# Sistemas Distribuídos

## Web Services

Edeyson Andrade Gomes

www.edeyson.com.br

# SUMÁRIO

- Visão geral
- Arquitetura Web Services
- Desenvolvimento

# Visão Geral

- Web Services
  - Usados para disponibilizar serviços na WEB, podendo ser acessados por outras aplicações;
  - Identificados por uma URI (*Unique Resource Identifier*);
  - Descritos e definidos usando XML;
  - Baseado em tecnologias padrão - XML e HTTP;
    - SOAP (Simple Object Access Protocol) - troca de mensagens entre aplicações e Web Services.

# Princípios

- Web-based Protocols
  - Web-services baseados em HTTP
    - Protocolos podem atravessar firewalls e podem trabalhar em ambientes heterogêneos
- Interoperability
  - SOAP define um padrão comum que permite a diferentes sistemas interoperarem
- XML-based (XML schema)
  - Documentos legíveis por máquinas
- Modularity
  - Componentes de Serviços são reusáveis e podem ser compostos

# Princípios

- Availability
- Machine-readable description
  - Usado para identificar a interface, a localização e acessar informações
- Implementation-independence
- Published
  - Repositórios de serviço
  - Busca de descritores

# SOAP

- Protocolo projetado para invocar aplicações remotas através de RPC (Chamadas Remotas de Procedimento) ou trocas de mensagens, em um ambiente independente de plataforma e linguagem de programação.

- Um padrão normalmente aceito para utilizar-se com Web Services.

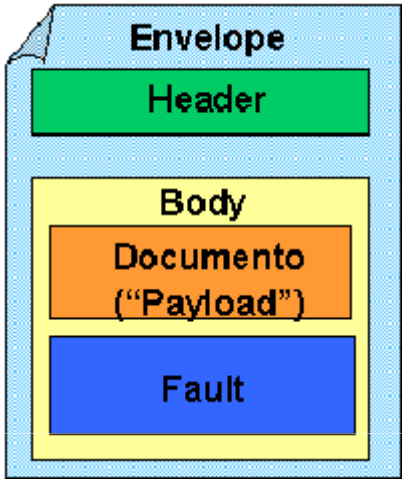
- Pretende garantir a interoperabilidade e intercomunicação entre diferentes sistemas, através da utilização de uma linguagem (XML) e mecanismo de transporte (HTTP) padrões.

# SOAP

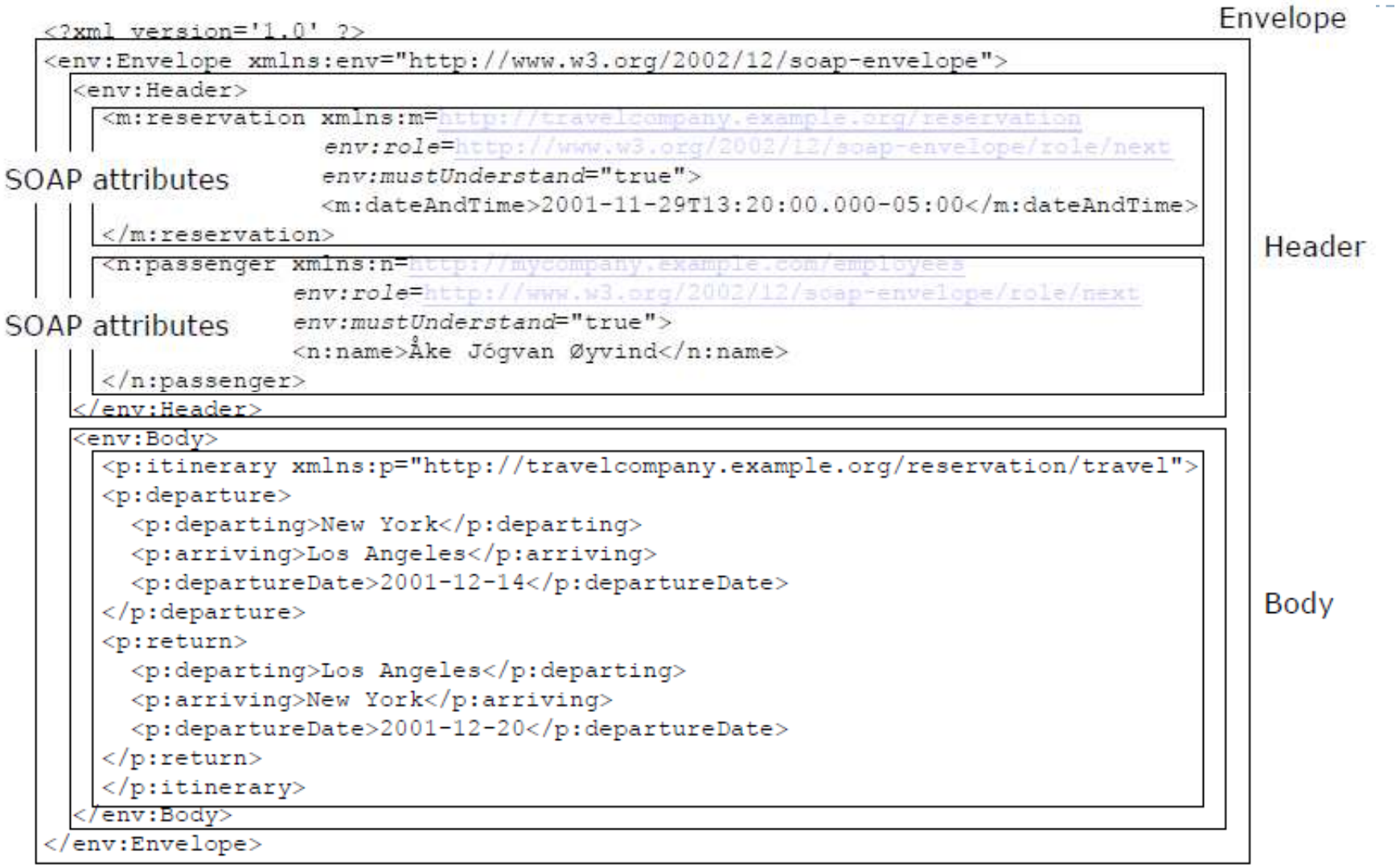
- Definido pelo consórcio W3C;
- Protocolo baseado em XML para a troca de informações em um ambiente distribuído;
- Padrão de utilização com Web Services;
- Normalmente utiliza HTTP como protocolo de transporte;.

# SOAP

- Formato de Mensagem:
  - **Envelope**
    - Elemento raiz obrigatório do documento XML.
    - Pode conter declarações de *namespaces* e também atributos adicionais como o que define o estilo de codificação - como os dados são representados no documento XML.
  - **Header**
    - Carrega informações adicionais.
    - Por exemplo, se a mensagem deve ser processada por um determinado nó intermediário.
  - **Body**
    - Elemento obrigatório que contém a informação a ser transportada para o destino final.
    - O elemento opcional *Fault* é usado para carregar mensagens de status e erros retornadas pelos "nós" ao processarem a mensagem.



