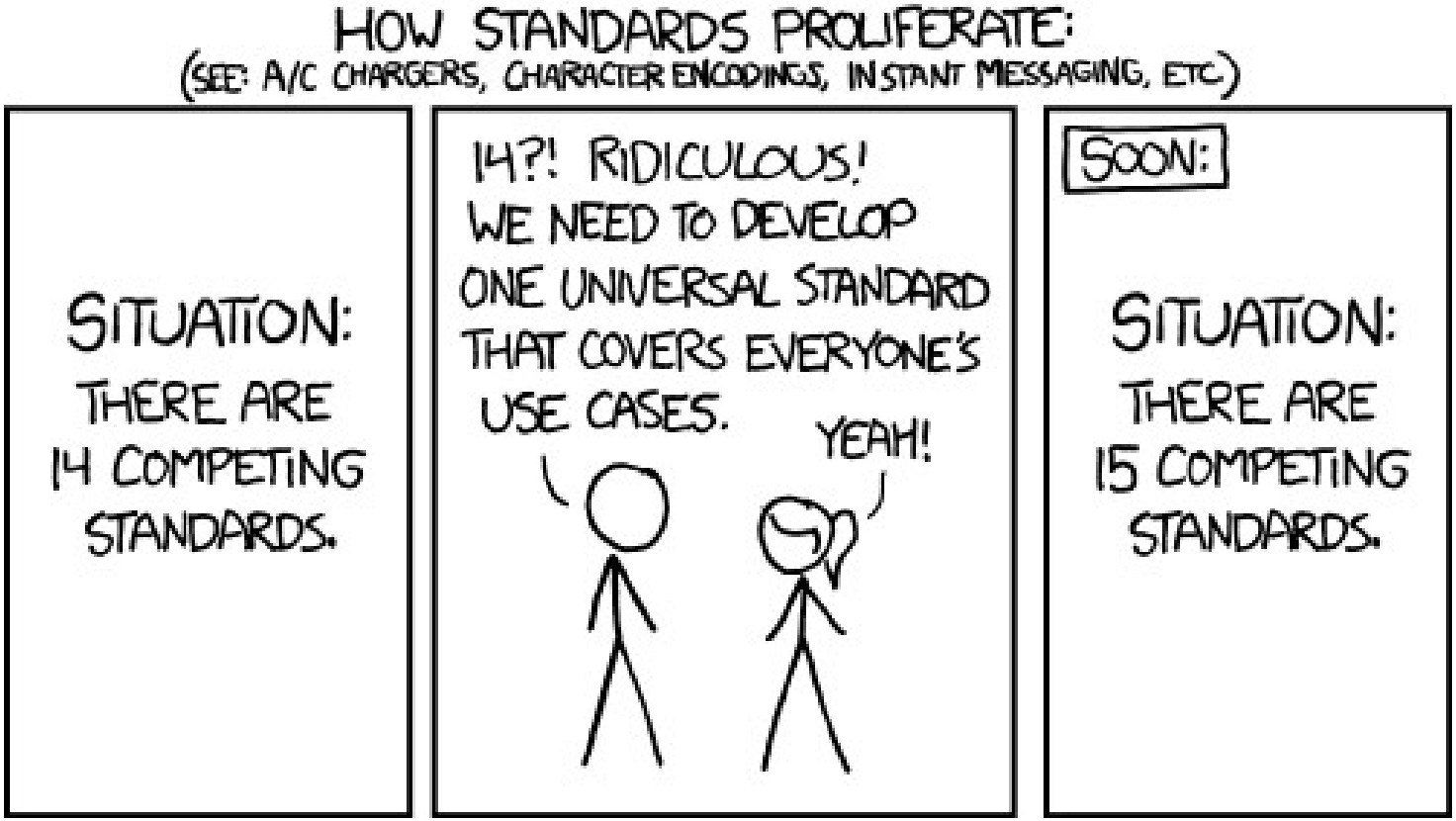
Web Services

Standards…



Credits: https://xkcd.com/927/

Primeiro objetivo:
Serviços e Aplicações de grande escala na Internet

Alinhar com a Internet

- Usar de forma direta o HTTP e HTTPS como protocolos de transferência de informação
- Usar URL e URI como referências remotas para objectos

Total independência das tecnologias proprietárias Java-Sun, Microsoft, IBM

Motivação dos Web Services (I)

- Protocolo simples para garantir a interoperação entre plataformas de múltiplos fabricantes
- Tratar todo o tipo de heterogeneidade de dados e informação com XML
- Permitir utilizar RPC ou MOM (*Message Oriented Middleware*)
  - Sistemas de comunicação síncronos e assíncronos

Motivação dos Web Services (II)

