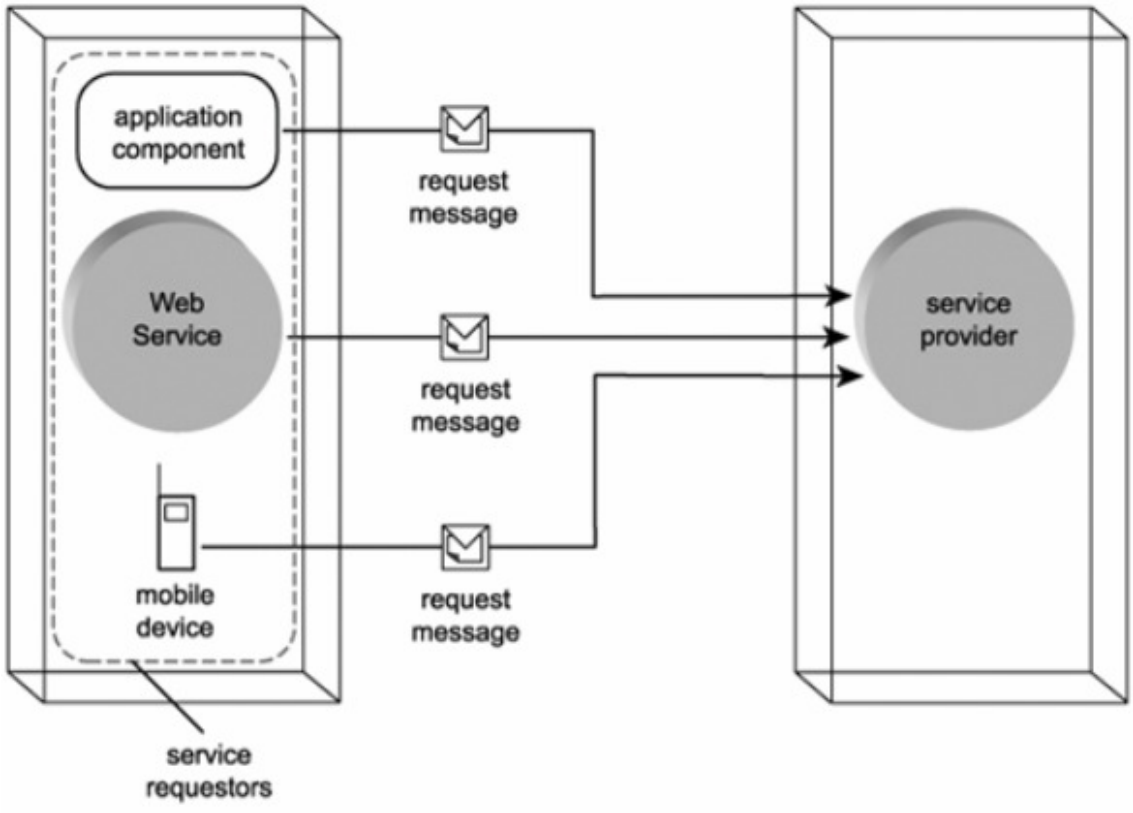


Framework de Web services para Arquiteturas orientadas a serviços

Composto de:

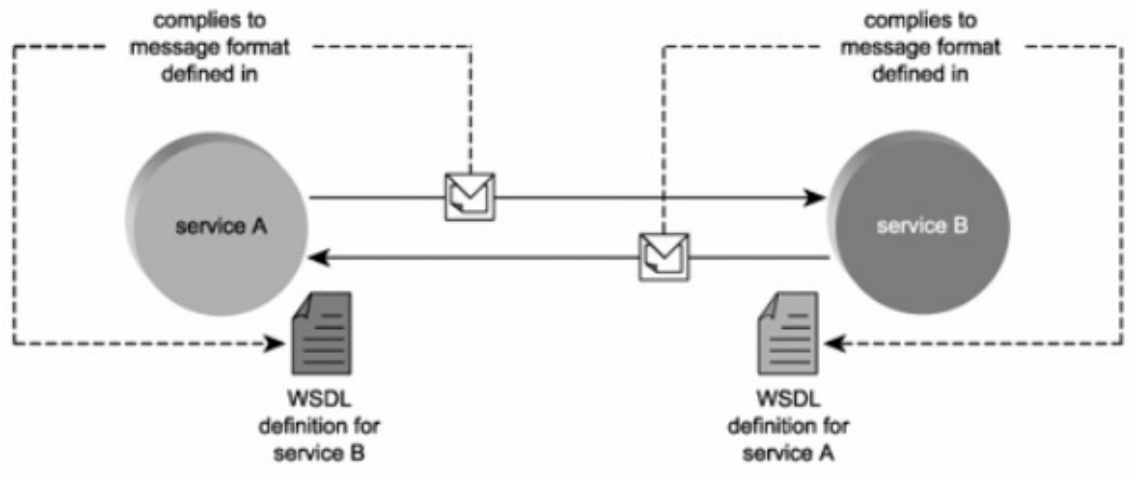
- Web services, descrições de serviços e mensagens
- um acordo de comunicação centralizado nas descrições de serviços baseadas em WSDL (Web service description language)
- um framework de mensagens baseado em tecnologia e conceitos SOAP (Simple Object Access Protocol)
- uma arquitetura para registro e descoberta de descrições de serviços em geral realizada através de UDDI (Universal Description, Discovery and Integration)
- arquitetura que dá suporte a padrões de mensagens e de composições