# SOAP


Envelope
SOAP attributes
SOAP attributes
Header
Body
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
<env:Header>
<m:reservation xmlns:m="http://travelcompany.example.org/reservation"
env:role="http://www.w3.org/2002/12/soap-envelope/role/next"
env:mustUnderstand="true">
<m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
</m:reservation>
<n:passenger xmlns:n="http://mycompany.example.com/employees"
env:role="http://www.w3.org/2002/12/soap-envelope/role/next"
env:mustUnderstand="true">
<n:name>Åke Jógvan Øyvind</n:name>
</n:passenger>
</env:Header>
<env:Body>
<p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
<p:departure>
<p:departing>New York</p:departing>
<p:arriving>Los Angeles</p:arriving>
<p:departureDate>2001-12-14</p:departureDate>
</p:departure>
<p:return>
<p:departing>Los Angeles</p:departing>
<p:arriving>New York</p:arriving>
<p:departureDate>2001-12-20</p:departureDate>
</p:return>
</p:itinerary>
</env:Body>
</env:Envelope>

# SOAP RPC

- **Chamada SOAP RPC**
  - 1. Chamadas de RPC são encapsuladas (*serialização*) segundo o padrão SOAP.
  - 2. O serviço remoto faz o processo contrário ao receber a mensagem, extraindo o método e seus parâmetros.
  - 3. A aplicação servidora processa a chamada e envia uma resposta ao cliente.
  - 4. A resposta é encapsulada e enviada pela rede.
  - 5. Na máquina cliente, a resposta é desencapsulada e repassada para a aplicação cliente.

# SOAP RPC

- A especificação SOAP (definida pela W3C) define as seguintes informações, como necessárias em toda chamada de RPC:
  - A URI do objeto alvo;
  - O nome do método;
  - Os parâmetros do método (requisição ou resposta);
  - Uma assinatura opcional do método;
  - Um cabeçalho (header) opcional.

# SOAP RPC

- Requisição

```
<?xml version='1.0' ?>
    <env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope" >
    <env:Header>
      <t:transaction xmlns:t="http://thirdparty.example.org/transaction"
                     env:encodingStyle="http://example.com/encoding"
                     env:mustUnderstand="true">5</t:transaction>
    </env:Header>
    <env:Body>
      <m:chargeReservation env:encodingStyle="http://www.w3.org/2002/12/soap-encoding"
                           xmlns:m="http://travelcompany.example.org/">
        <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
          <m:code>FT35ZBQ</m:code>
        </m:reservation>
        <o:creditCard xmlns:o="http://mycompany.example.com/financial">
          <n:name xmlns:n="http://mycompany.example.com/employees">
              Åke Jógvan Øyvind </n:name>
          <o:number>123456789099999</o:number>
          <o:expiration>2005-02</o:expiration>
        </o:creditCard>
      </m:chargeReservation>
    </env:Body>
    </env:Envelope>
```

transaction information

TID

method invocation

parameter 1

parameter 2

# SOAP RPC

- Resposta

```
<?xml version='1.0' ?>
    <env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope" >
    <env:Header>
      <t:transaction xmlns:t=http://thirdparty.example.org/transaction
                     env:encodingStyle=http://example.com/encoding
                     env:mustUnderstand="true">5</t:transaction>
    </env:Header>
    <env:Body>
      <m:chargeReservationResponse
                     env:encodingStyle=http://www.w3.org/2002/12/soap-encoding
                     xmlns:m="http://travelcompany.example.org/">
        <m:code>FT35ZBQ</m:code>
        <m:viewAt> http://travelcompany.example.org/reservations?code=FT352
        </m:viewAt>
      </m:chargeReservationResponse>
    </env:Body>
    </env:Envelope>
```

method result

output parameters

# WSDL

- WSDL ou *Web Service Description Language* é uma linguagem baseada em XML, utilizada para descrever um Web Service.
- Com a descrição do serviço a ser utilizado, a implementação do Web Service pode ser feita em qualquer linguagem de programação.
  - Normalmente são utilizadas linguagem construídas para interação com a WEB, como Java Servlets ou ASP, que, em seguida, chamam um outro programa ou objeto.

# WSDL

- Quando um cliente deseja enviar uma mensagem para um determinado Web Service, ele obtém a descrição do serviço (através da localização do respectivo documento WSDL), e em seguida constrói a mensagem, passando os tipos de dados corretos (parâmetros, etc) de acordo com a definição encontrada no documento.

- Em seguida, a mensagem é enviada para o endereço onde o serviço está localizado, a fim de que possa ser processada.

- O Web Service, quando recebe esta mensagem, valida-a conforme as informações contidas no documento WSDL.

- À partir de então, o serviço remoto sabe como tratar a mensagem, sabe como processá-la e como montar a resposta ao cliente.

# WSDL

```
<?xml version="1.0">
    <definitions name="StockQuote>
    <types>
          <schema>
                definition of types in XML Schema ……
          </schema>
    </types>
    <message name="GetTradePriceInput">
          definition of a message....
    </message>
    <portType name="StockQuotePortType">
          <operation name="GetLastTradePrice">
                definition of an operation ……
          </operation>
    </portType>
    <binding name="StockQuoteSoapBinding">
          definition of a binding ……
    </binding>
    <service name="StockQuoteService">
          <port name="StockQuotePort">
                definition of a port ……
          </port>
    </service>
    </definitions>
```

# WSDL

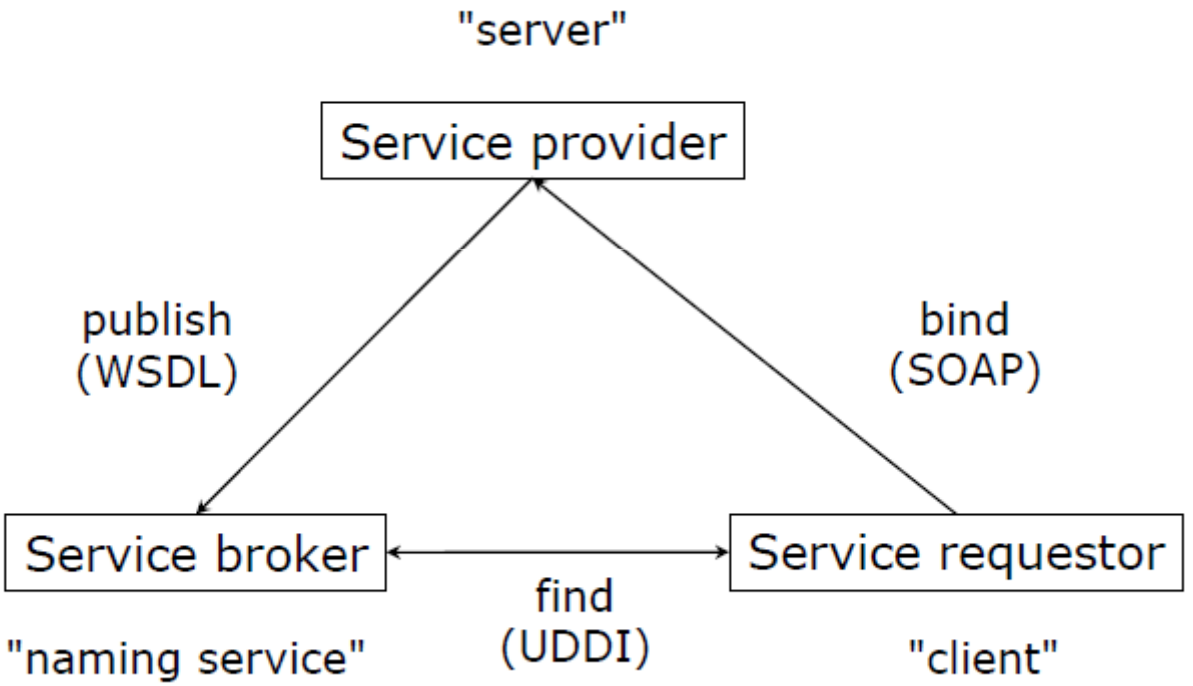
- Definição de Tipos

```
<?xml version="1.0"?>
<schema targetNamespace="http://example.com/stockquote/schemas"
        xmlns="http://www.w3.org/2000/10/XMLSchema">

    <element name="TradePriceRequest">
        <complexType>
            <all>
                <element name="tickerSymbol" type="string"/>
            </all>
        </complexType>
    </element>
    <element name="TradePrice">
        <complexType>
            <all>
                <element name="price" type="float"/>
            </all>
        </complexType>
    </element>
</schema>
```