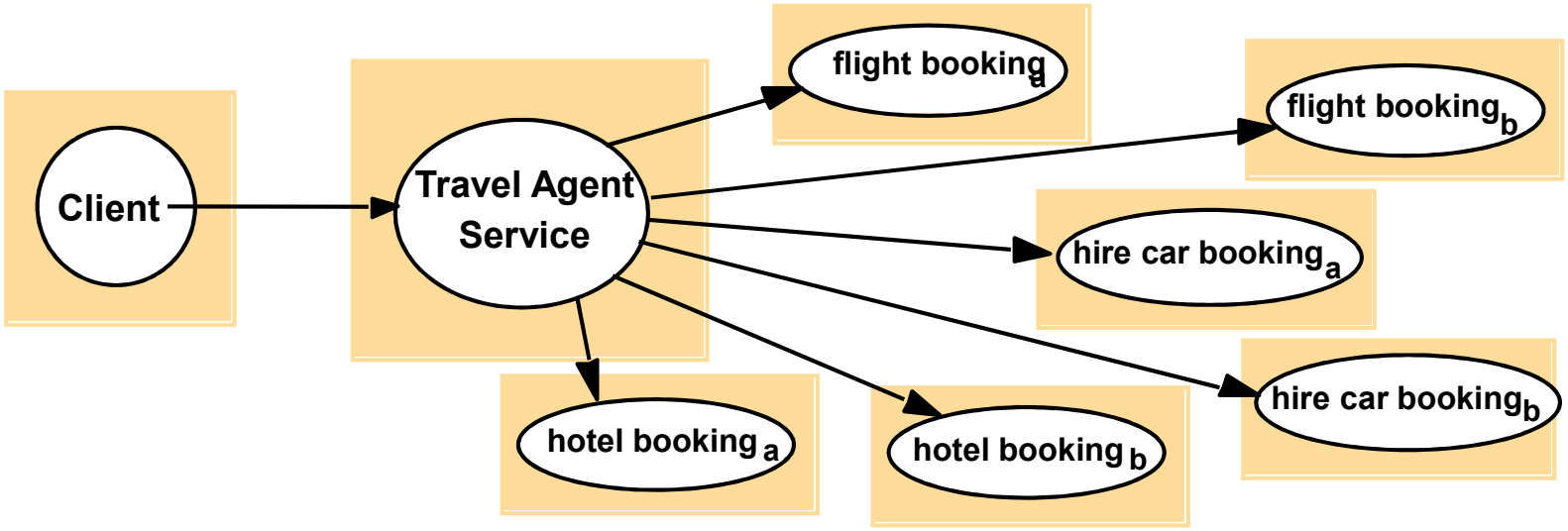
- Usar de forma direta o HTTP e HTTPS como protocolos de transferência de informação
  - Passar através das *Firewalls*, que bloqueiam dados binários, usar *well-known ports*
- Usar URL e URI como referências remotas para objectos
- Permitir a transferência de todo o tipo de informação
  - Desde estruturas de dados a documentos estruturados e informação multimédia
- Eliminar a distinção de sistemas para transferência de documentos (mensagens de texto, faturas, …) e sistemas para transferência de dados

Exemplo:
Amazon Store Web Services

- Permitem programar aplicações que usam serviços da Amazon através de Web Services
  - Vários serviços disponíveis para execução em *cluster* de grande escala
- Exemplo de serviço:
  controlo de inventário e
  vendas para vendedores

  - Existe API que permite a clientes remotos:
    - Registar fornecimentos de produtos para armazéns da Amazon
    - Verificar estado dos fornecimentos
    - Consultar estado de vendas dos produtos fornecidos

