

Arquitetura Orientada a Serviços

Fernando Trinta & Windson Viana

Cenário Atual para Aplicações Distribuídas

- Mercado Competitivo e Globalizado
 - Time-to-Market reduzido
- Mudança constante de tecnologia
- Heterogeneidade
 - Plataformas
 - Linguagens
- Fusão de Empresas
 - Parcerias
- Alinhamento Ti e estratégias de negócio das empresas

Cenário Atual de Aplicações Distribuída

- Processos de Negócio e Sistemas cada vez mais complexos
 - Necessidade de automação e integração
- E ainda...
 - Escalabilidade
 - Redução de Custos

Complexidade de Sistemas Atuais

Sistemas de Informação Passados

Homogêneos

Estáticos

Monolíticos

Sistemas de Informação Atuais

Heterogêneos

Dinâmicos

Elementos
Independentes

Questões Chave

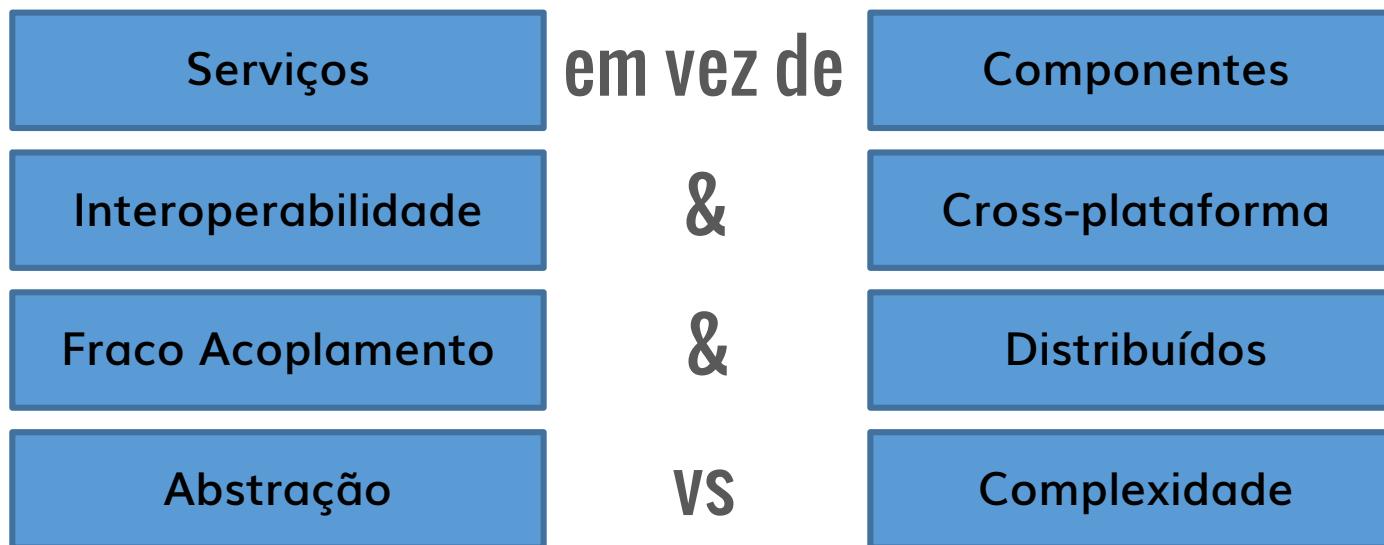
Maior Abstração

Reuso

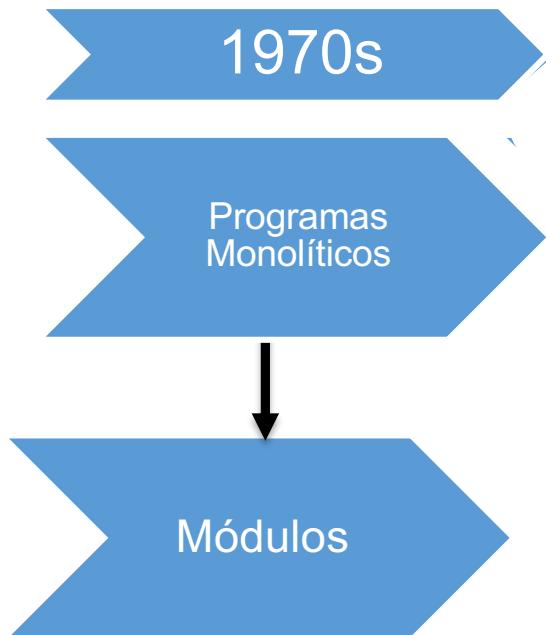
Fraco Acoplamento

- Elevar o nível de abstração na modelagem e projeto de sistemas
 - Objetos, componentes, frameworks, middleware
- Reuso
 - “Não Reiventar a roda”
- Fraco Acoplamento

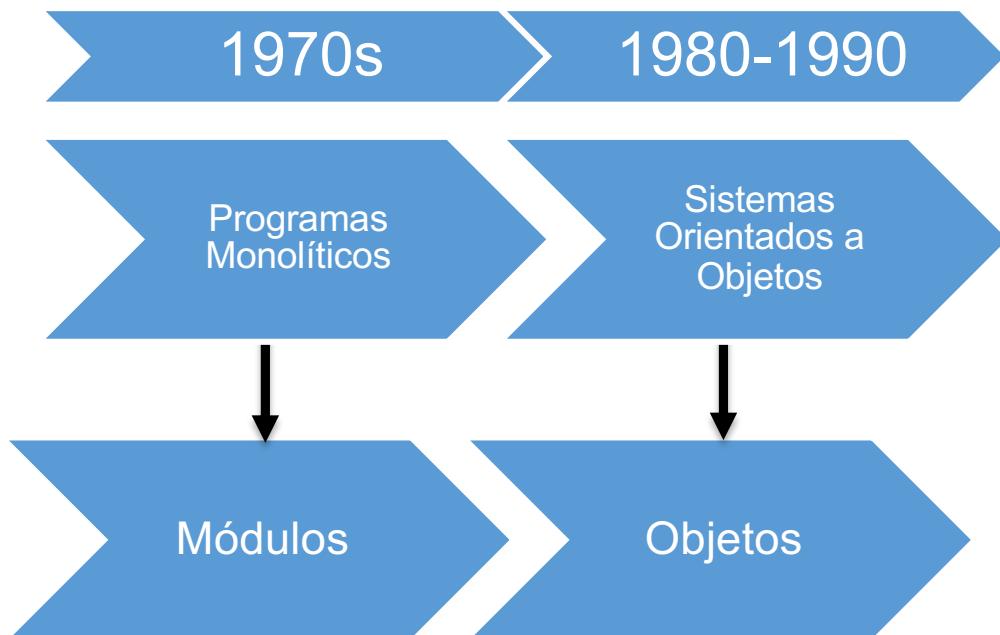
Para onde se está indo...



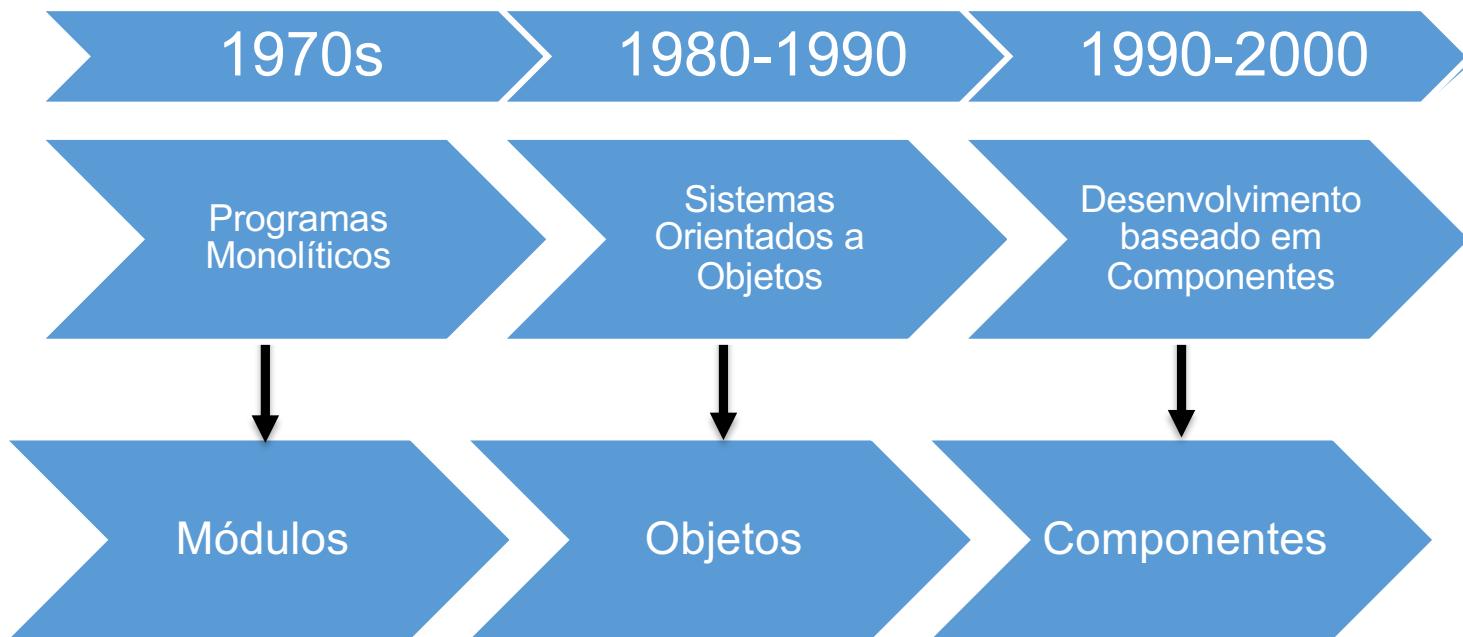
Para onde se está indo



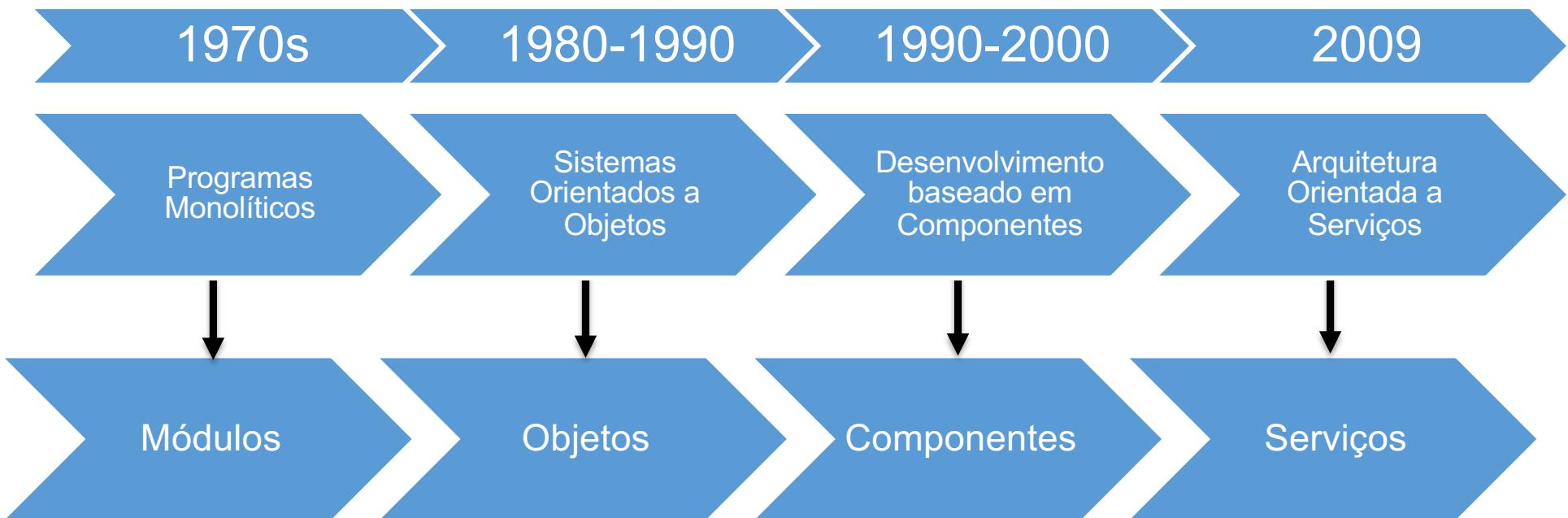
Para onde se está indo



Para onde se está indo



Para onde se está indo



Exemplo1: Zona Azul

- Especificação de um sistema de pagamento de estacionamento na cidade, de acordo com a rua/bairro, a partir do smartphone do motorista

Exemplo 2: Shazam



<https://www.shazam.com/>

Arquitetura Orientada a Serviços

Uma abordagem para criar sistemas computacionais distribuídos baseada no encapsulamento de funções de negócio em serviços que podem ser acessados de forma fracamente acoplada.