Provedor e requisitante:

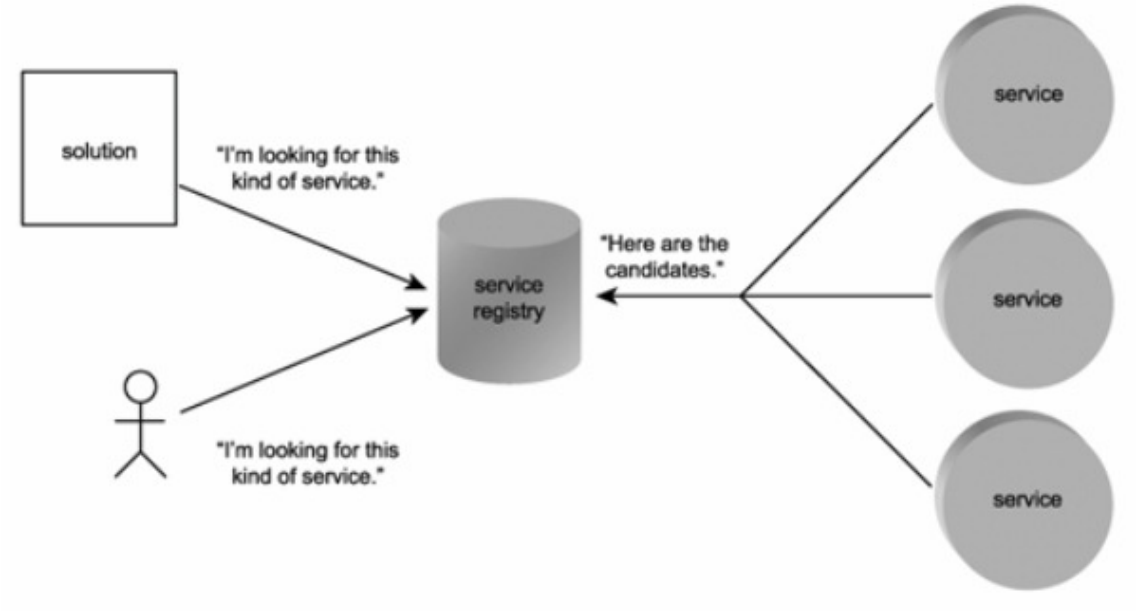


Descrições de serviços:

Necessários para estabelecer uma comunicação entre os serviços de forma consistente e com baixo acoplamento.



Registro e descoberta de serviços:



5-Computação na nuvem

- Um estilo de computação no qual os recursos escaláveis são providos através de serviços na internet.
- Refere-se à utilização da memória e das capacidades de armazenamento e cálculo de computadores e servidores compartilhados e interligados por meio da Internet, seguindo o princípio da computação em grade.

Características:

- *entrega de serviços*: a funcionalidade é provida através de serviços
- *fora do espaço físico da empresa*: os serviços cruzam os limites físicos e de segurança (firewall) da empresa
- *virtualização*: a infraestrutura é provida em diferentes níveis de abstração através de máquinas virtuais
- *elasticidade*: recursos são escaláveis aumentando ou diminuindo conforme a necessidade
- *custo flexível*: o custo é associado diretamente à medida de uso dos recursos
- *acesso universal*: democratização de acesso aos recursos