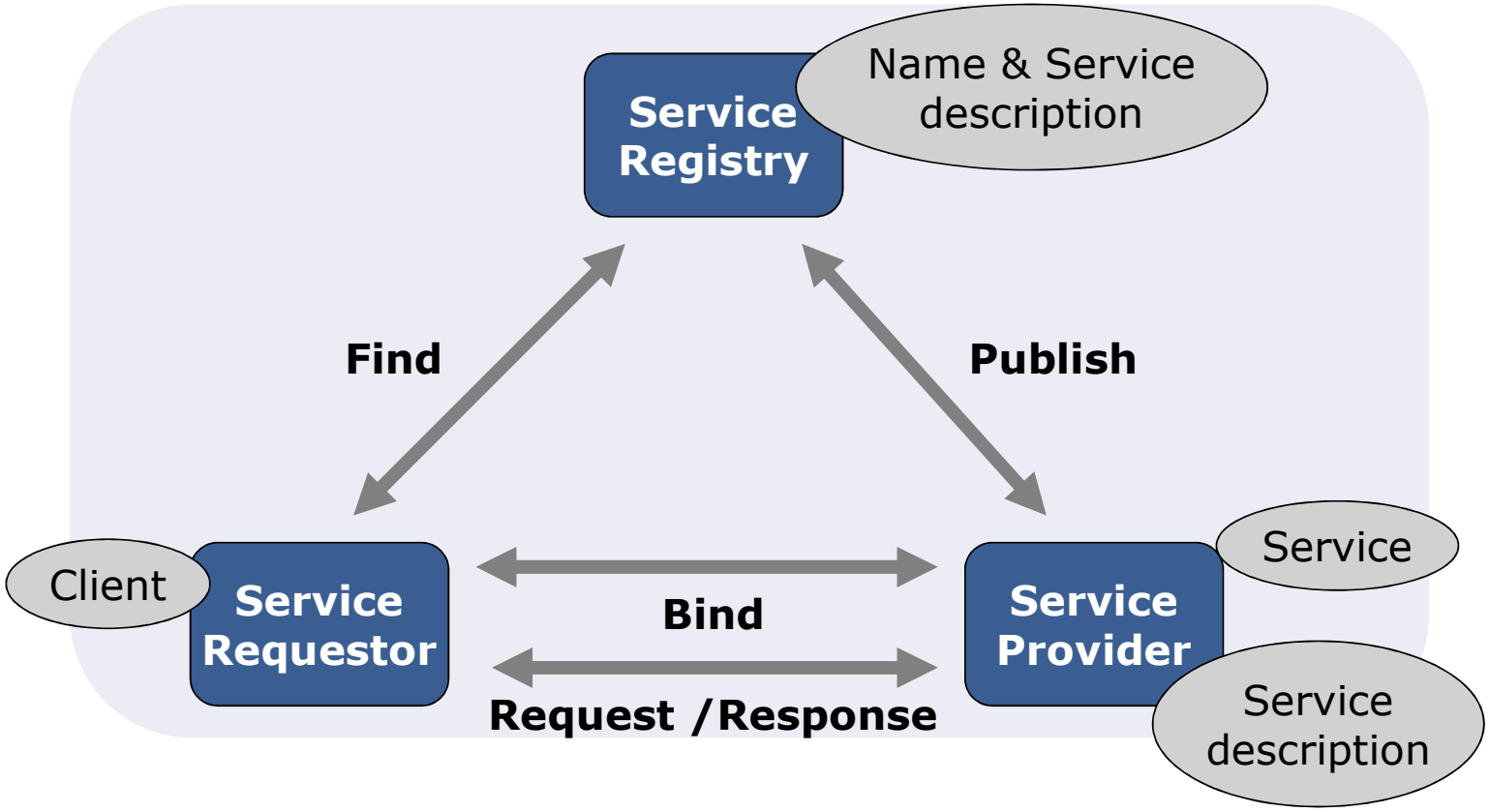
Exemplo: Composição de Web Services



Evolução

- 1997
  - A Sun distribui o JDK 1.1 que inclui o Remote Method Invocation (RMI) que define um modelo de computação distribuída usando objetos Java.
    O RMI é semelhante ao CORBA e ao DCOM mas funciona só com objetos Java
  - Microsoft desenvolveu o COM+ sucessor do DCOM muito próximo do modelo CORBA
- 1999
  - A SUN distribui o J2EE (Java 2 Platform Enterprise Edition) que integra o RMI e o IIOP tornando mais simples a interoperação de sistemas entre sistemas Java e CORBA.
  - O SOAP (*Simple Object Access Protocol*) apareceu pela primeira vez.
- 2001
  - A IBM e a Microsoft propõem as pilhas de protocolos dos Web Services à W3C (*World Wide Web Consortium*)
    - *Wire stack*
    - *Description stack*
    - *Discovery stack*

Modelo dos Web Services
(arquitetura básica)