Arquitetura Orientada a Serviços

"(...)um paradigma de design pretendido para a criação de unidades de lógica de solução que são individualmente moldadas para que possam ser coletivamente e repetidamente utilizadas para realizar um conjunto específico de objetivos (...)"

Thomas Erl. SOA Principles of Service Design. Prentice Hall, 2016.

O que SOA não é...

- Um produto
- Uma tecnologia específica
- Uma aplicação
- Um padrão específico
- Um conjunto específico de regras
- Web Services
- Bala de prata (*Silver bullet*)

O que SOA é...

- Estilo Arquitetural
- Abordagem para sistemas distribuídos complexos (grandes)
 - heterogêneos
- Abordagem para projeto de aplicações distribuídas com base em:
 - Serviços Fracamente Acoplados
 - Interfaces e protocolos padronizados
 - Integração e Interoperabilidade

Benefícios

- Facilita o gerenciamento do crescimento e evolução de sistemas corporativos
- Facilita a oferta de escalabilidade
- Reduz custo através da cooperação das organizações
- *Como paradigma*
 - Organiza grandes sistemas em rede que precisam de interoperabilidade
 - Acrescenta valor aos componentes individuais

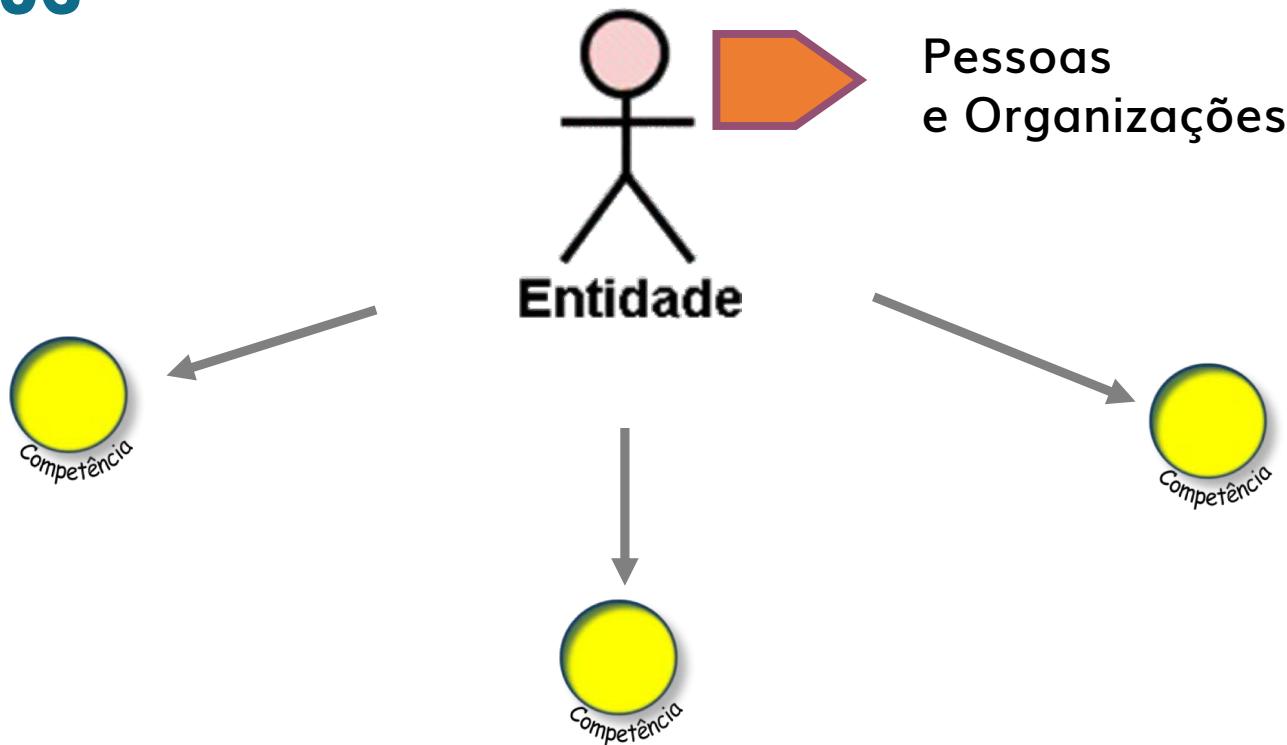


Modelo de Referência para SOA

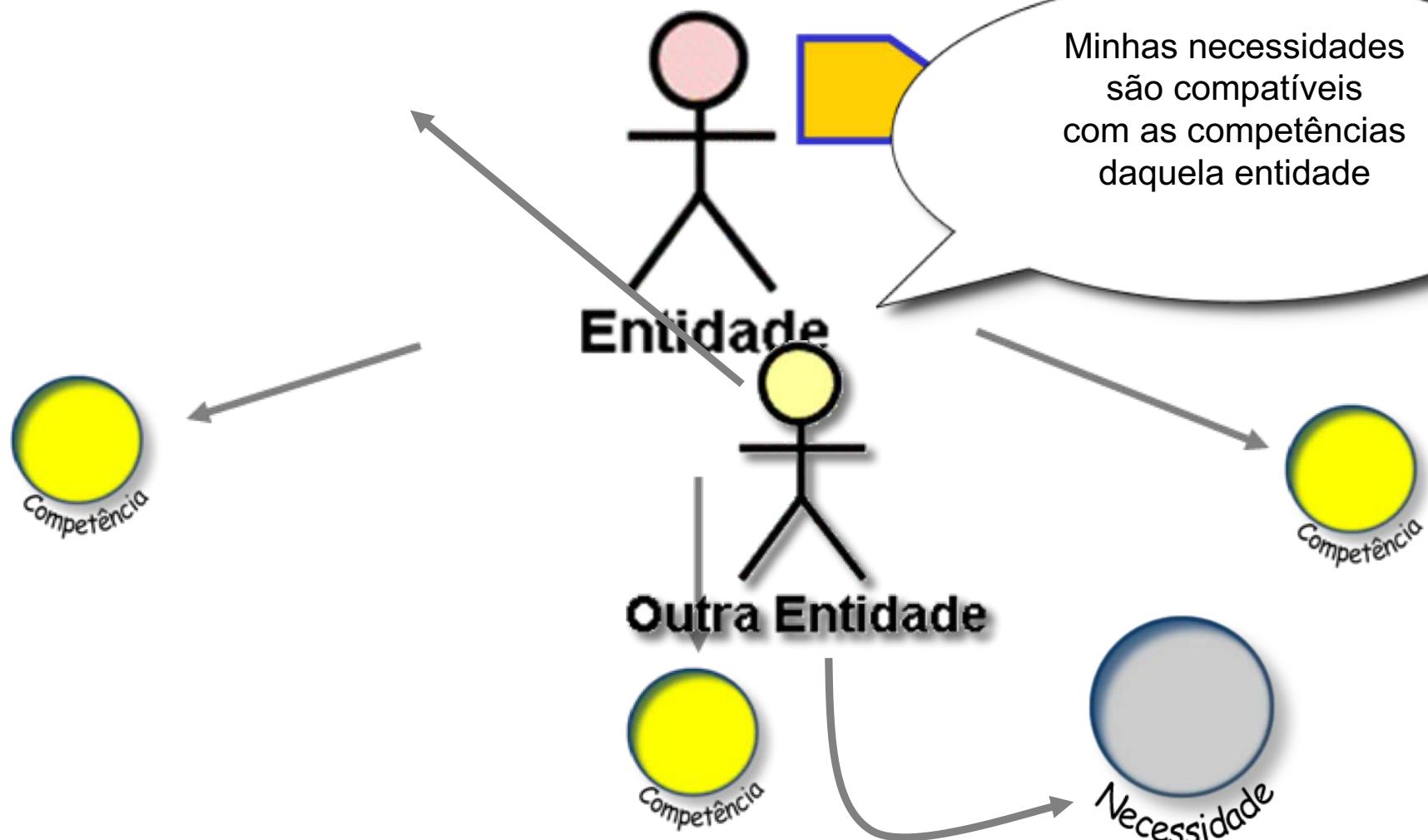
O que é um modelo de referência

- Um modelo de referência é um framework abstrato para entendimento dos relacionamentos significantes entre as entidades de algum ambiente.
- Um modelo de referência consiste de um conjunto mínimo de conceitos unificados, axiomas e relacionamentos com um domínio de um problema particular.
- Um modelo de referência é independente de padrões específicos, tecnologias, implementações ou outro detalhe concreto.