# Arquitetura

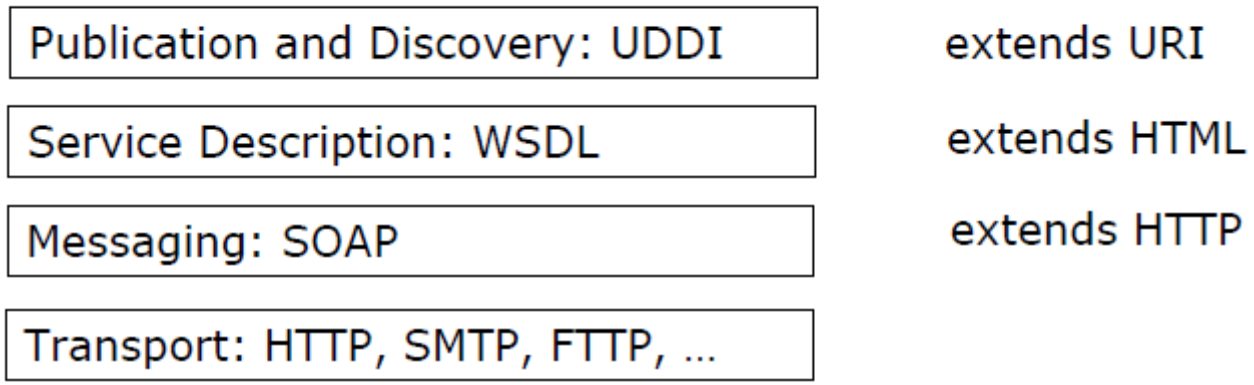
- Service-oriented architecture



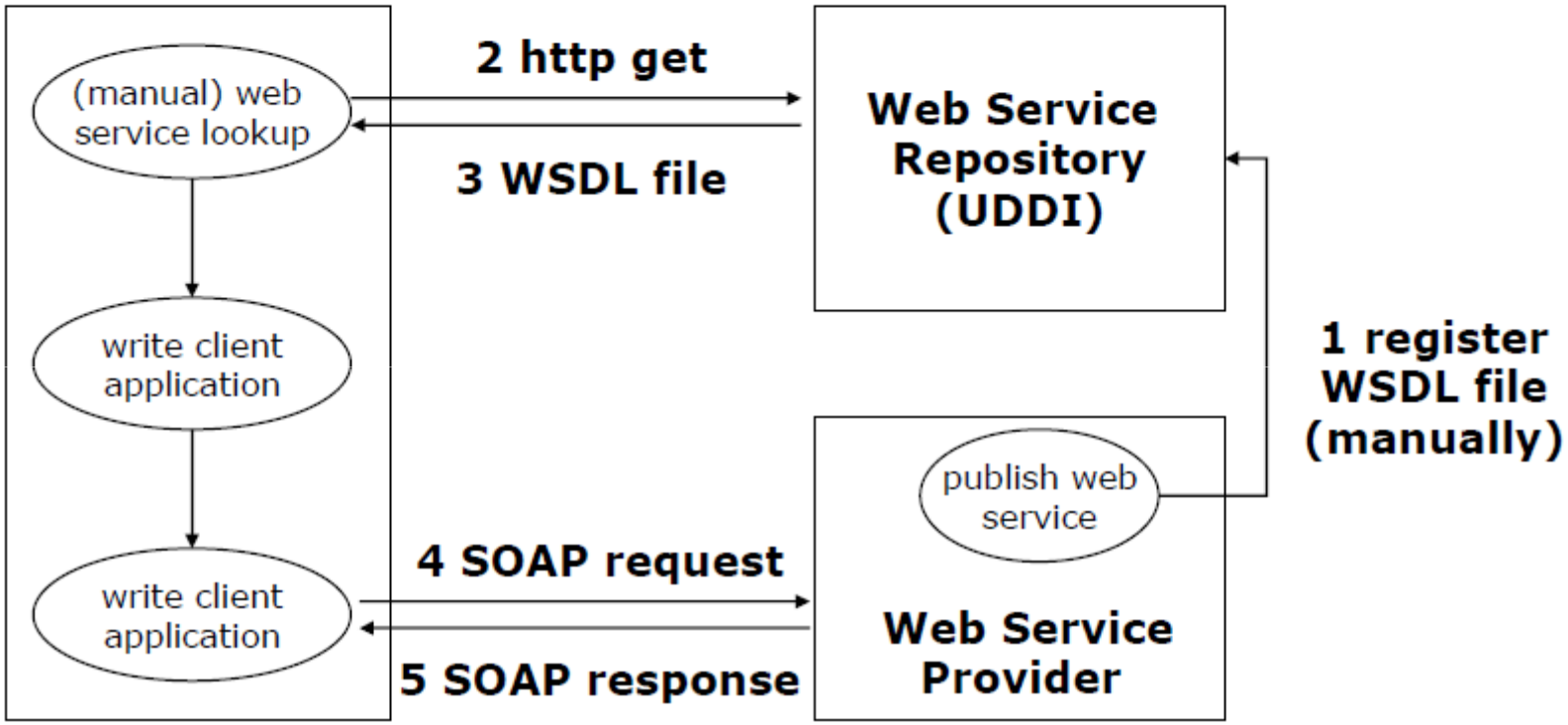

# Arquitetura

- The Web architecture consists of three components
  - The *service providers that publish available services and offer bindings for* services
  - The *service brokers that allow service providers to publish their services* (register and categorize). They provide also mechanisms to locate services and their providers
  - The *service requestor that uses the service broker to find a service and then* invokes (or binds) the service offered by a service provider.

# Pilha WS

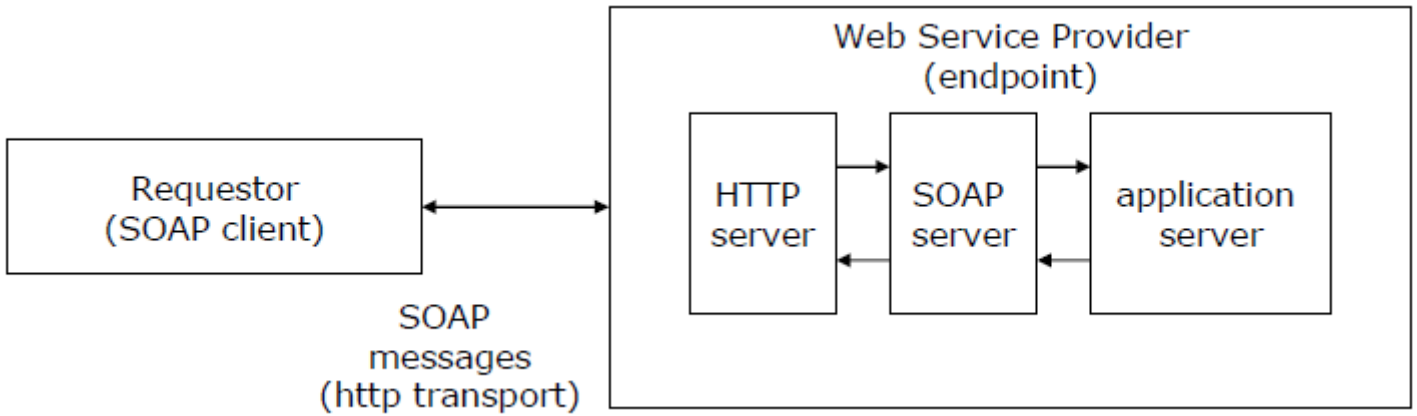

Publication and Discovery: UDDI
Service Description: WSDL
Messaging: SOAP
Transport: HTTP, SMTP, FTTP, ...
extends URI
extends HTML
extends HTTP

# Cenário WS


(manual) web service lookup
write client application
write client application
2 http get
3 WSDL file
4 SOAP request
5 SOAP response
Web Service Repository (UDDI)
publish web service
Web Service Provider
1 register WSDL file (manually)