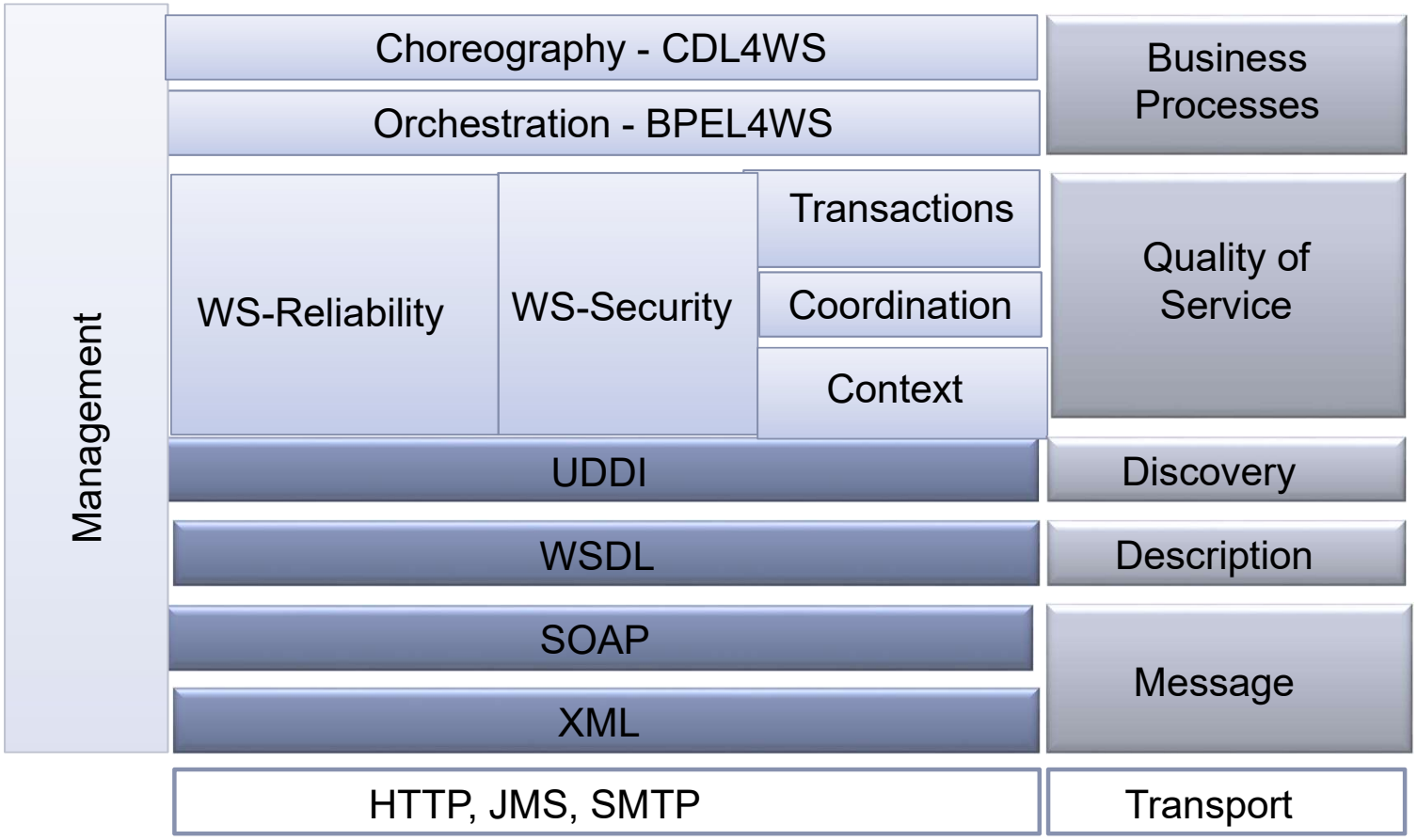


Arquitetura dos Web Services

- Um serviço de Diretório para registo e pesquisa dos serviços:
  UDDI (ou outros)
- Um protocolo de pedido/resposta para invocação do serviço:
  SOAP (ou outros)
- Uma especificação da interface do serviço
  WSDL

- Páginas amarelas e diretório
  (endereço, contactos, identificadores, categorizadores, ...)
- Interação
- Contratos

Web Services standards



A base de desenvolvimento dos Web Services

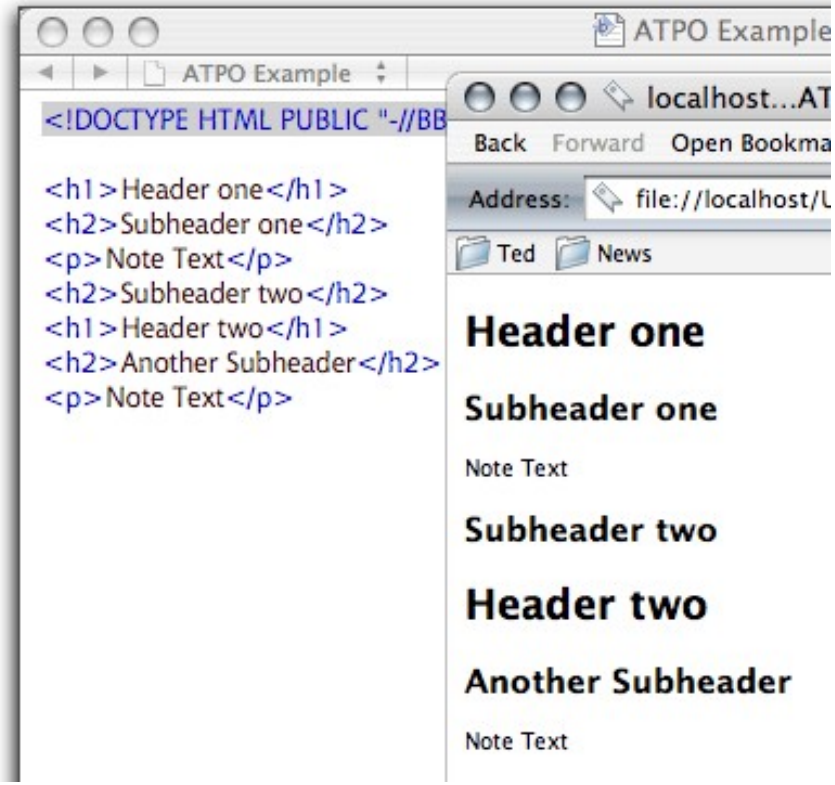
World Wide Web (WWW)

- Sistema para publicação e acesso a recursos e serviços na Internet
- Iniciada no CERN (Centro Europeu de Investigação Nuclear), Suíça/França) em 1989
  - Objetivo: partilhar documentos entre os cientistas do CERN, ligados pela Internet

Componentes fundamentais

- HyperText Markup Language (HTML)
  - Linguagem para especificar conteúdos e apresentação das páginas apresentadas nos browsers
- Uniform Resource Locator (URLs)
  - Identificam documentos e outros recursos da WWW
- HyperText Transfer Protocol (HTTP)
  - Protocolo de interação cliente-servidor baseado em TCP/IP

HTML



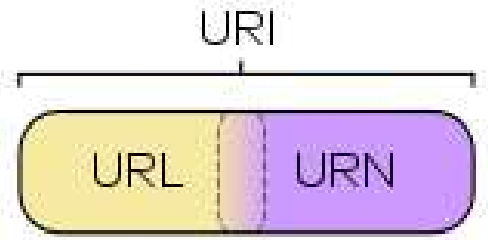
Essencialmente orientado a apresentação da informação e não à descrição dos tipos de dados

Pouco útil na evolução para os Web Services

Uniform Resource Identifiers (URIs)

- Standard de nomes de recursos na WWW
- Sintaxe: [prefixo]:[sufixo-específico-do-protocolo]
  - Exemplos:
    urn:isbn:0-486-27557-4
    https://www.tecnico.ulisboa.pt/index.html
    mailto:info@tecnico.ulisboa.pt
- Duas funções distintas:
  - **Identificar** univocamente um recurso na Internet
    - Chamados URNs (Uniform Resource Names)

  e/ou
  - **Localizar** um recurso na Internet
    - Chamados URLs (Uniform Resource Locator)



HyperText Transfer Protocol - HTTP

- Foi o protocolo de base da *World Wide Web* definido em 1990
- Um cliente Web comunica com um servidor Web usando uma ou várias ligações TCP
- Um porto normalmente predefinido para o servidor Web é o porto 80
- O protocolo é muito simples:
  - O cliente estabelece uma ligação TCP com o servidor
  - Envia um pedido
  - Lê a resposta
  - O servidor fecha a ligação

Pedido / Resposta
Mas é um RPC ?