Entidades



Suportam soluções para problemas encontrados no decorrer de seus negócios





Pode requerer a combinação
de várias competências
Existência independente da
competência



Pode tratar mais de uma necessidade
Existente independente de necessidade

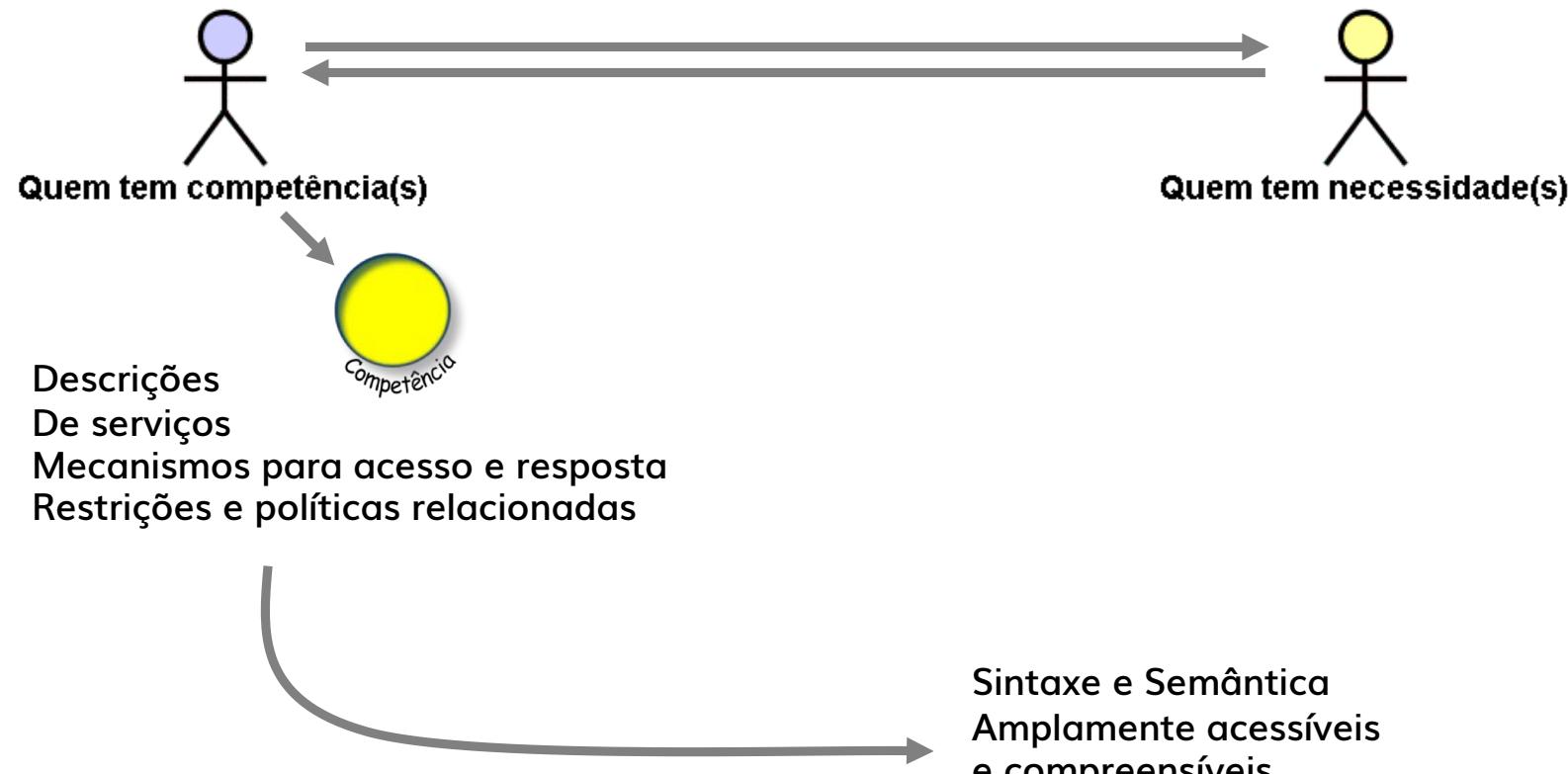


Compatibilização
Combinação de competências para tratar
necessidades

Conceitos Chave

- Visibilidade
- Interação
- Efeitos

Visibilidade

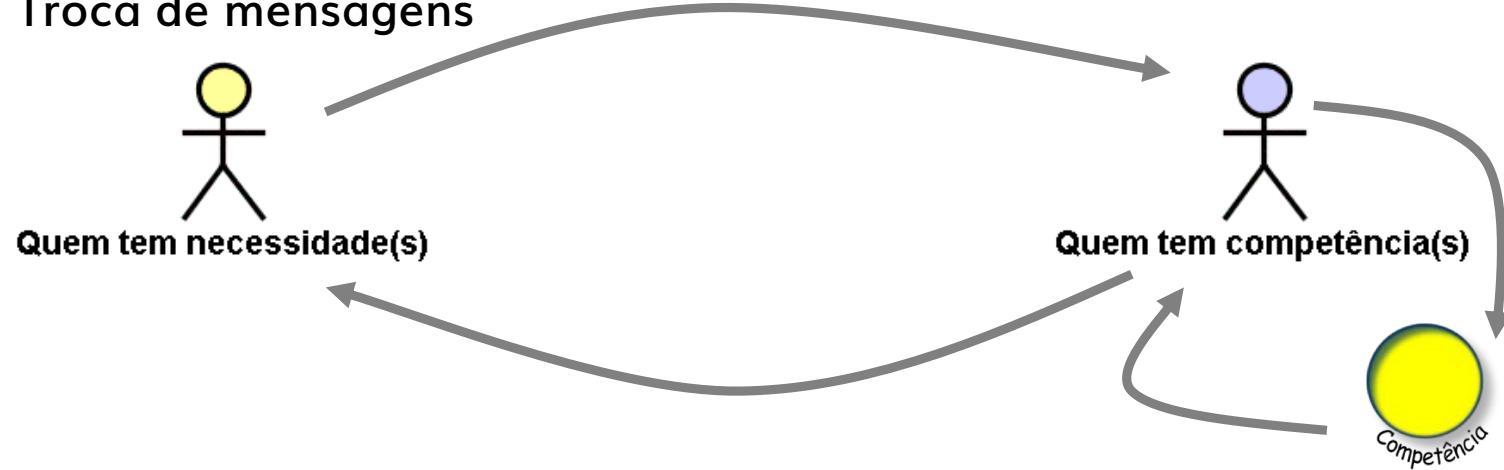


Visibilidade

- Visibilidade
 - Capacidade para aqueles com necessidades e aqueles com competências estarem aptos a se enxergarem mutuamente.
 - A visibilidade é promovida através da descrição do serviço, que contém as informações necessárias para interagir com o serviço.
 - Descrita em termos de entradas, saídas e semânticas associadas ao serviço.

Interação

- Uso da competência
 - Troca de mensagens



Interação

- Interação
 - Atividade que usa a competência.
 - Tipicamente mediada por troca de mensagens, uma interação prossegue através de uma série de ações de troca de informações e invocações.

Efeitos

- Efeitos
 - O propósito de usar as competências é provocar um ou mais efeitos no mundo real.
 - O resultado de uma interação é um efeito (ou um conjunto de efeitos).
 - Este efeito pode ser:
 - O retorno de uma informação, ou
 - A mudança no estado de entidades (conhecidas ou desconhecidas) que estão envolvidas na interação.

Serviço

- O nome “serviço” é definido no dicionário como “o desempenho de trabalho (uma função) por alguém para outro”.
- Contudo, serviço, como o termo é geralmente entendido, também combina as ideias relacionadas com:
 - A competência de executar o trabalho para outro
 - A especificação do trabalho oferecido para outro
 - A oferta para executar trabalho para outro

“Mecanismo pelo qual as necessidades e as competências são colocadas juntas”

Serviço

- O serviço é o bloco de construção principal de um software orientado a serviços
- Os serviços são módulos independentes e autocontidos que oferecem funcionalidades de negócio específicas
- Os serviços são descritos de forma padronizada, possuem interfaces bem definidas publicadas e se comunicam com outros serviços por meio de invocações remotas.

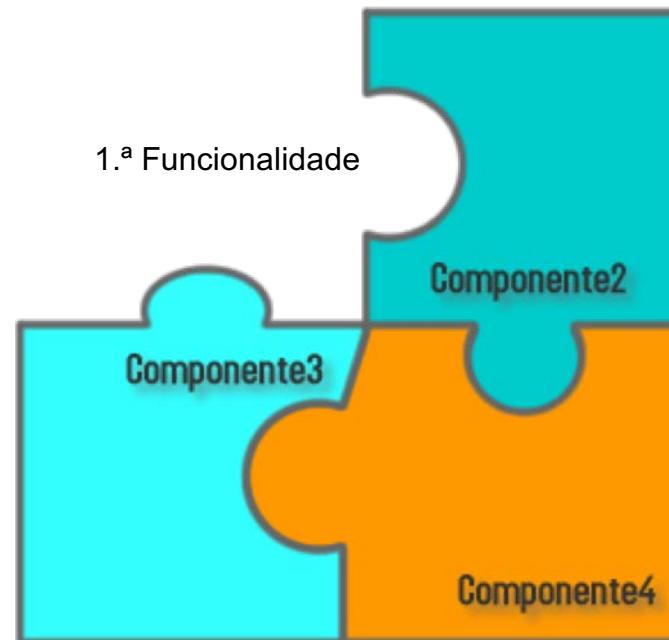
Serviço & Componentes

- “Um componente é um pacote coerente de implementação que
 - pode ser desenvolvido e distribuído independentemente,
 - provê interfaces explícitas e bem especificadas,
 - define interfaces que ele precisa de outros componentes, e
 - pode ser combinado com outros componentes pela configuração de suas propriedades, sem a necessidade de modificação.”