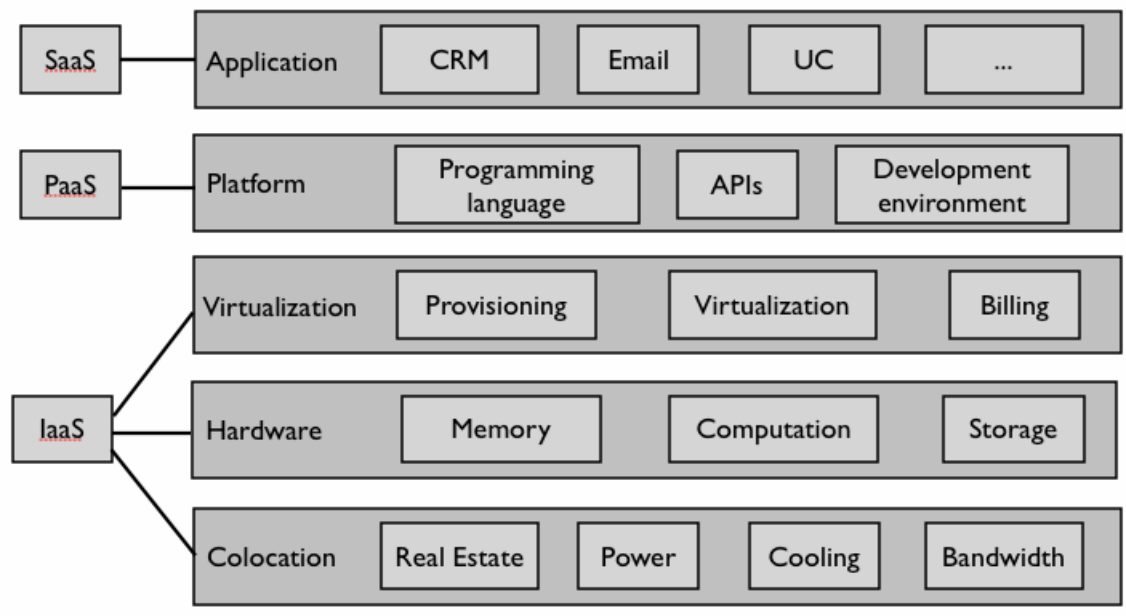
- *gerenciamento no nível de serviços*: nível de compromentimento entre o provedor do serviço e o cliente

Pilha de serviços:

Ao invés de oferecer soluções empacotadas que são instaladas monoliticamente nos desktops e servidores, a funcionalidade requerida pelos usuários é toda decomposta em primitivas que são combinadas quando necessário para criar soluções úteis.

É difícil agregrar toda a funcionalidade de maneira otimizada a não ser que você conheça todos os serviços disponíveis. Fica mais fácil se os serviços estiverem organizados através de um modelo que ilustra as interrelações entre os serviços.

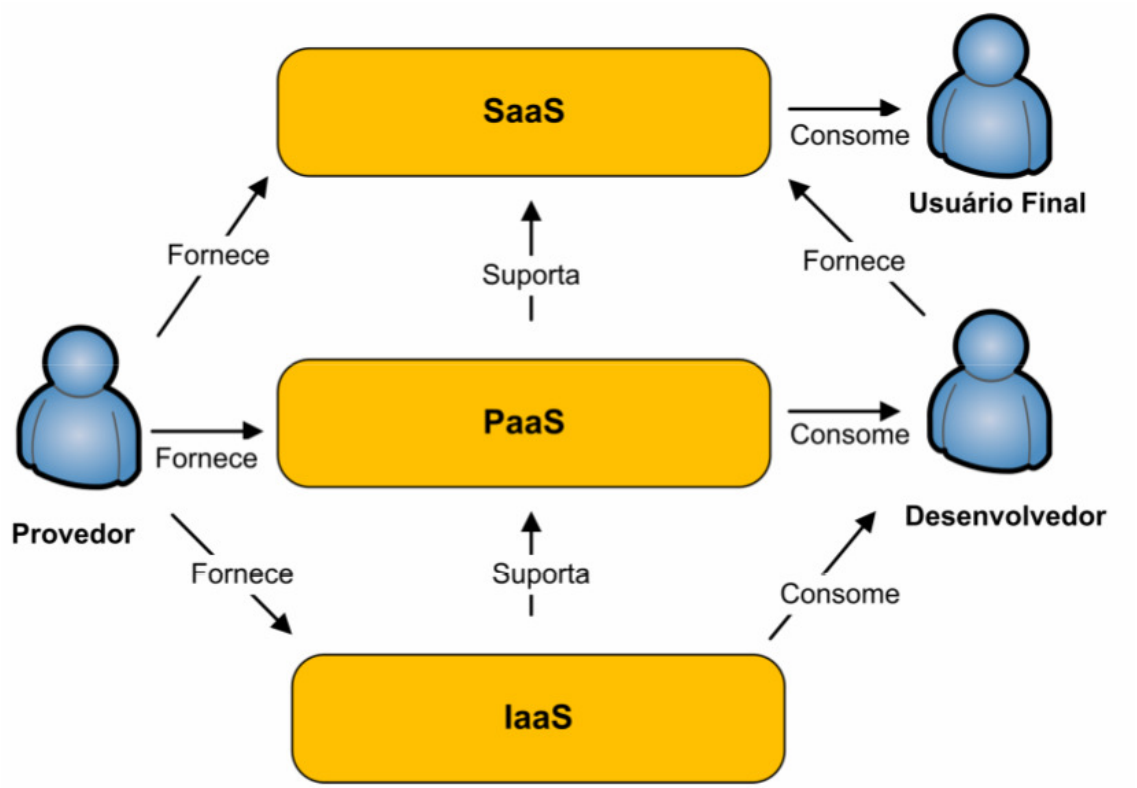
- *SaaS - Software as a Service*: nível de serviços mais abstratos com alvo nos usuários finais.
- *PaaS - Platafrom as a Service*: nível de serviços que provê plataformas de desenvolvimento com alvo nos desenvolvedores.
- *IaaS - Infrastructure as a Service*: nível de serviços mais baixo utilizados diretamente pelos desenvolvedores ou atra'ves das plataformas.