- Nas versões posteriores existem *persistent connections* que permanecem estabelecidas durante uma interacção

HyperText Transfer Protocol - HTTP

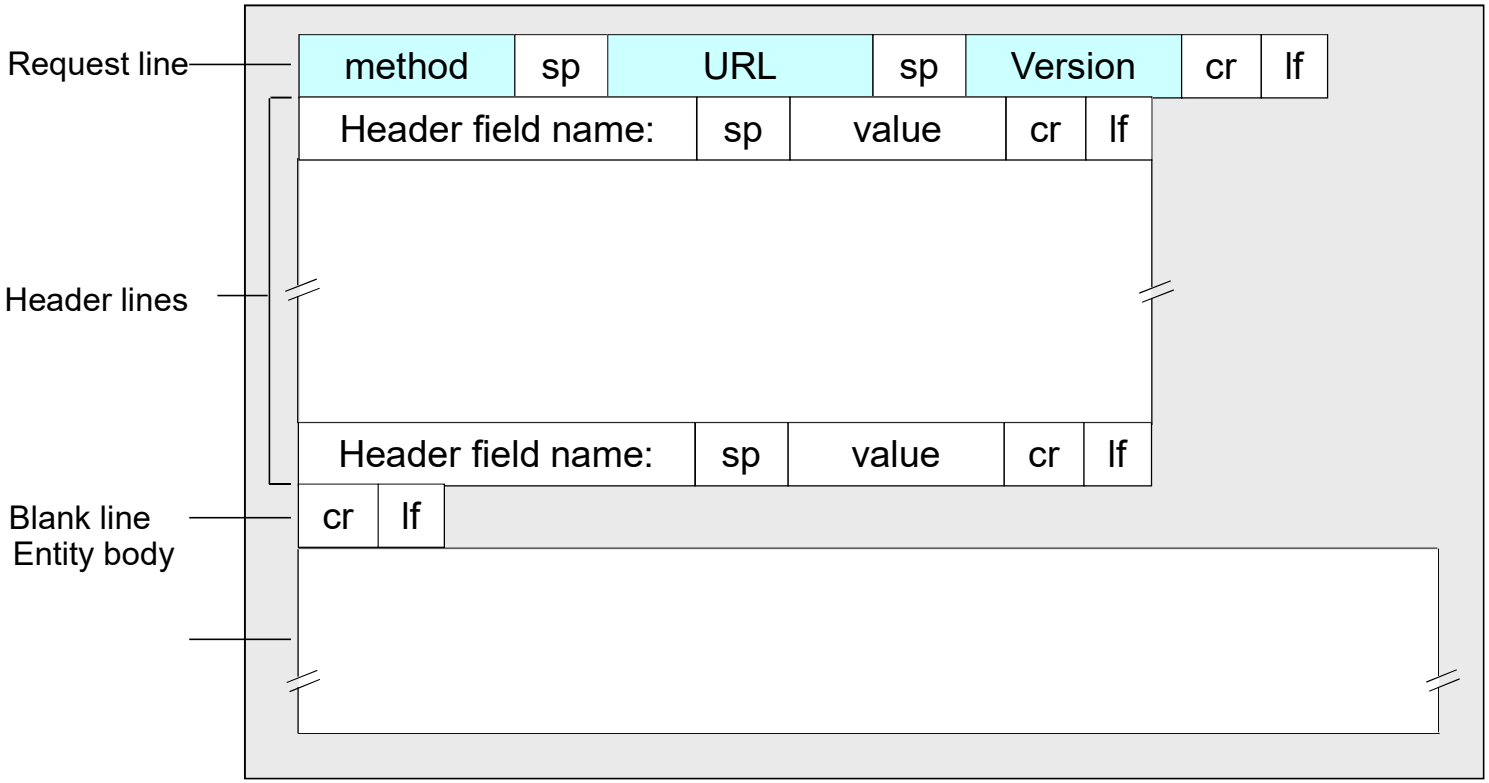
- Protocolo de Pedido-Resposta do tipo RPC
- Diferença: funções remotas estão predefinidas:
  - GET, PUT, POST, etc.
- O protocolo permite parametrizar
  - Conteúdos – os pedidos dos clientes podem especificar que tipo de dados aceitam
  - Autenticação – credenciais e desafios são utilizados para uma autenticação do tipo *password*

Método	Descrição
GET	Pedido de documento
HEAD	Pedido apenas de cabeçalho de documento
PUT	Pedido para guardar um documento
POST	Fornecimento de informação para ser acrescentada ao documento
DELETE	Pedido para apagar um documento