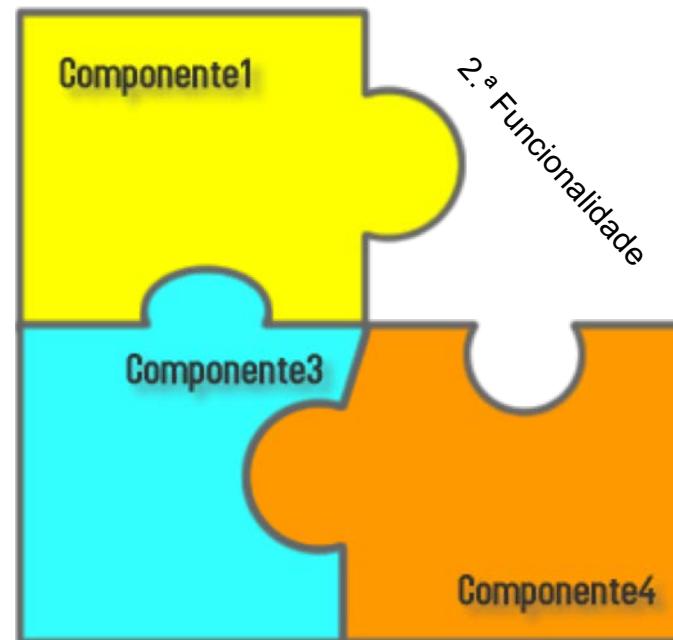
Serviço & Componentes



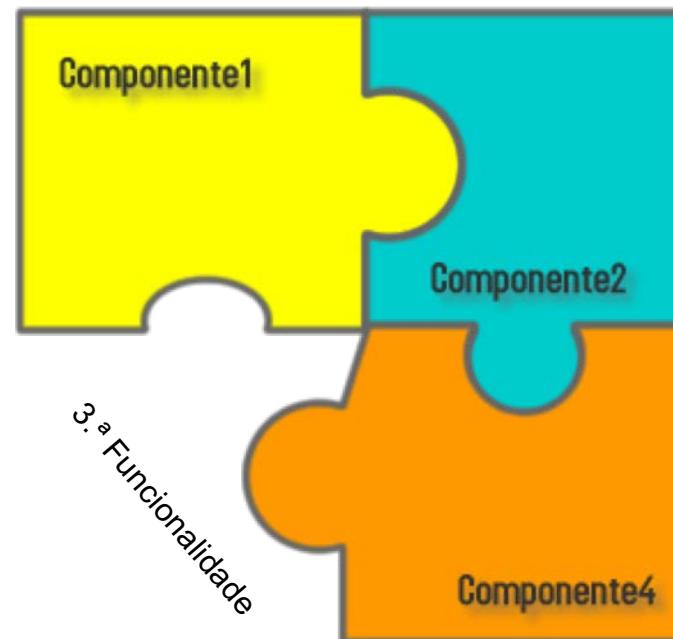
Serviço & Componentes



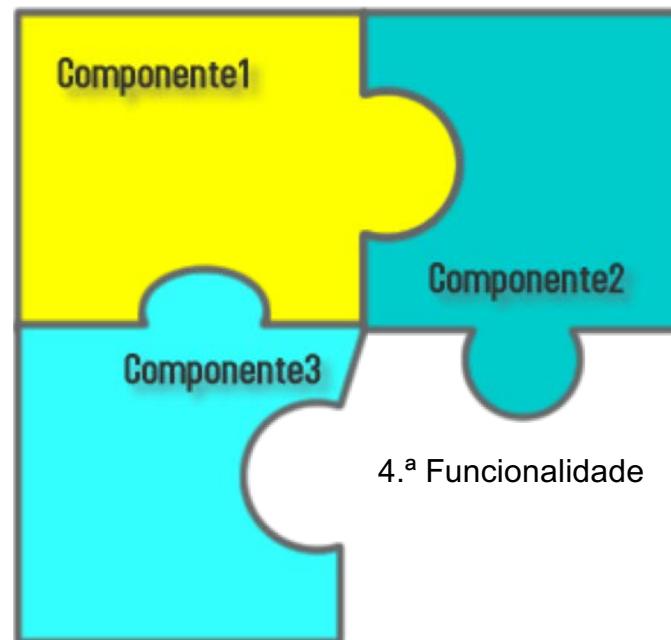
Serviço & Componentes



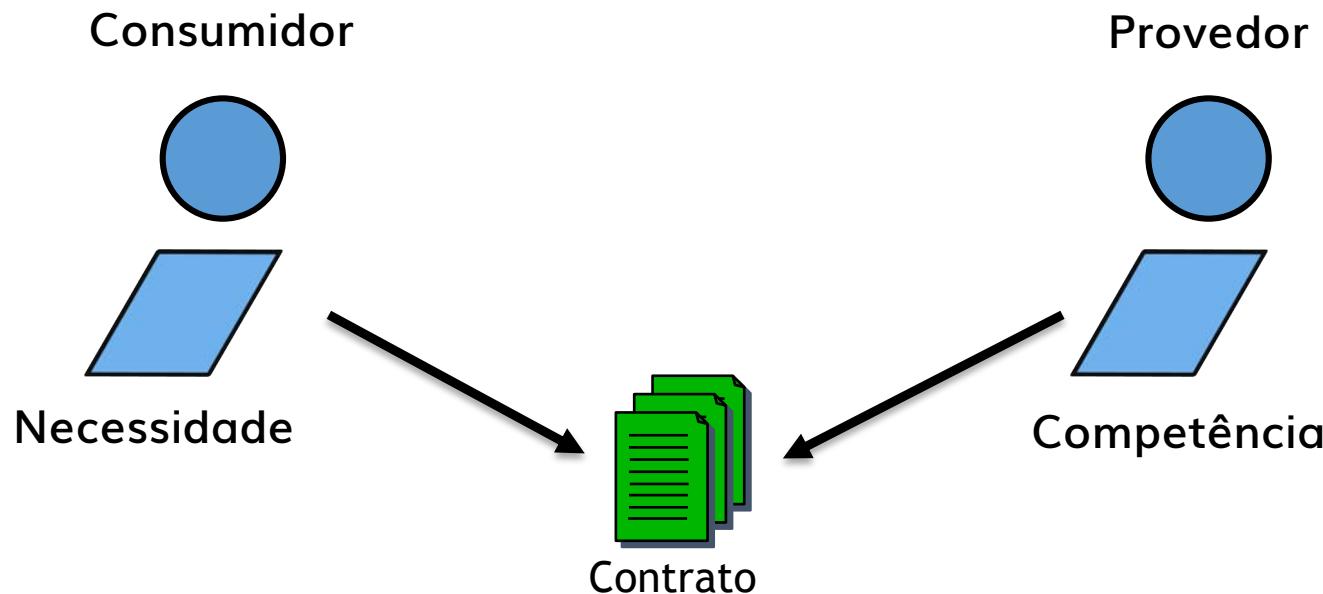
Serviço & Componentes



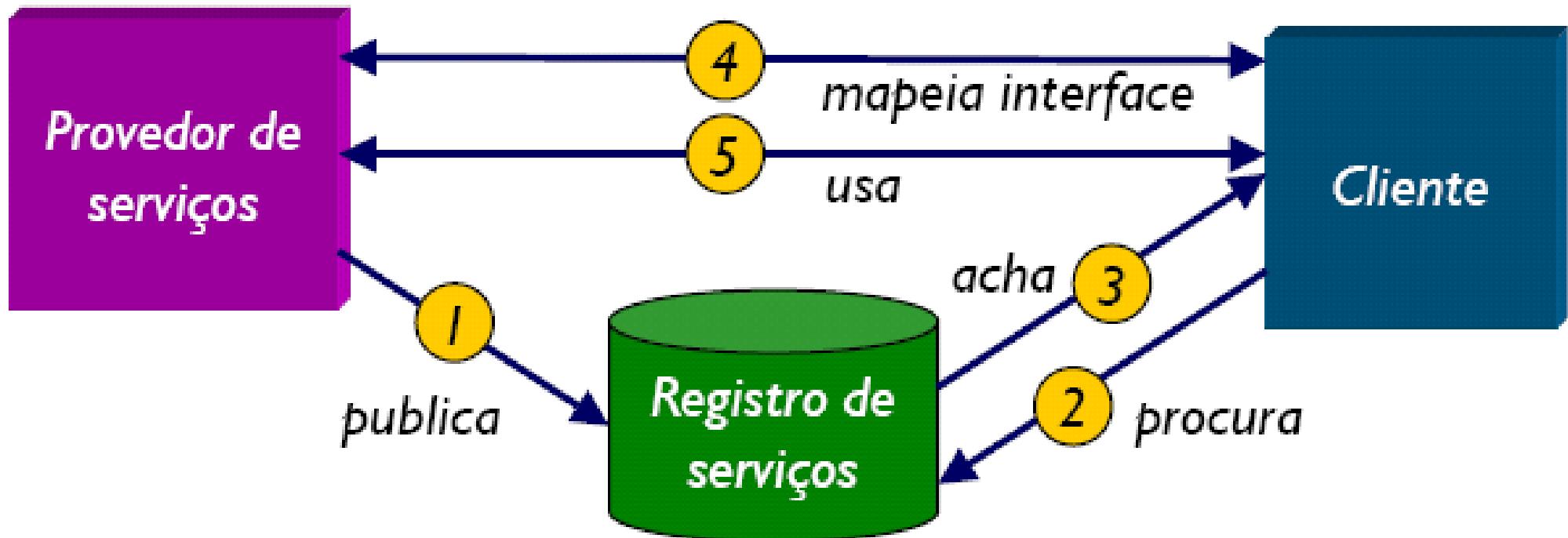
Serviço & Componentes



Serviços e Entidades



Outra visão



Serviços são...

1. Reutilizáveis
2. Compartilham um contrato formal
3. Possuem baixo acoplamento
4. Abstraem a lógica
5. São capazes de se compor
6. São autônomos
7. Evitam alocação de recursos por longos períodos
8. Devem possuir a capacidade de serem descobertos

Ser Reutilizável

- Um serviço reutilizável é aquele que não carrega *particularidades técnicas* de uma implementação ou *regra de negócio* específica e é genérico o suficiente para atender outros projetos.
- Pontos Positivos:
 - Um serviço reutilizável abrange diversos cenários de uso por consistir em uma *lógica mais genérica*. Isso é mais simples quando a construção dos serviços é feita de forma corporativa, como nas fábricas de software.
- Pontos Negativos:
 - *maior esforço* para projeto
 - Efeitos colaterais em casa de alterações

Contrato Formal

- Contratos são documentos textuais que descrevem o que o serviço faz.
 - WSDL
 - SOAP
 - UDDI
- Aspectos Positivos
 - Traduzem funcionalidade dos serviços especificados para que os clientes possam buscá-los e utilizá-los conforme a sua necessidade
- Aspectos Negativos
 - Necessidade de diversos padrões de contratos
 - Tornar-se uma tarefa complexa

Baixo Acoplamento

- Capacidade de ser independente de outros serviços para realizar a sua tarefa
 - Alta coesão (atividade definida e coerente)
- Tipos de Acoplamento:
 - Implementação, Contrato, Service Policies, Processos, Estrutura de Dados, Infraestrutura, Semântico
- Aspectos Positivos:
 - Reuso e flexibilidade
- Aspectos Negativos:
 - Dificuldade em ter algo “totalmente” desacoplado

Abstrair a lógica

- Serviços devem ser tratados como caixa-preta
- Ignorar diferenças entre tecnologias
- Aspectos Positivos
 - Facilidade de implementação
- Aspectos Negativos
 - Em sistemas legados, pode ser difícil conseguir transparência para consumidores

Composição de Serviços

- Criar serviços que sejam capazes de se juntar e serem acessados de forma a englobar e atender um problema maior
- Aspectos Positivos
 - Dividir para conquistar
- Aspectos Negativos
 - Evitar a interdependência entre serviços

Autonomia

- Capacidade de se autogovernar
- Sem dependência de elementos externos para sua lógica

Como implementar SOA?

- Descrição de um serviço de forma independente de uma plataforma
- Integração firewall-friendly entre Serviços hospedados em domínios distintos
- Programação Multiplataforma

Web Services

Web Services

- Web Services foi a tecnologia mais popular para implementação de serviços em SOA inicialmente
- Baseados em um conjunto de padrões como WSDL, SOAP, UDDI (SOAP-based Web Services)
 - Mais recentemente.. RESTfull Web Services
- Diversas linguagens de programação e frameworks dão suporte a construção de Web Services
 - Java, PhP, Ruby, Python, .NET, etc..

Web Services

- Um Serviço Web é a lógica de uma aplicação, disponível programaticamente e acessível sobre a Internet/Web, através da padronização de XML e construída sobre protocolos padronizados XML
- W3C define Web Service como um software projetado para garantir interoperabilidade entre interações máquina-máquina em uma rede de computadores
- Um Web Service que pode ser invocado ou composto através de tecnologias standards da WEB (HTTP, XML, ...) por um cliente que não necessariamente é um Browser

Web Services

- Web services se referem a um conjunto de padrões cujo foco é prover interoperabilidade de sistemas distribuídos
 - Define protocolos de comunicação
 - Define formato das interfaces para especificar os serviços
 - Define formato para as mensagens

Web Services

- Conceito surgiu no início do ano 2000
 - Se referia a um conjunto de padrões utilizados para a comunicação de sistemas através da rede (comumente na internet)
 - Padrões: XML, HTTP
- A Microsoft e algumas empresas começaram a trabalhar em um protocolo chamado SOAP
 - Objetivo era trocar dados representados em XML sobre uma conexão baseada em HTTP
 - IBM deu suporte a criação de outros padrões no ano 2000: Universal Description Discovery and Inovation (UDDI) e Web Services Description Language (WSDL)
 - Padrão ganhou popularidade e várias empresas deram suporte