# Implementação WS

- Application Server (web service-enabled)
  - provides implementation of services and exposes it through WSDL/SOAP
  - implementation in Java, as EJB, as .NET (C#) etc.
- SOAP server
  - implements the SOAP protocol
- HTTP server
  - standard Web server
- SOAP client
  - implements the SOAP protocol on the client site



# Pilha WS

- A set of standards for implementing web services
  - **UDDI provides a mechanism for clients to find web services**
    - UDDI registry is similar to a CORBA trader or a DNS for business applications.
  - **WSDL defines services as collections of network endpoints or *ports***
    - A port is defined by associating a network address with a binding (servers)
    - a collection of ports define a service
  - **SOAP is a message layout specification that defines a uniform way of**
    - passing XML-encoded data and to bind to HTTP as the underlying communication protocol
    - SOAP is basically a technology to allow for "RPC *over the Web*"

# OOA X SOA

# Arquitetura Orientada a Serviço

- Paradigma dos serviços web
  - Evolução lógica dos sistemas orientados a objeto para sistemas orientados a serviços.
  - SOA (Arquitetura Orientada a Serviço) deve especificar:
    - Informação sobre o que o serviço faz;
    - Endereço onde este está localizado;
    - Como este deve ser invocado;
    - Qualificação do serviço e política de segurança utilizada por ele;
    - Uma interface pública, seja para uma Intranet ou Internet.

# Arquitetura Orientada a Serviço

- Quando se utiliza *Web Services* para a construção de tal arquitetura, as soluções consistem de coleções de serviços autônomos, identificados por URL, com interfaces documentadas através de WSDL e processando mensagens XML.
- SOA é um complemento a abordagens orientadas a objetos (OO) e procedurais.
- SOA é uma arquitetura flexível regida pelo princípio de responder uma requisição em curto espaço de tempo, reduzindo custo, risco de desenvolvimento e manutenção do serviço.

# Benefícios dos WS

- Fraco acoplamento
  - Possibilita que clientes e provedores adotem a tecnologia que for mais apropriada aos seus interesses.
- Interoperabilidade
  - Qualquer serviço pode interagir com outro.
- Usa mensagens baseadas em XML
  - Permite um modelo flexível para troca de dados que é independente do ambiente de desenvolvimento.
- Não precisa abandonar investimentos já realizados em softwares:
  - Aplicações podem ser utilizadas por *Web Services* adicionando-se um *front-end (Façade)*.

# Benefícios dos WS

- O uso de padrões e protocolos utilizados pela Internet
  - Muitas empresas já possuem a maior parte da infraestrutura necessária para suportar *Web Services*.
- Reduz complexidade por encapsulamento
  - Todos os componentes são serviços.
  - O importante é o tipo de serviço provido, e não como ele foi implementado.
  - Isto reduz a complexidade do sistema, pois os desenvolvedores das aplicações não precisam se preocupar com detalhes da implementação dos serviços que eles estão invocando.
- Um conjunto de forncedores de tecnologia concordaram em apoiar um conjunto de padrões
  - IBM, Microsoft, Sun resolveram apoiar este conjunto que define como sistemas diferentes devem interagir entre si.

# XML

- XML (*Extensible Markup Language)* surgiu da necessidade em se estabelecer uma linguagem que desse ao usuário a possibilidade de criar próprias *tags* de marcação.
- A XML é vista como "um mecanismo flexível, que acomoda a estrutura de aplicações específicas.
- Ela oferece um mecanismo para codificar as informações manipuladas pela aplicação e estrutura básica.
- Ou seja, ela pode ser definida como um conjunto de padrões para troca e publicação de informações de uma forma estruturada.

# XML Schema

- DTD *(Document Type Definition)* é a linguagem de modelagem para a XML.
  - É um mecanismo para descrever a estrutura de documentos.
  - É um mecanismo para descrever cada objeto (elemento, atributo, etc) que possa aparecer no documento, começando com elementos.
- A utilização de DTDs determina a forma como se descreve a estrutura de um documento e como a aplicação o lê.

# SOAP

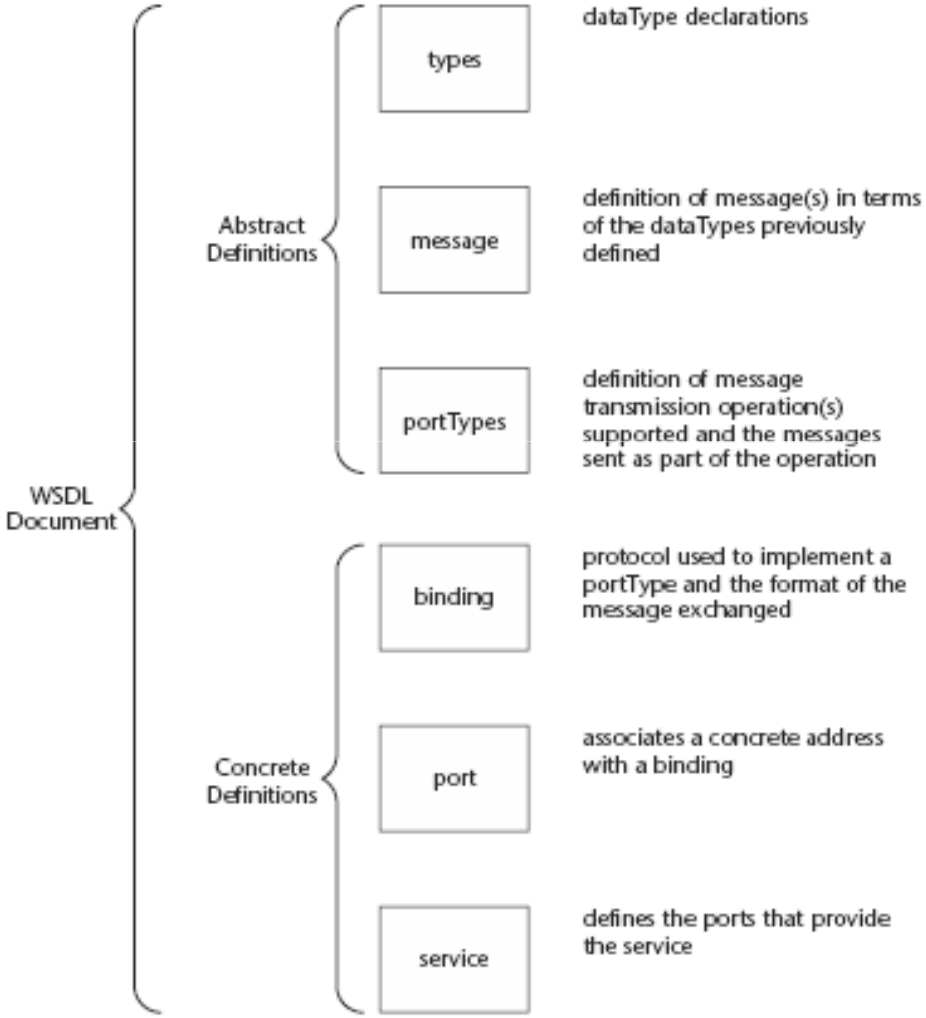
- SOAP *(Simple Object Access Protocol)* é um protocolo baseado em XML para troca de informações entre computadores.
  - Ele permite que aplicações clientes se conectem facilmente a serviços remotos e invoquem métodos remotos.
- SOAP é um protocolo projetado para invocar aplicações remotas através de RPC (*Remote Procedure Calls* ) ou trocas de mensagens em um ambiente independente de plataforma e linguagem de programação.