Mensagem de Pedido

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept-language: fr
```

Formato genérico da mensagem de pedido

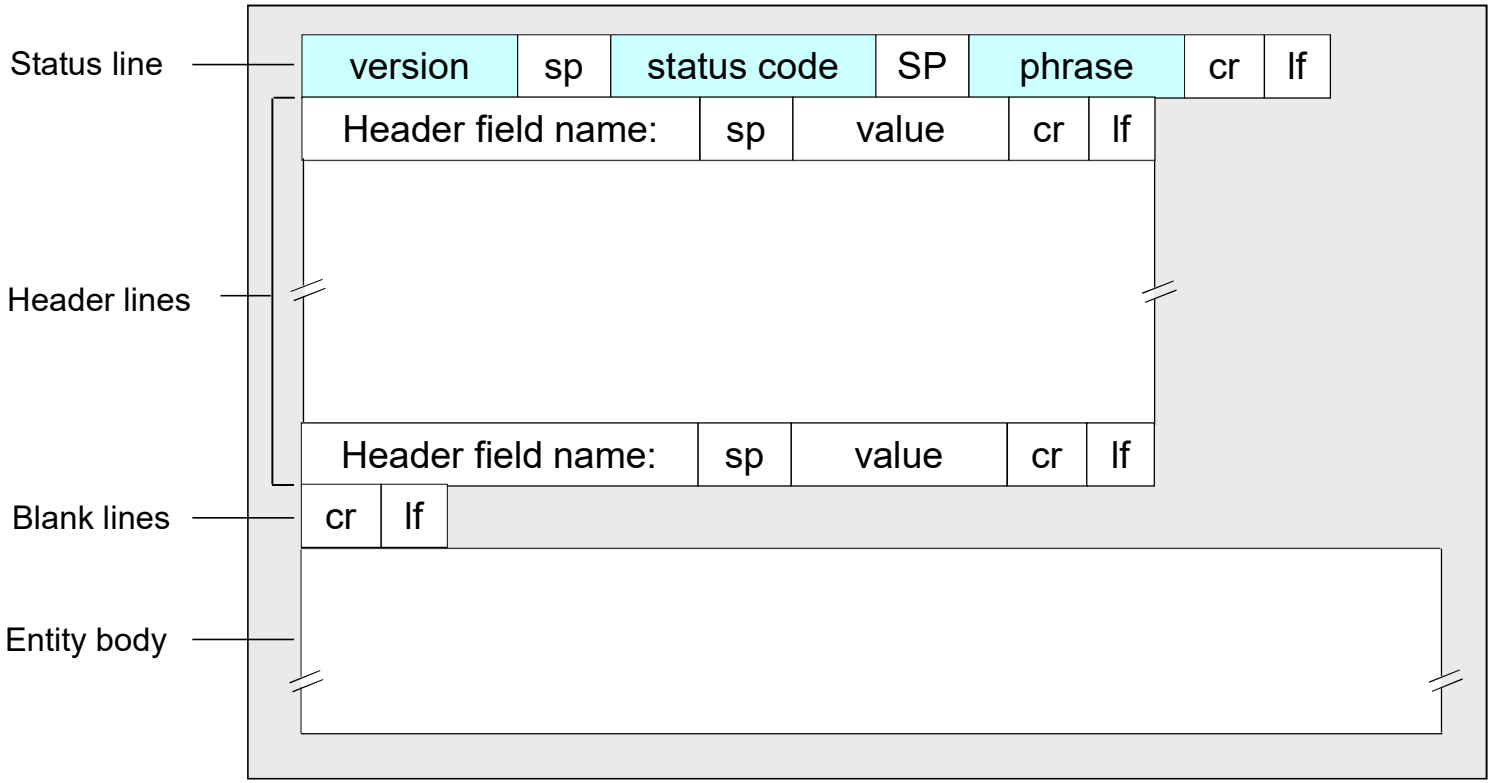


Mensagem de Resposta

HTTP Response Message

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 03 Jul 2013 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Sun, 5 May 2013 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html

data ...
```

Formato genérico da mensagem de resposta



HTTP

- Portabilidade:
  - Os pedidos e respostas são transformados em cadeias de caracteres, eliminando o problema da heterogeneidade
    mas tornando as mensagens mais longas
  - Contudo não permite distinguir tipos de dados
    ex. um cadeia de caracteres com algarismos pode ser uma *string*, um *int* ou um número real
- A semântica é no-máximo-uma-vez
  - Ligação TCP/IP sem repetições

Importante

- HTTP é um protocolo de pedido-resposta
- Mas não é um RPC porque apenas pode executar as operações predefinidas
  - Não tem mecanismos para disponibilizar serviços genéricos
- Não resolve o problema geral de heterogeneidade dos dados nas mensagens