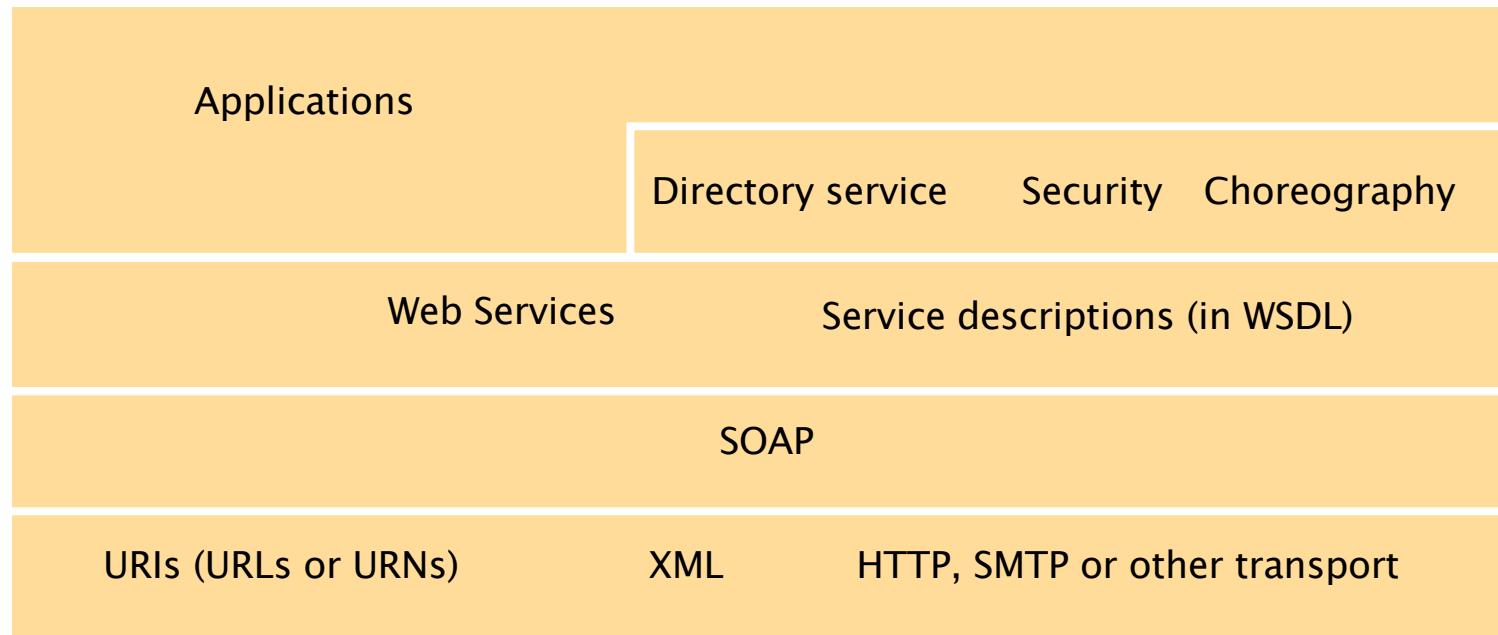
Web Services

- Hoje em dia existem diversas padronizações relacionadas a web services
 - Segurança (WS-Security)
 - Process Modeling (BPEL)
 - Service Management
 - E várias outras assuntos...
- Diversas organizações de padronização
 - OASIS, W3C, WS-I, etc...

Abordagens Principais

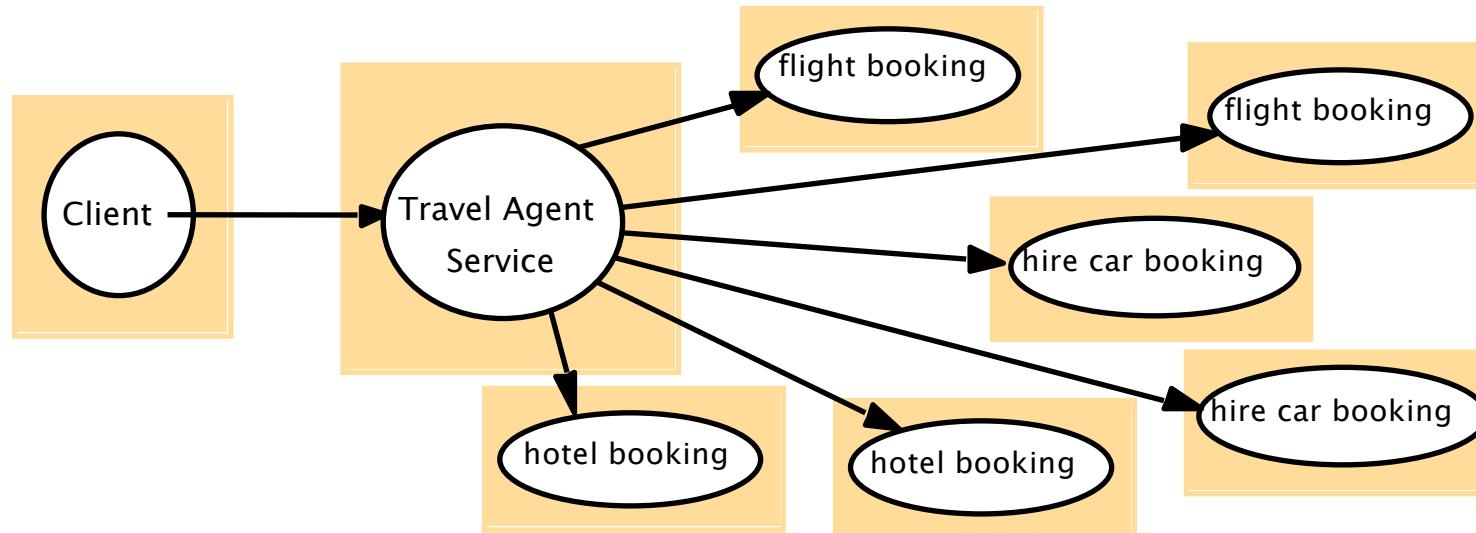
- SOAP – Envio e recebimento de mensagens XML para invocação de serviços remotos
- REST – Serviço é tratado como recurso por meio de operações básicas (GET, POST, PUT, UPDATE)

Infraestrutura



Composição

- Orquestração e Coreografia
 - Ferramentas como BPEL



Web Services vs RMI

- Vantagens:
 - Suporte a operações assíncronas
 - Orientado a documento
 - XML
 - Protocolo de comunicação independente de linguagem
 - SOAP
 - Descrição de serviço independente de linguagem
 - WSDL
 - Interoperabilidade sobre plataformas
 - RMI consegue parcialmente através de IIOP

Web Services vs CORBA

- Vantagens:
 - Fraco acoplamento cliente-servidor
 - Independente de linguagem
 - Localização por URL
 - Aplicabilidade a sistemas de arquitetura de segurança com Firewall
 - HTTP porta 80
 - Mais aplicável a sistemas com interface Web
 - Conversão: SOAP message => HTML
 - Melhor suporte à mobilidade e à distribuição
 - Troca de mensagens = facilidade para troca de endereços
 - Proxies realizam trabalho transparente para os envolvidos
 - Simples reenvio da mensagem em caso de erro
- Clientes limitados (ex: celulares)
 - Necessidade: enviar / receber SOAP messages
 - Parser ajustado apenas pela funcionalidade requerida pela aplicação do cliente

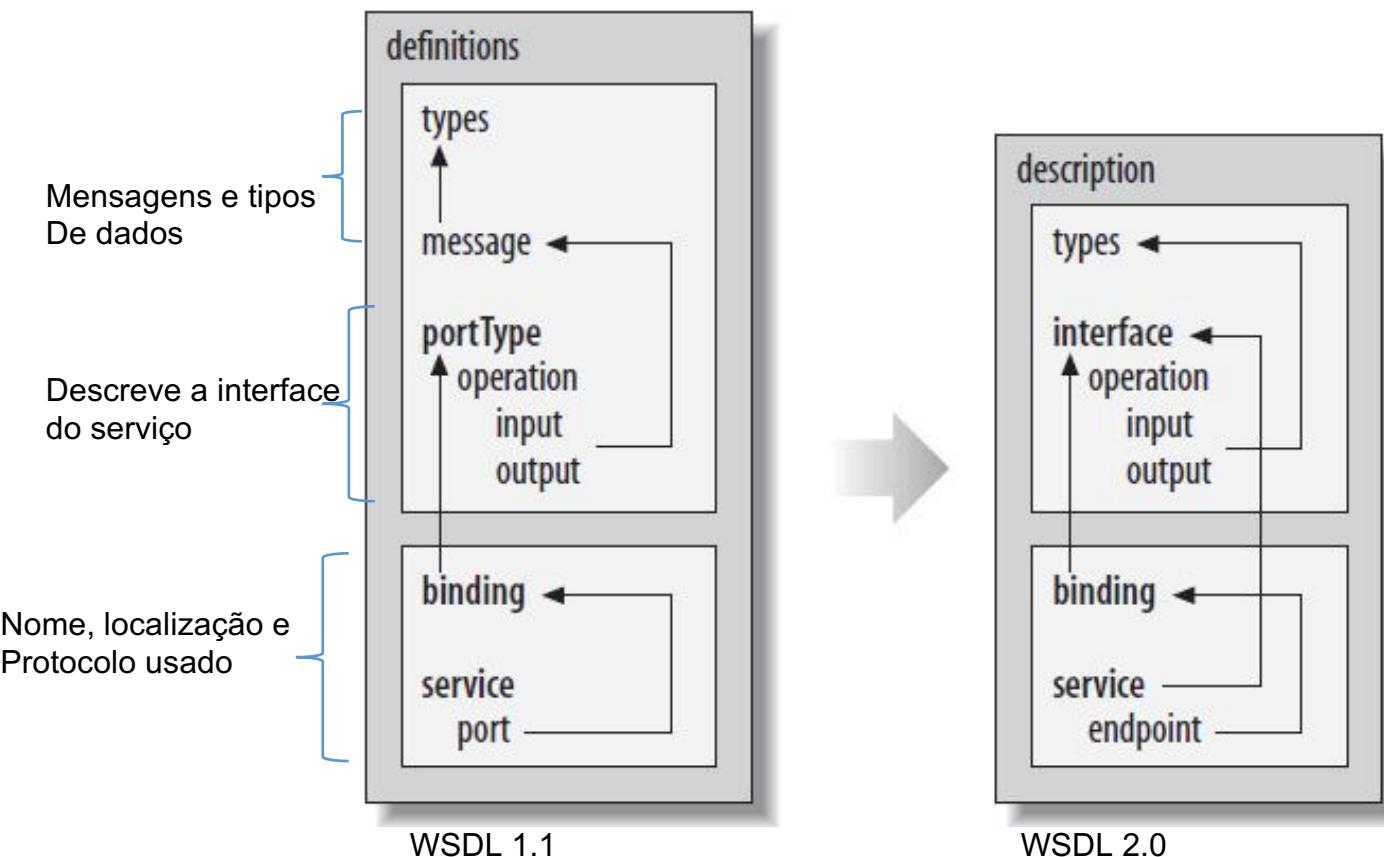
Padrões Fundamentais

- Existem 5 padrões fundamentais de Web Services sendo que 2 deles (XML, HTTP) já existiam antes e foram utilizados
 - XML: usado como formato geral para representar as mensagens e tipos de dados
 - HTTP: usado para transportar mensagens pela internet
 - WSDL: usada para definir a interface do serviço com base em XML
 - SOAP: protocolo de mais alto nível que o HTTP que especifica o formato para trocar as mensagens entre os web Services (usa o HTTP)
 - UDDI: usado para publicar e procurar serviços

WSDL e XML

- Um documento WSDL é um documento XML que descreve Web Services como um conjunto de pontos de serviço (endpoints) que operam baseados em trocas de mensagens.
- As operações e mensagens relativas a um serviço são descritas de forma abstrata e em seguida ligados a protocolos de rede e formatos de mensagens concretos como o objetivo de definir um ponto de serviço

WSDL - Estrutura do documento



Elementos do WSDL

- Service - um container para conjunto de funções de expostas
- Port - Endereço ou ponto de conexão para o *Web Service*.
- Binding - Especifica o tipo de porta (ex: define o estilo de *SOAP binding*).
- Seções de *binding* também definem as operações;
- Port Type – Definem as operações que podem ser executadas, e as mensagens trocadas para executar a operação;
- Message - A mensagem contém as informações necessárias para executar a operação;
- Types – Tipos de dados envolvidos

```
<service name="CustomerService" >
    <port name="CustomerPort"
        binding="tns:CustomerSOAPBinding">
        <soap:address
            location="http://soa-in-practice.com/customer11"/>
    </port>
</service>
```