# SOAP

- Foi criadopara ser simples e extensível;
- Provê um *fra mework* para descrever o conteúdo das mensagens e instruções sobre os processos;
- Todas as suas mensagens são no formato XML;
- É um protocolo de transporte independente.
  - HTTP é um dos transportes suportados, por isso SOAP pode rodar sobre a infraestrutura da Internet já existente;
- Não existe uma *distributed garbage collection*. Conseqüentemente, chamadas por referência não são suportadas por SOAP. Clientes SOAP não detém nenhum estado de referência a objetos remotos;
- É independente e não é amarrado a nenhuma linguagem de programação.

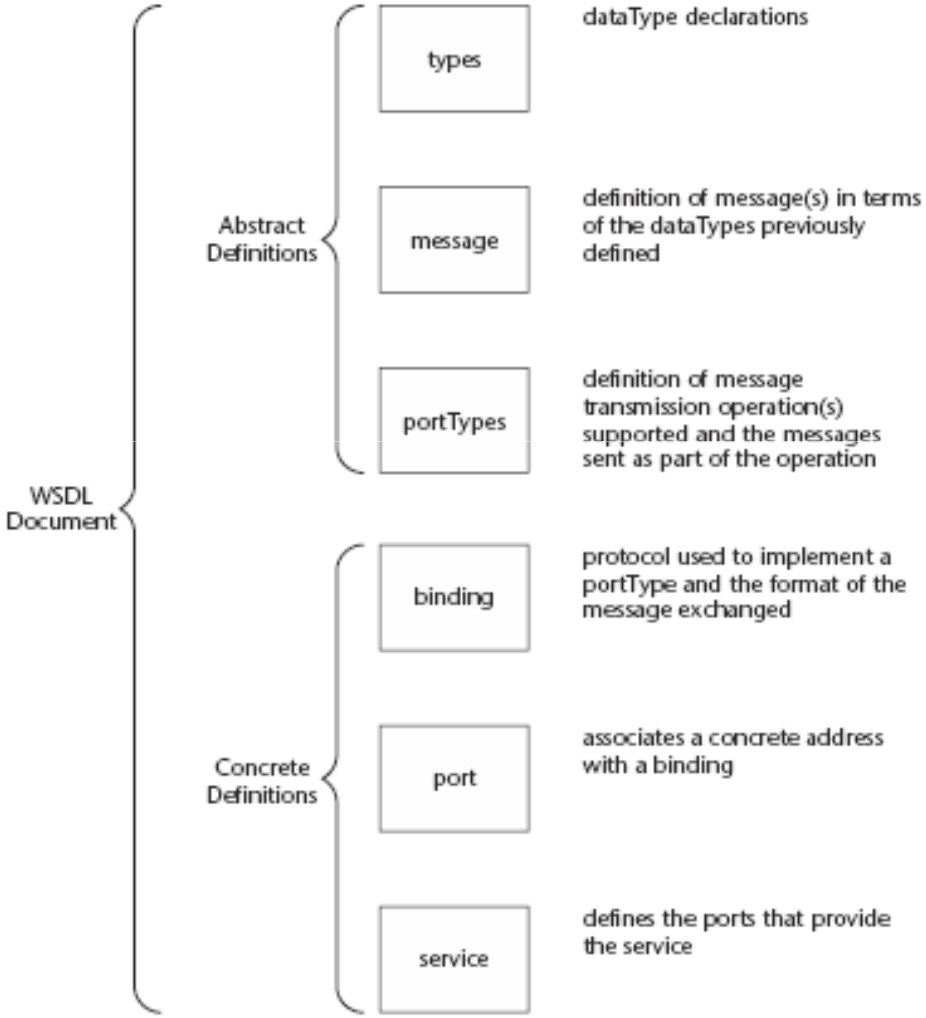
# WSDL

- *types* descreve todos os tipos de dados usados entre o cliente e o servidor.
- O WSDL não é preso a um sistema de tipos exclusivo, mas sim, utiliza a especificação *W3C XML Schema* como padrão.
- Porém, se o serviço utilizar apenas *XML Schema* construída sobre tipos simples como *string* e *integer*, o elemento *types* não será utilizado.



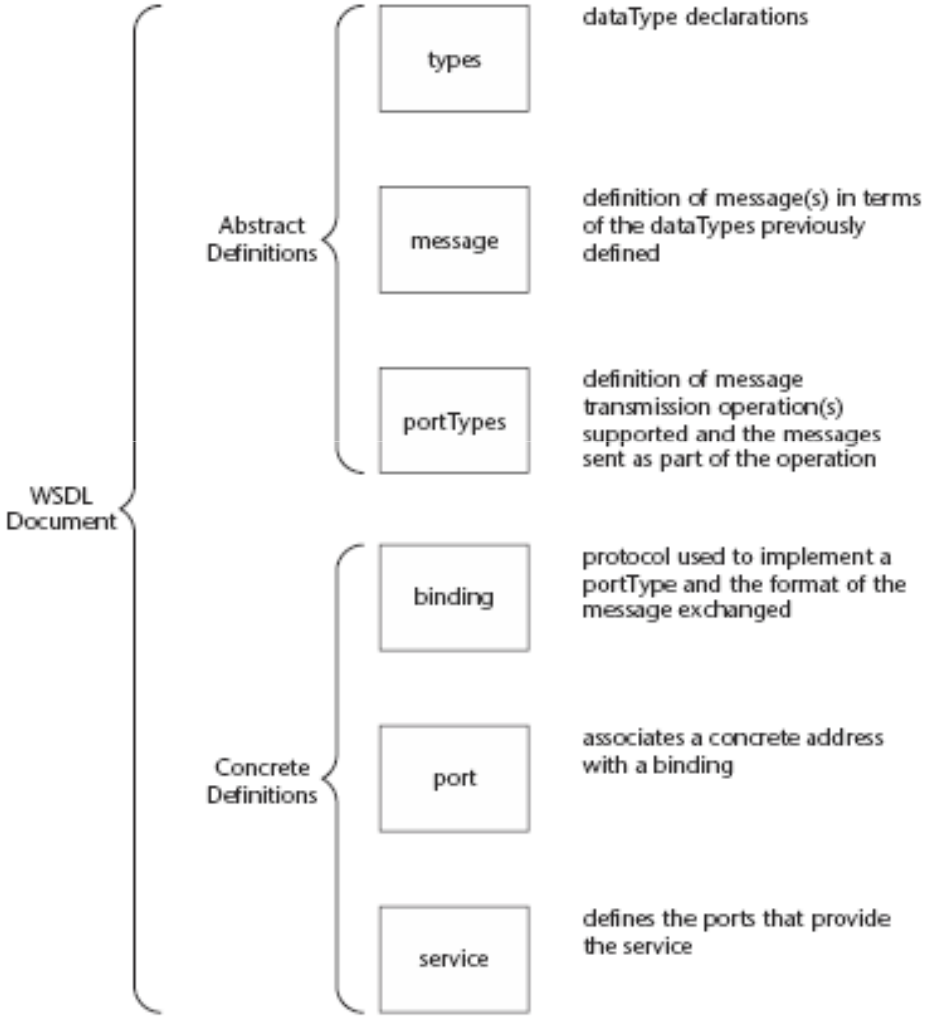
# WSDL

- *message* é um resumo de definição dos tipos de dados sendo trafegados.
- Este elemento pode conter uma ou mais partes, e essas partes podem ser comparadas a parâmetros de uma função.