Páginas Dinâmicas na WWW

Páginas dinâmicas

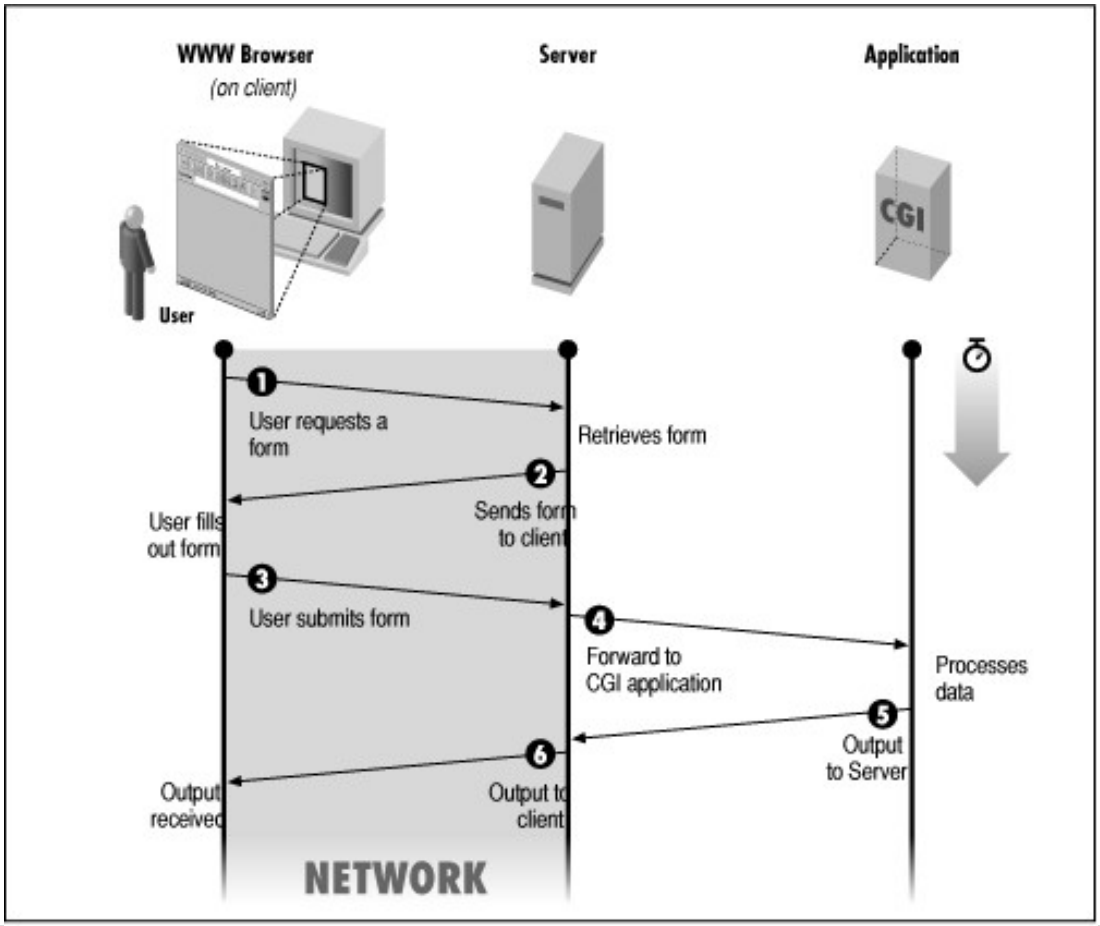
- Serviços interativos são difíceis de implementar com páginas estáticas
- Solução: gerar dinamicamente as páginas apresentadas

Páginas dinâmicas: Abordagem do lado do servidor

- URL não indica página estática mas sim um programa
  - Quando chega um pedido, o servidor Web executa o programa com os parâmetros fornecidos no pedido
  - Programa pode manter estado e aceder a base de dados no servidor
  - Programa gera documento HTML que é enviado ao cliente
  - Exemplos: CGI, Java Servlets, etc.
- Programa pode correr fora ou dentro do servidor
  - No segundo caso, chama-se *Servidor Aplicacional*

Mecanismo usado para invocar um serviço remoto

Páginas dinâmicas: Abordagem do lado do servidor



- Desvantagens?

Páginas dinâmicas: Abordagem do lado do cliente

- URL referencia uma página estática
- A página inclui código para o *browser* que se executará no cliente
  - Para além do conteúdo estático HTML
- Exemplos: JavaScript

Páginas dinâmicas: Abordagem do lado do cliente

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>Form Validation Example</title>
    <script>
    <!--
        function check_myForm()
        {
            var name = new String(document.myForm.name.value);
            var phone = new String(document.myForm.phone.value);
            var age = new String(document.myForm.age.value);

            if(name.length < 3 || name.indexOf(' ') == -1)
            {   alert("Invalid Name"); return false; }

            if( (phone.length != 12) || (phone.charAt(3) != '-') || (phone.charAt(7) != '-') )
            {   alert("Invalid Phone Number"); return false; }