Serviço de Nome “CustomerService”
disponível na URL “<http://soa-in-practice.com/customer11>”
e que utiliza o Binding “CustomerSOAPBinding”

Exemplo de WSDL (cont.)



```
<binding name="CustomerSOAPBinding"
    type="tns:CustomerInterface" >      (1)
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />  (2)
<operation name="getCustomerAddress">
    <soap:operation
        soapAction="http://soa-in-practice.com/getCustomerAddress" />
    <input>
        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
</operation>
</binding>
```

Define o protocolo e o formato usados para prover o serviço

Define para qual interface o binding é definido (1)

Define este binding para usar SOAP com http (2)

Exemplo de WSDL (cont.)



```
<portType name="CustomerInterface" > (1)
  <operation name="getCustomerAddress"> (2)
    <input message="tns:getCustomerAddressInput" /> (3)
    <output message="tns:getCustomerAddressOutput" /> (4)
  </operation>
</portType>
```

Define a Interface “CustomerInterface” (1)

Interface contém apenas uma operação “getCustomerAddress” (2)

Define as mensagens que serão enviadas quando a operação for chamada

Input message → Request (3)

Output Message → Response (4)

```
<message name="getCustomerAddressInput">
    <part name="params" element="xsd1:getCustomerAddress"/>
</message>
<message name="getCustomerAddressOutput">
    <part name="params" element="xsd1:getCustomerAddressResponse"/>
</message>
```

Define as duas mensagens referenciadas em <portType>
Usam tipos de dados definidos na seção <types>

Exemplo de WSDL (cont.)



```
<types>
  <xsd:schema
    targetNamespace="http://soa-in-practice.com/xsd
    xmlns="http://soa-in-practice.com/xsd">

    <xsd:element name="getCustomerAddress">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="customerID" type="xsd:long"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="getCustomerAddressResponse" type="Address"/>
      <xsd:complexType name="Address">
        <xsd:sequence>
          <xsd:element name="street" type="xsd:string"/>
          <xsd:element name="city" type="xsd:string"/>
          <xsd:element name="zipCode" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>

  </xsd:schema>
</types>
```

**Input
"customerID" de
tipo long**

**Output "Address"
composto por 3 strings:
street, address e zip code**

SOAP request de acordo com exemplo anterior

```
<?xml version='1.0' ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        ...
    </soap:Header>
    <soap:Body>
        <getCustomerAddress xmlns="http://soa-in-practice.com/xsd">
            <customerID>12345678</customerID>
        </getCustomerAddress >
    </soap:Body>
</soap:Envelope>
```

SOAP response de acordo com exemplo anterior

```
<?xml version='1.0' ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        ...
    </soap:Header>
    <soap:Body>
        <getCustomerAddressResponse xmlns="http://soa-in-practice.com/xsd">
            <address>
                <street>Gaussstr. 29</street>
                <city>Braunschweig</city>
                <zipCode>D-38106</zipCode>
            </address>
        </getCustomerAddressResponse>
    </soap:Body>
</soap:Envelope>
```

SOAP enviado por HTTP

the request might look as follows (SOAP 1.1):

```
POST /customer11 HTTP/1.1
Host: soa-in-practice.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: " http://soa-in-practice.com/getCustomerAddress"

<?xml version='1.0' ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  ...
</soap:Envelope>
```

A corresponding response might have the following format:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

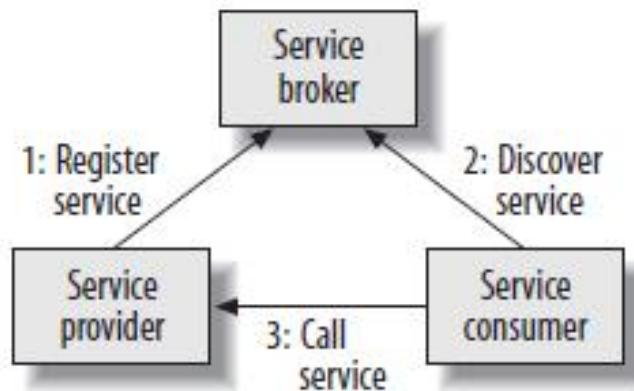
<?xml version='1.0' ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  ...
</soap:Envelope>
```

Arquitetura WS

- Camadas
 - Descoberta
 - Publicação (UDDI /direta)
 - Descrição dos serviços (WSDL)
 - Troca de Mensagens baseadas em XML (SOAP)
 - Rede (HTTP, SMTP, FTP)

UDDI

- Universal Description, Discovery, and Integration
- A idéia é funcionar como uma espécie de catálogo de serviços.
 - Serviços são publicados por service providers
 - Serviços podem ser pesquisados e localizados por brokers
 - Consumers estão interessados nos serviços



UDDI

- Idéia original do UDDI era servir como um ponto central (MarketPlace para serviços)
 - Espécie de páginas amarelas
- Na prática a idéia não obteve êxito
- Algumas empresas dão suporte a idéia mas não são um ponto central único para qualquer serviço
 - Ex: SAP UDDI

REST

Cenário para SOA

- SOAP, WSDL etc...
 - Muita complexidade
 - Verbose

Proposta REST

- Representação e Manipulação de RECURSOS na Web
 - Baseia-se na simplificação da comunicação Cliente/Servidor
- Escrita de Serviços Web
 - Simplificação
 - Interoperabilidade
 - Flexibilidade



Tese de Roy Fielding (2000)
Architectural styles and the design of network-based software architectures

O que rest não é

- Um protocolo
- Um padrão
- Um substituto para SOAP
 - Seria mais uma alternativa

Objetivos REST

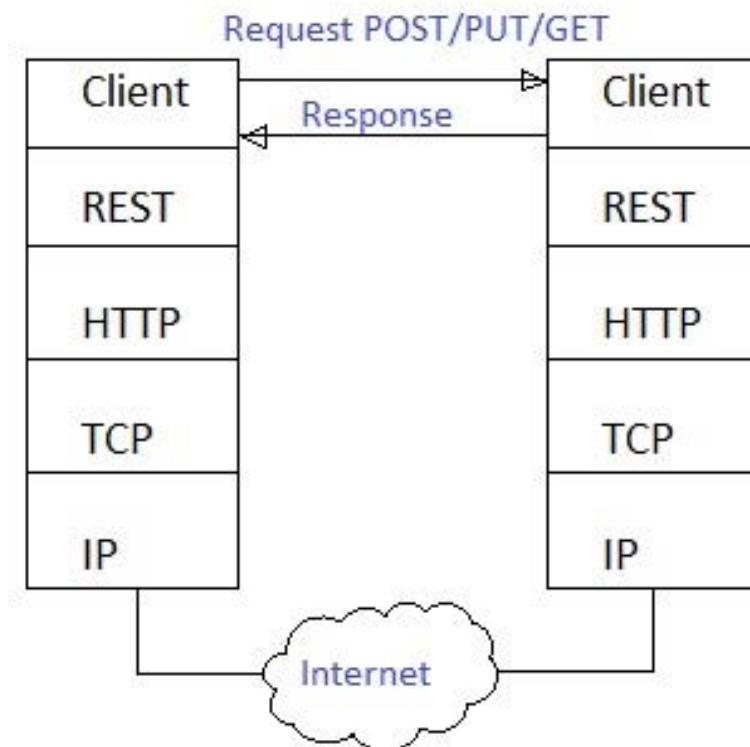
- Escalabilidade
- Simplicidade
- Modificabilidade
- Portabilidade
- Confiabilidade

REST

- Representational State Transfer
 - Transferência de Estado Representacional
- Estilo arquitetônico
 - Tecnicamente não é um padrão
- Ideia: uma rede de páginas da web onde o cliente avança por um aplicativo selecionando links
 - Quando o cliente atravessa o link, acessa um novo recurso,i.e., transfere o estado
- Usa padrões existentes, por exemplo, HTTP
 - REST é uma arquitetura sobre como comunicar entre um cliente e um servidor

Um estilo arquitetural

- REST já é a arquitetura da Web como funciona hoje
- É um modelo de arquitetura de software
 - Usado para descrever sistemas distribuídos como WWW (World Wide Web)
- Ele foi desenvolvido em paralelo com o protocolo HTTP



Cliente-Servidor

- Separação de Interesses
- Cliente e Servidor são independentes um do outro
- Cliente não sabe nada sobre o recurso mantido pelo servidor
- Servidor responde enquanto os pedidos sejam feitos de forma correta
- Objetivo: independência de plataforma (interoperabilidade) e melhoria de escalabilidade

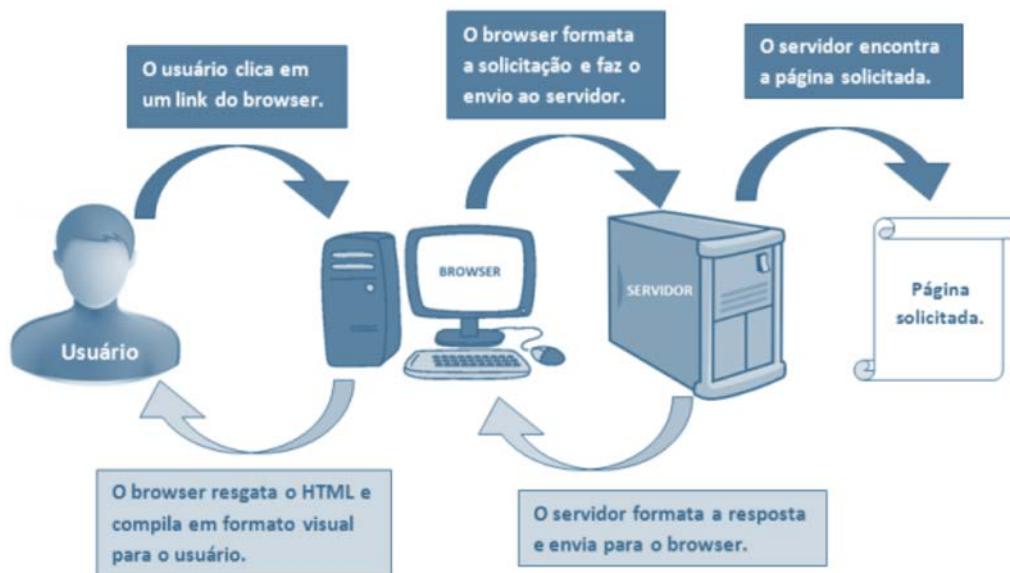
Como funciona?

- Cliente solicita um recurso específico do servidor
 - O servidor responde a essa solicitação, entregando o recurso solicitado
- Um modelo no qual as representações dos recursos são transferidas entre o cliente e o servidor
- A Web, como sabemos, já está nesta forma!

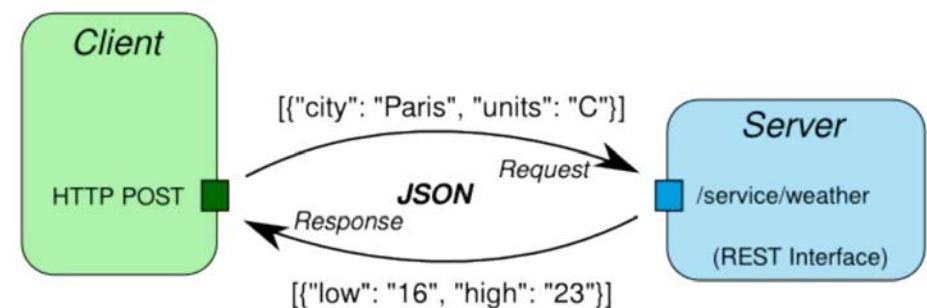
IMPORTANTE!

- Servidor não possui informações sobre nenhum cliente!
- Sem diferença entre chamadas recorrentes de um mesmo cliente

Analogia



JSON / REST / HTTP



Recursos

- Tudo em REST são recursos
- Recursos são mapeamentos consistentes de um identificador
 - como um caminho de URL
 - para alguns conjuntos de visualizações no estado do lado do servidor
- Endereçável exclusivamente por meio de um URI

<https://viacep.com.br/ws/60125035/json>

Roy T. Fielding

“Se uma visão não é adequada às suas necessidades, sinta-se livre para criar um diferente recurso que lhe disponibilize uma visão melhor”

“Estas visões não necessitam ter nada a ver com como a informação é armazenada no servidor... Elas precisam apenas ser compreensíveis (e acionáveis) pelo cliente”

HTTP Normal

REQUEST