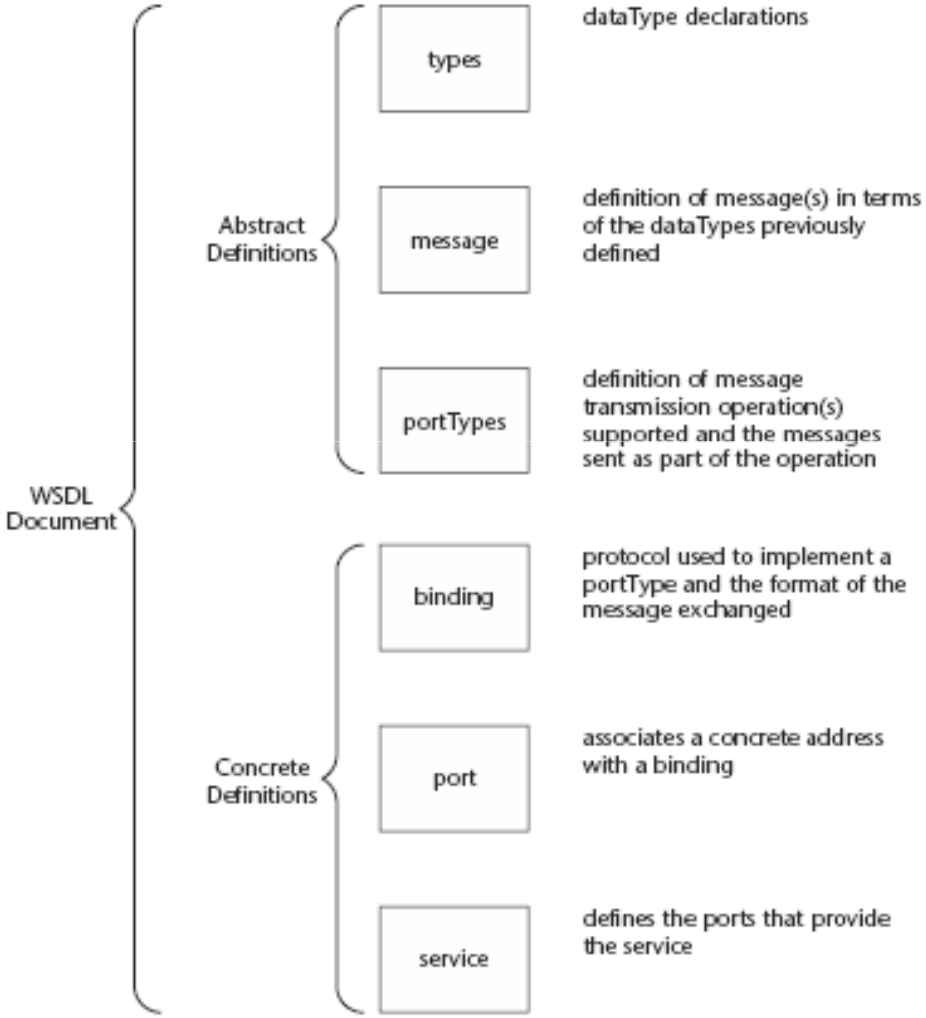
# WSDL

- *port type* são *operações* mapeadas para um ou mais *end points*, definindo uma coleção de operações para serem ligadas.
- Ou seja, eles são um resumo da configuração das operações suportadas por um ou mais *end points*. Estes por sua vez são o endereço alvo do serviço de comunicação.



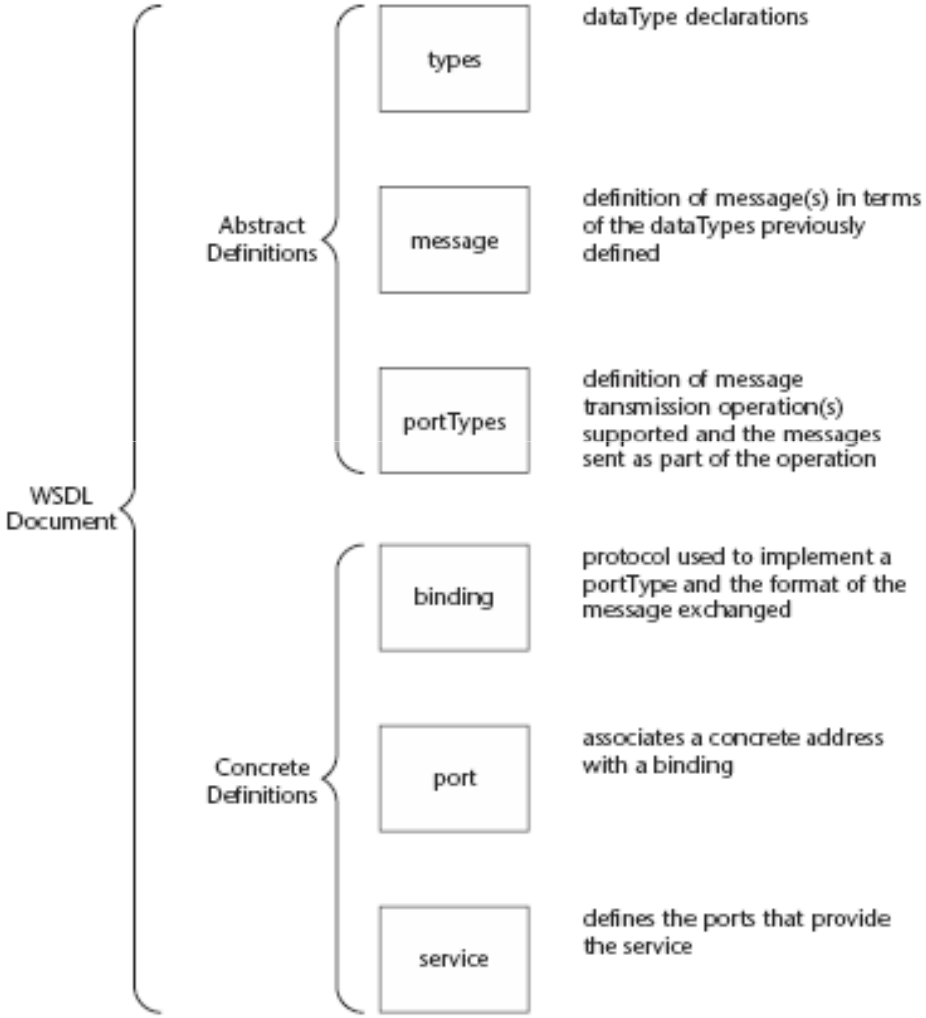
# WSDL

- O *binding* define o formato da mensagem e detalhes de protocolos para cada *port*, que corresponde a um *end point* definido como uma combinação de uma ligação e um endereço de rede.