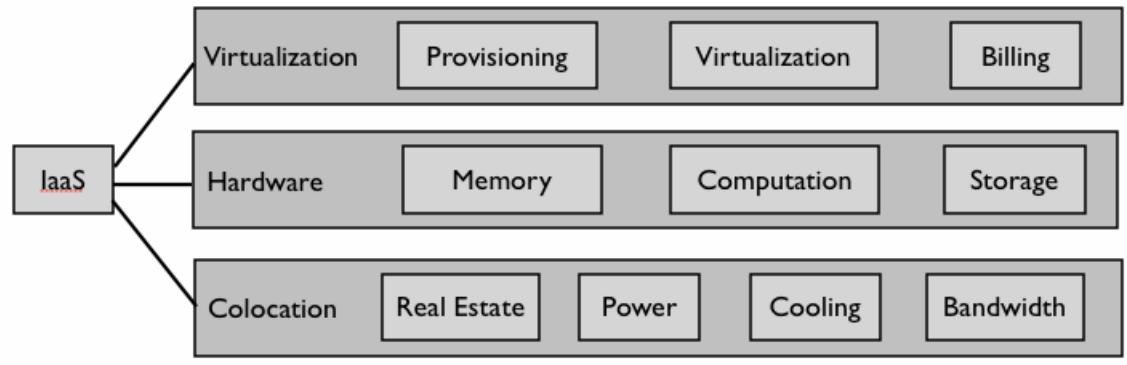
Exemplos:

- *IaaS*: Amazon Elastic Compute Cloud (EC2)
- *PaaS*: Google App Engine
- *SaaS*: Salesforce.com

Arquitetura na nuvem



Infraestrutura



6-REST- *Representational State Transfer*

É um estilo arquitetural que define um conjunto de restrições aplicáveis a web services de forma melhorar performance, escalabilidade e facilidade de modificação, de forma a permitir que os serviços funcionem melhor na web.

Princípios que tornam esse estilo simples, leve e rápido:

Identificação de Recursos

- Os recursos são os diversos componentes do estilo arquitetural e são identificados por URIs (*Uniform Resource Identificators*).
- Tudo deve ser representado como um recurso: produtos, categorias de produtos, clientes, carrinhos de compras, estado da aplicação, link para o próximo passo de um processo, uma página de resultados contendo diversos URI que são considerados os links de possível navegação da aplicação.

Interface Uniforme

O mesmo conjunto restrito de operações se aplica a todos os recursos. Existe um conjunto pequeno de verbos (operações) aplicado a um conjunto grande de nomes (recursos). Na Web isso significa as operações do HTTP:

- para criar um recurso, use POST.
- para listar um recurso, use GET.
- para alterar o estado de um recurso, use PUT.
- para remover um recurso, use DELETE.

Mensagens autodescritivas

Recursos são desacoplado de suas representações e são acessados através de suas representações:

- XML, HTML, JSON, etc são usados nas representações de recursos
- Representações de recursos possuem links para recursos identificados (através de URIs na Web).

- Os recursos podem ser usados através dos links de navegação.
- Aplicações que são RESTful navegam entre recursos ao invés de chamar recursos.

Interações sem estado

- Isso não significa que as aplicações não tem estado.
- Significa que o estado é guardado no cliente ou no recurso, mas não deve ser guardado na aplicação no servidor.
- O estado do recurso é gerenciado no servidor.
- Para troca de estado, podem ser usados cookies, campos escondidos, etc.

Referências

Oracle- JEE-Architecture
Material produzido pela Prof. Maria Augusta Vieira Nelson