```
GET /news/ HTTP/1.1
Host: example.org
Accept-Encoding: compress, gzip
User-Agent: Python-httplib2
```

GET request to «<http://example.org/news/>»
Method = GET

HTTP Normal

RESPONSE

HTTP/1.1 200 Ok

Date: Thu, 07 Aug 2008 15:06:24 GMT

Server: Apache

ETag: "85a1b765e8c01dbf872651d7a5"

Content-Type: text/html

Cache-Control: max-age=3600

<!DOCTYPE HTML>

...

HTTP

- O Pedido é um recurso identificado por uma URI
 - URI = Unified Resource Identifier
- No exemplo passado, o recurso é `http://example.org/news/`
- Recursos (addressability) é muito importante
 - Todo recurso em REST é endereçado por uma URI.
- Para mudar o estado do sistema, simplesmente mude um recurso.

Exemplos URI

- **http://localhost:9999/restapi/books**
 - GET – get all books
 - POST – add a new book
- **http://localhost:9999/restapi/books/{id}**
 - GET – get the book whose id is provided
 - POST – update the book whose id is provided
 - DELETE – delete the book whose id is provided

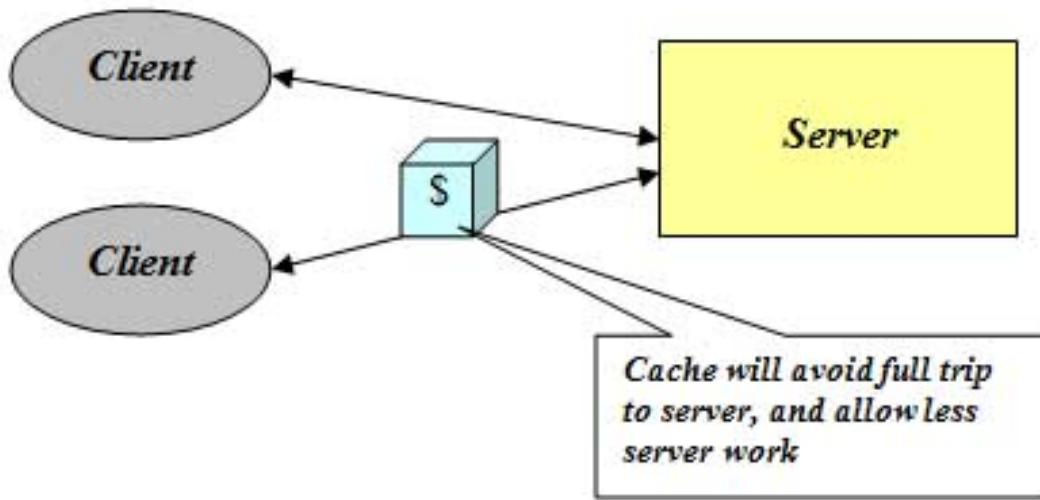
Características

- Recursos: Estado e funcionalidades da aplicação são abstraídos em recursos
 - URI: Todo recurso é unicamente acessado por URLs
 - Interface Uniforme
 - Interfaces para transferir estado entre cliente e recursos consistem de métodos (ações)
 - Stateless
 - Cacheable

Stateless

- Cada pedido é independente de outros pedidos
- Nenhum dado relacionado à sessão do cliente é armazenado no servidor
 - Se houver necessidade de dados específicos relacionados à sessão de um cliente, estes devem ser guardados pelo cliente e transferidos a cada pedido
- Não manter informações sobre as sessões de clientes facilita a escalabilidade
- Problemas:
 - Cliente deve carregar informações necessárias para cada pedido.
Aumento do tráfego de rede
 - Servidor pode ficar sobrecarregado para validação de pedidos

Cacheable



Respostas HTTP devem ser *cacheable* por clientes.
Importante para desempenho.

Interface Uniforme

- Todos os recursos são acessados com uma interface genérica (baseada em HTTP)
- Facilita o gerenciamento da comunicação
- Facilita a evolução das aplicações
 - Clientes e Servidor evoluem de forma independente
- Analogia LEGO:
 - Pode haver diferentes peças, mas poucas formas (uniformes) de conectá-las

Métodos HTTP

GET	Buscar recurso	Seguro, Idempotente
PUT	Atualizar recurso	Idempotente
POST	Incluir recurso	
DELETE	Remover recurso	
HEAD		
OPTIONS		

CRUD

Operação	Método
Create	POST
Read	GET
Update	PUT ou POST
Delete	DELETE

Códigos para resposta

Código	Significado
200 OK	The resource was read, updated or deleted
201 Created	The resource was created
400 Bad Request	The data sent in the request was bad
403 Not Authorized	The Principal named in the request was not authorized to perform this action
404 Not Found	The resource does not exist
409 Conflict	A duplicate resource could not be created
500 Internal Server Error	A service error occurred

Exemplo



A screenshot of a web browser window titled "Windson". The address bar shows the URL <https://viacep.com.br/ws/60115222/json/>. The page content displays a JSON object representing a Brazilian address:

```
{  
  "cep": "60115-222",  
  "logradouro": "Avenida Rui Barbosa",  
  "complemento": "de 2101/2102 ao fim",  
  "bairro": "Joaquim Távora",  
  "localidade": "Fortaleza",  
  "uf": "CE",  
  "unidade": "",  
  "ibge": "2304400",  
  "gia": ""  
}
```

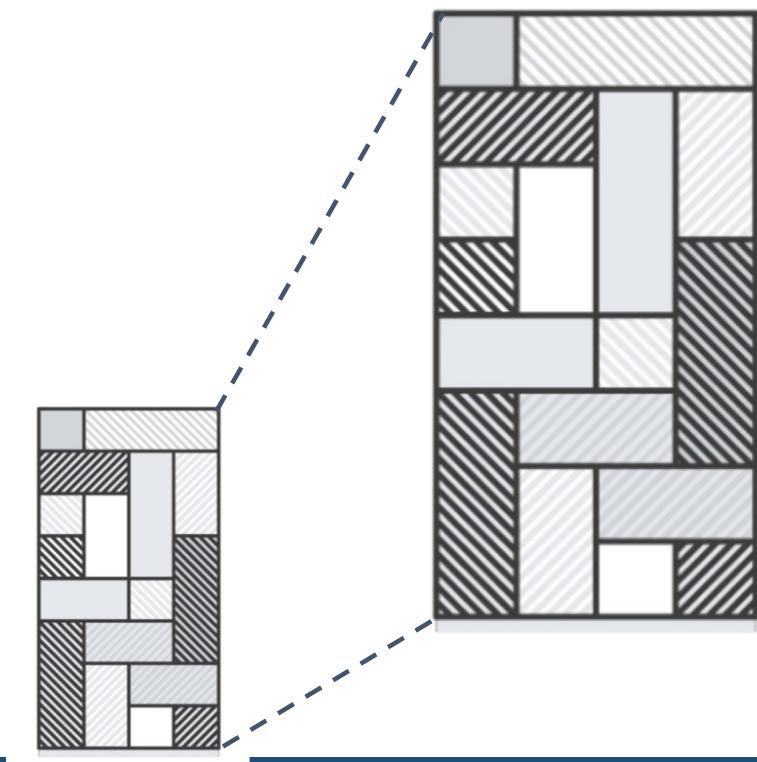
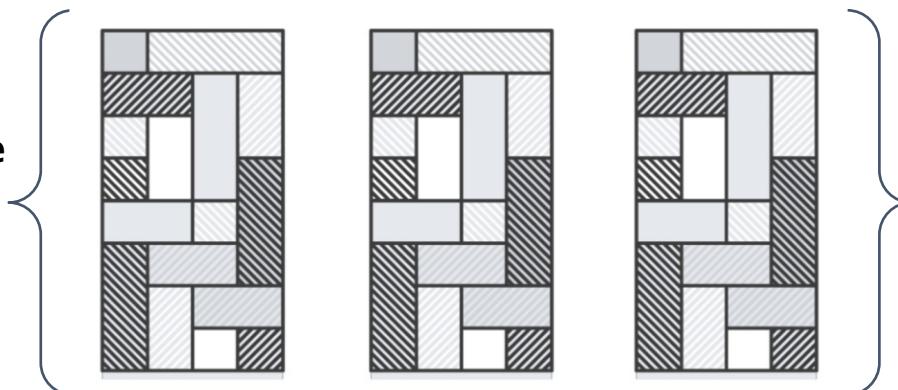
E o momento atual?

Escalabilidade

Vertical

- Desafios
 - Dificuldade de escalar
 - Arquitetura difícil de manter e evoluir

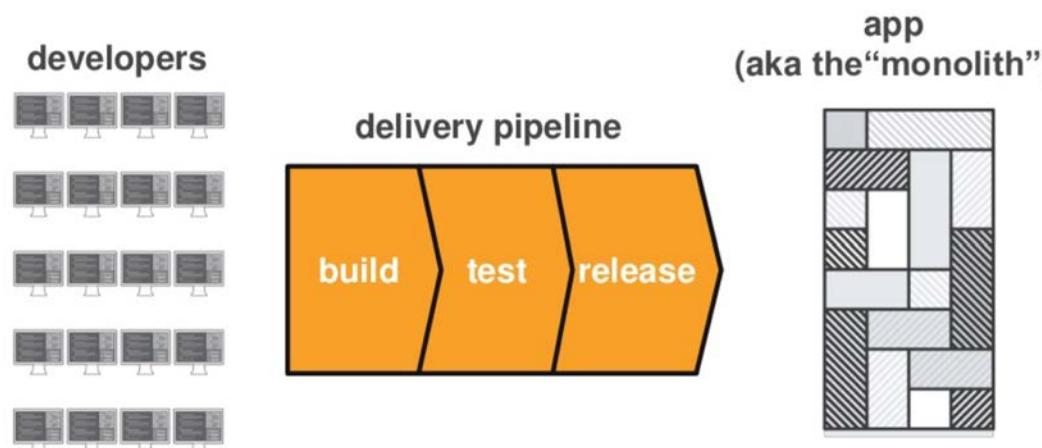
Escalabilidade
Horizontal



Microservices

□ Desafios

- Ciclos longos de build/test/release
- Longo tempo para adicionar novas funcionalidades
- Falta de agilidade, inovação, frustração de clientes



Aplicações Monolíticas

- ❑ Tudo integrado



Desenvolvimento de Software para a Nuvem

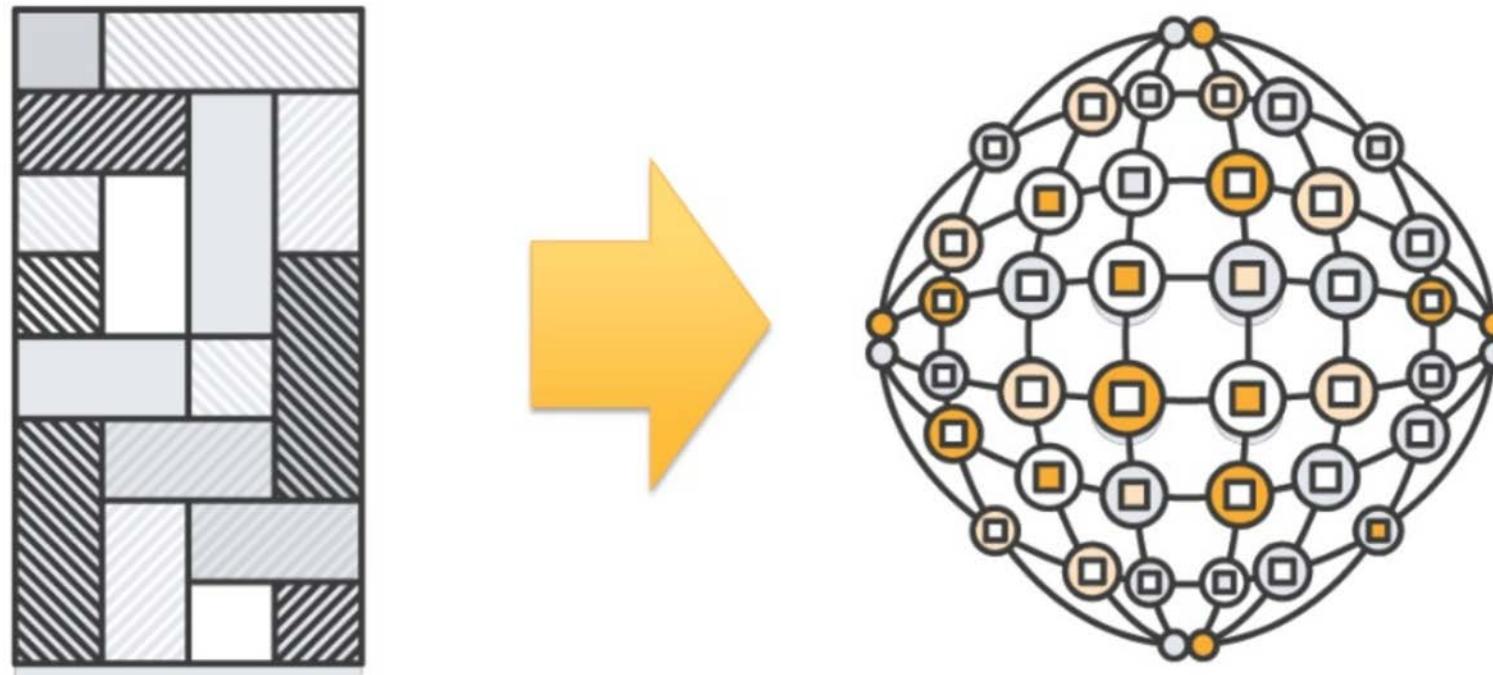
Arquitetura de Microsserviços

- ❑ É uma abordagem que desenvolve um aplicativo único como uma suíte de pequenos serviços, cada um executando seu próprio processo e se comunicando através de mecanismos leves, muitas vezes em uma API com recursos HTTP.
 - ❑ Serviços funcionam através de mecanismos de deploy independentes totalmente automatizados.
 - ❑ Há o mínimo possível de gerenciamento centralizado desses serviços, que podem ser escritos em diferentes linguagens de programação e utilizam diferentes tecnologias de armazenamento de dados.

Arquitetura de Microsserviços

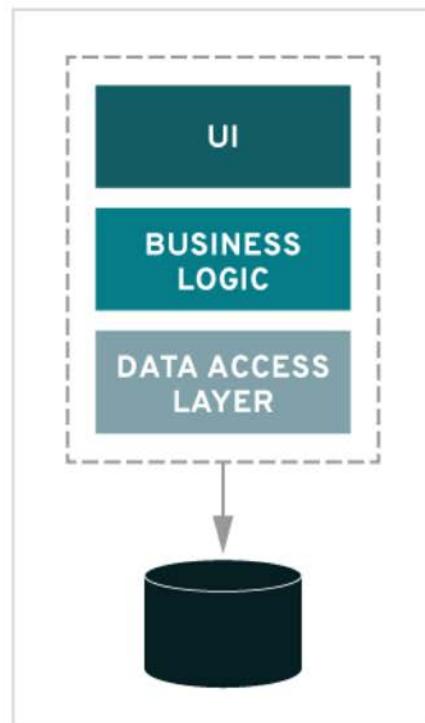
- ❑ Decompõe a aplicação por funções básicas.
- ❑ Cada função é denominada um serviço e pode ser criada e implantada de maneira independente.
- ❑ cada serviço individual pode funcionar ou falhar sem comprometer os demais.

Aplicações Monolíticas x Microserviços



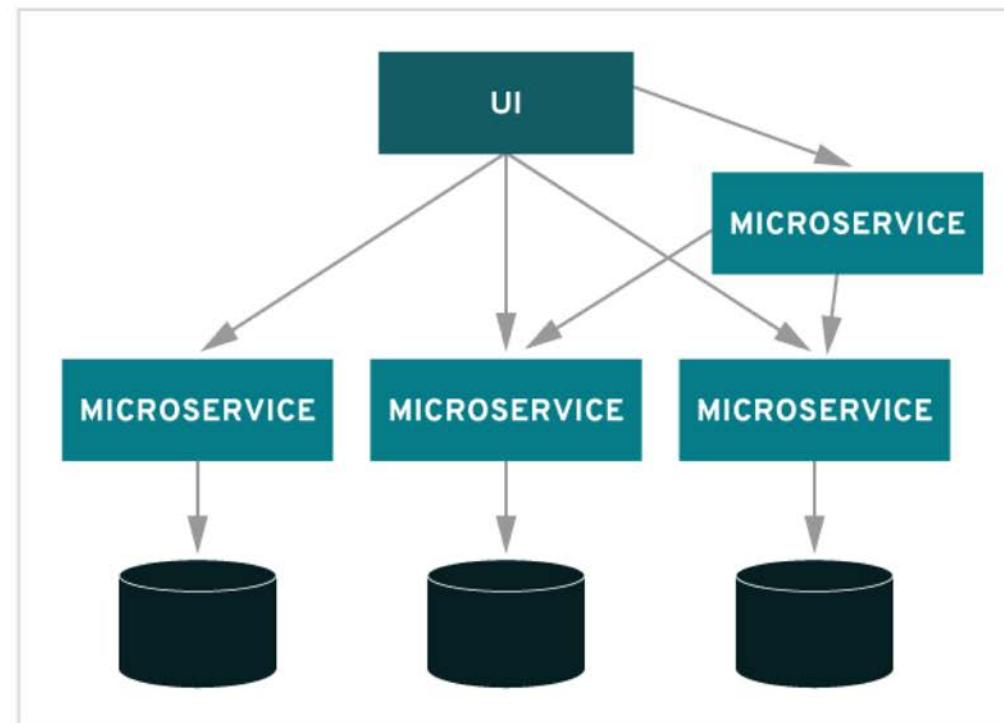
Aplicações Monolíticas x Microserviços

MONOLITHIC



VS.

MICROSERVICES

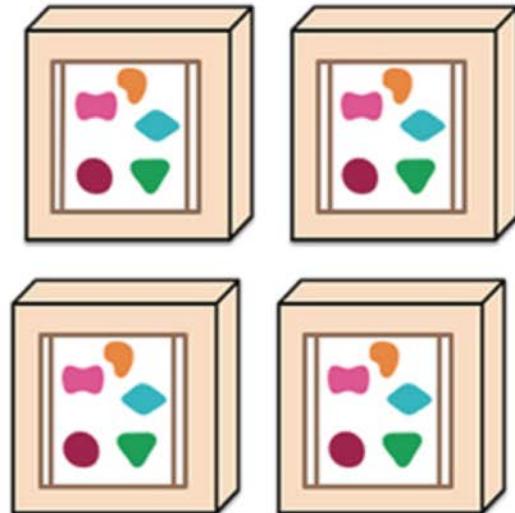


Aplicações Monolíticas x Microserviços

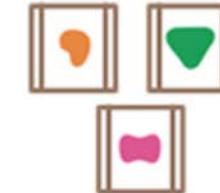
Um aplicativo monolítico tem todas as suas funcionalidades em um único processo...



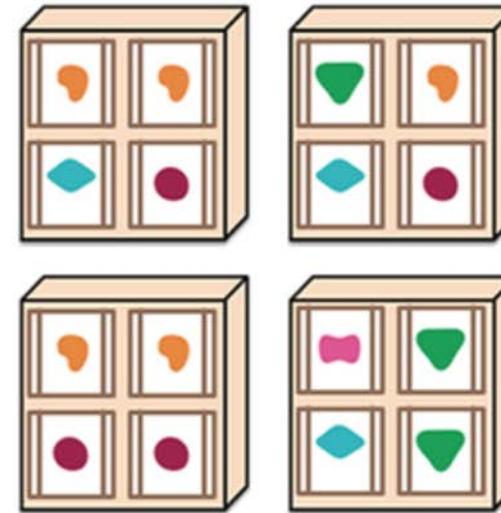
...e escala replicando o monolito em servidores múltiplos



A arquitetura de micro-serviços coloca cada elemento de funcionalidade em um serviço separado...



...e escala distribuindo os serviços entre os servidores, replicando por demanda.



Benefícios da arquitetura de microsserviços

- Mais open source: a adoção de novos frameworks/tecnologias pode ser aplicada em um microsserviço de cada vez.
- Os microsserviços são pequenos
 - Melhor isolamento de falhas; iniciam mais rápido.
- Lançamento no mercado com mais rapidez
- Altamente escalável
- Resiliente
- “Fácil” de implementar e implantar
- Acessível

Desvantagens da arquitetura de microserviços

- Complexidades adicionais do desenvolvimento de sistemas distribuídos:
 - Aplicações muito mais complexas e constituídas por mais elementos;
 - Uso de transações distribuídas ou consistência eventual;
 - Gerenciamento de um número maior de aplicações.
- Para ser utilizada de forma eficaz, a arquitetura de microserviços exige um alto nível de automação.

Desafios ao migrar para arquitetura de microsserviços

- Mudanças não somente nas aplicações, mas também no modo como as pessoas trabalham.
 - Mudanças organizacionais e culturais
- Complexidade e eficiência
- Identificação das dependências entre os serviços.
- Testes de integração e testes end-to-end podem ser mais difíceis e importantes como jamais foram, pois uma falha pode reverberar em outros serviços.

Desafios ao migrar para arquitetura de microsserviços

- Controle de versão: ao atualizar para versões novas, a compatibilidade com versões anteriores pode ser rompida.
- Implantação (Configuração inicial): é necessário investir em automação, pois a complexidade dos microsserviços é demais para a implantação manual.
- Geração de logs: é necessário ter logs centralizados para unificar e facilitar o acompanhamento dos mesmos.

Desafios ao migrar para arquitetura de microsserviços

- Monitoramento: é importante uma visão centralizada do sistema para identificar as fontes de problemas.
- Depuração: a depuração remota não é uma opção e não funciona com centenas de serviços.
- Conectividade: considere a detecção de serviços, seja de maneira centralizada ou integrada.



Dúvidas?