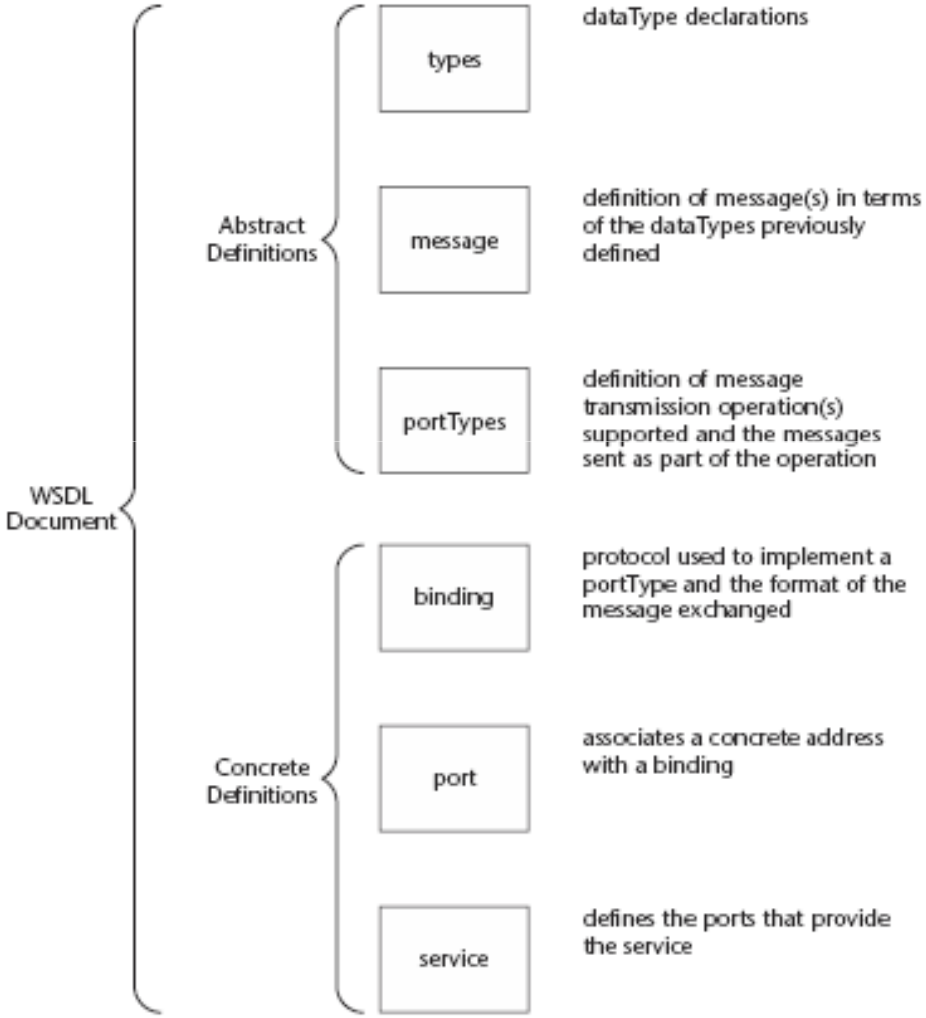
# WSDL

- *Ports* é o elemento mais importante no WSDL, ele define um *WS*, suas operações desempenhadas e mensagens que estão envolvidas.
- Eles estão presentes no *operation types* ( tipos de operação) que são constituídas por: *one-way, request-response, solicite-response* e *notification.*

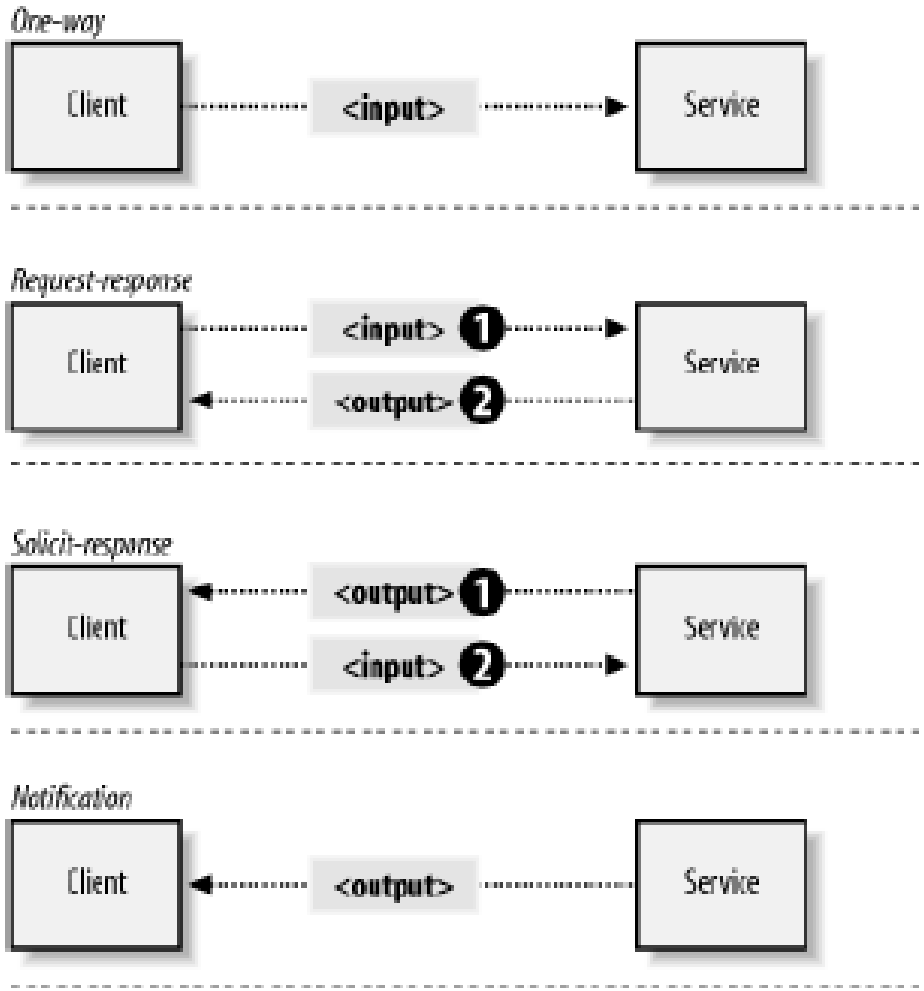


# WSDL

- O *binding* define o formato da mensagem e detalhes de protocolos para cada *port*, que corresponde a um *end point* definido como uma combinação de uma ligação e um endereço de rede.
- *service* definirá o endereço que chamará o serviço especificado, e normalmente inclui a URL para invocar o serviço *SOAP*.



# Operações suportadas pelo WSDL



# Elementos WSDL

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions
    targetNamespace="http://201.32.217.28:8080/axis/webserviceSoma.jws"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://201.32.217.28:8080/axis/webserviceSoma.jws"
    xmlns:intf="http://201.32.217.28:8080/axis/webserviceSoma.jws"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

- *definitions* especifica o nome do documento.
- O uso dos *namespaces* é importante para diferenciar os elementos, e permite ao documento referenciar várias especificações externas, incluindo a especificação WSDL, a especificação SOAP, e a *XML Schema*.
- *targetNamespace* é uma convenção da *XML Schema* para que o documento WSDL possa referenciar a ele próprio.

# Elementos WSDL

- <wsdl:message name="somaRequest">
  <wsdl:part name="valor1" type="xsd:int" />
  <wsdl:part name="valor2" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="somaResponse">
  <wsdl:part name="somaReturn" type="xsd:int" />
  </wsdl:message>
- *part :* especifica os parâmetros da função, que neste caso são valor1 e valor2, e os valores de retorno, neste caso *somaReturn*.
- O *part* contém o elemento *type* - os valores dos atributos tem que estar especificados em um *XML Shema*
- No exemplo, o tipo do valor1 está atribuído a *xsd:int*, onde o prefixo *xsd* referencia ao *namespace* da *XML Schema*, definido anteriormente no elemento *definitions*

# Elementos WSDL

```
<wsdl:portType name="webserviceSoma">
    <wsdl:operation name="soma" parameterOrder ="valor1 valor2">
        <wsdl:input message="impl:somaRequest" name="somaRequest" />
        <wsdl:output message="impl:somaResponse" name="somaResponse">
    </wsdl:operation>
</wsdl:portType>
```

- *port type* define uma operação chamada de "soma".
- *Operation* consiste em uma *input message* (somaRequest), e uma *output message* (somaResponse)

# Elementos WSDL

- <wsdl:binding name="webserviceSomaSoapBinding" type="impl:webserviceSoma">

- *binding* mostra detalhes de como a *port type operation* irá ser transmitida.
- *Bindings* podem estar disponíveis por múltiplos transportes, incluindo *HTTP GET, HTTP POST* ou *SOAP*.

- O atributo de *type* referencia o *port type* definido anteriormente.
- No exemplo, o *binding* irá prover detalhes específicos de como a *operation* "Soma" irá ser transportada pela Internet

# Elementos WSDL

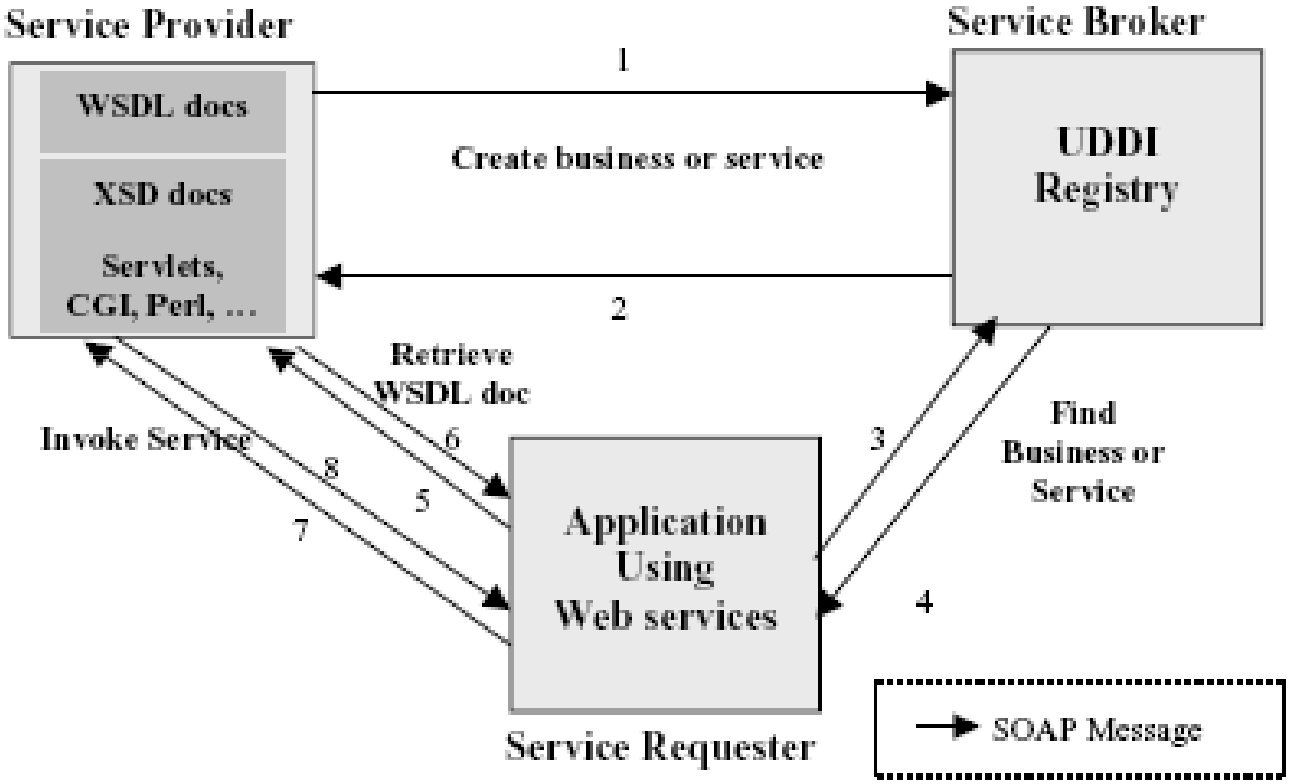
- <wsdlsoap:address location="http://201.32.217.28:8080/axis/webserviceSoma.jws" />

- *Service* diz onde o serviço está hospedado.

- Um documento WSDL tem muita redundância. As descrições de *Web Services* permitem componentes especificados independentemente.
  - *operations* referenciam *messages*.
  - *bindings* referenciam *operations*.
  - *messages* definem como as operações e mensagens são transmitidas.
  - Esta redundância é responsável pelo grande tamanho de um documento WSDL comparado com as mensagens SOAP que ele define.
    - Este é o preço da modularidade.

# UDDI

- Uma vez criada a descrição do *WSDL*, todas as informação pertinentes a um serviço já estão prontas para serem acessadas.
- O UDDI (*Universal, Description, Discovery and Integration)* irá provêr um método padronizado para se publicar e descobrir informações sobre um *Web Service*.


Service Provider
WSDL docs
XSD docs
Servlets,
CGI, Perl, …
Service Broker
UDDI
Registry
1
Create business or service
2
Retrieve
WSDL doc
6
Invoke Service
8
5
7
3
Find
Business or
Service
Application
Using
Web services
4
Service Requester
SOAP Message

# Composição de WS

- *Web Services Composition* é a habilidade de um negócio prover serviços com características adicionais para os seus clientes através da composição de *Web Services* básicos, possivelmente oferecidos por diferentes companhias.
- *Web Services Composition* se refere a agregação e composição de *Web Services* para a criação de processos de negócios.

# Composição de WS

- A especificação de um *Web Service* é expressa pelo WSDL, que especifica apenas a sintaxe das mensagens que entram e saem de um programa.
- A ordem em que as mensagens devem ser trocadas entre serviços tem que estar descritas separadamente em uma especificação de fluxo de mensagens.
- A grande diferença entre o WSDL e uma linguagem de composição é vista quando se trata dos estados dos processos.

# Composição de WS

- O único estado suportado pelo WSDL é o estado entre enviar e receber uma mensagem em uma operação de *request-response* ou *solicit-response*.
- As linguagens para composição deve guardar os estados dos processos (que são mais complexos do que uma simples *request-response)*. Guardando esses estados será possível determinar o que deverá ser feito, permitindo assim a transação entre os negócios - WS.

# ORCHESTRATION X CHOREOGRAPHY

- Linguagens para composição de *Web Services* são utlizadas para prover a *orchestration* e *choreography* destes.
- *Web Service orchestration* descreve processos de negócios executáveis que interagem com *Web Services* internos ou externos em ordem predefinida para prover a lógica do negócio.
- *Web Service choreography* descreve a colaboração entre processos de negócios, onde a colaboração significa trocas de mensagens realizadas com os serviços *Web*.

# ORCHESTRATION X CHOREOGRAPHY

- em *orchestration*, o processo é sempre controlado pela perspectiva de uma das partes do negócio, enquanto *choreography* é mais colaborativo, onde todas as partes envolvidas no processo descrevem as suas partes na interação.

# ORCHESTRATION X CHOREOGRAPHY

| Web Services Orchestration | Web Services Choreography |
|---|---|
| - inclue ordem de execução nas interações entre *Web Services*. | - traça a sequência das mensagens envolvendo múltiplas partes e fontes. |
| - descreve o fluxo do processo. | - associada a troca de mensagens públicas, não processos executáveis. |
| - pode incluir ambos *Web Services* internos quanto externos, mas o processo é sempre controlado por apenas uma parte. | - mais colaborativa. |

# BPEL4WS

- BPEL4WS define  um modelo e uma gramática para descrever o comportamento de um processo de negócio baseado na interação entre o processo e seus parceiros.
- A interação entre cada um ocorre através de interfaces *web services*,e a estrutura do relacionamento ao nível de interface é encapsulada com o que se chama de *partner link* .

# BPEL4WS

- O processo de negócio define como os diversos serviços e seus participantes irão ser coordenados para atingir o objetivo da interação, assim como o estado e lógica necessária para essa coordenação.
- BPEL4WS possui mecanismo de tratamento de falhas de processo e exceções de negócio.
- Suporta comportamento transacional que irá especificar qual atividade ou composições de atividades devem ser compensadas no caso dessas falhas ou exceções ocorrerem, ou ainda no caso de ser requisitada uma reversão por algum participante.

# BPEL4WS

- Definir relacionamento entre *web services* distintos, com regras de negócios distintas, é uma tarefa complexa.
- Logo, este padrão de integração entre *web services* oferece a opção de descrever publicamente uma interface de comunicação entre serviços utilizando o próprio WSDL.
- O processo de negócio, ou *business process*, com *web services*, disponibilizado pelo BPEL4WS, pode ser descrito de duas formas: *Executable business processess (ochestration)* e *Abstract process* ou *Business protocol (choreography)*.