            if(age > 65 || age < 21)
            {   alert("Invalid Age, Must be between 21-65"); return false; }
            alert("Form Data Validated, no problems found");
            JavaScript:document.myForm.submit();
        }
    //-->
    </script>
</head>
<body>
<form name="myForm" action="http://www.profiler.noaa.gov/jsp/FormTester.jsp">
    Enter Your Name:<input type="text" name="name" size="40"> (First and Last)<br>
    Phone Number:<input type="text" name="phone" size="14"> (###-###-####)<br>
    Age:<input type="text" name="age" size="4"> (between 21 - 65)<br>
    <input type="button" value="Validate Form Data" onClick="check_myForm()">
</form>
</body>
</html>
```

- Limitações?
- Vantagens?

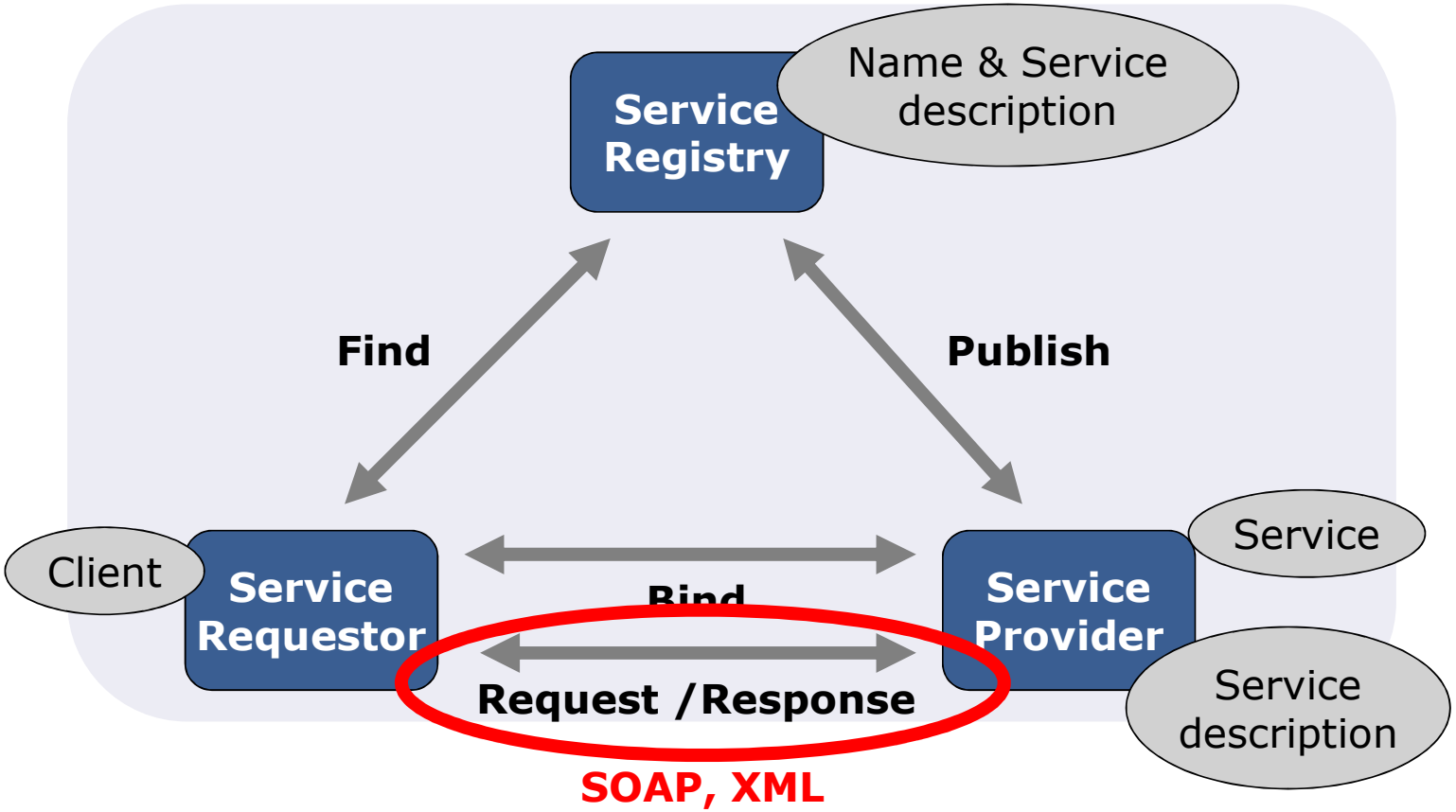
Páginas dinâmicas: Abordagem mista

- Combina ambas as abordagens
  - Permite distinguir entre código a executar no cliente e no servidor
- Exemplos: JSP, ASP, ASP.Net

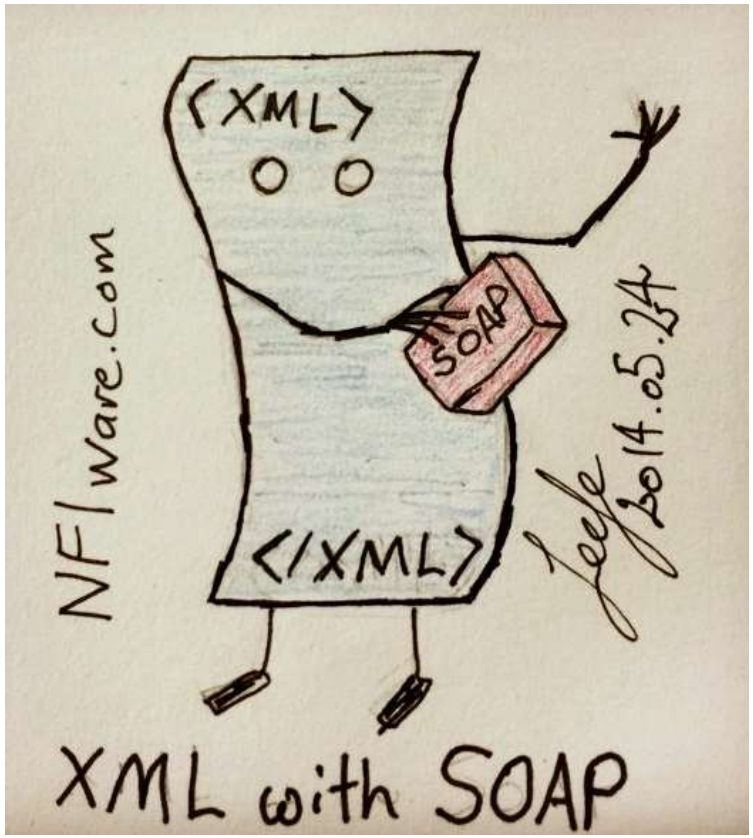
Exemplo simples – mensagem
Com definição de namespace

```
<mensagem xmlns="urn:empresa.pt:mensagem"
   id="74536">
     <!-- isto é um comentário -->
     <de>João</de>
     <para>Carla</para>
     <assunto>Reunião</assunto>
     <texto>Confirmo a reunião dia 6</texto>
</mensagem>
```

Modelo dos Web Services
(arquitetura básica)



SOAP ?



CREDITS: http://www.nfiware.com/2014/05/24/xml-with-soap-keeping-web-services-clean-cartoon/

SOAP

Protocolo de comunicação dos Web Services

Simple Object Access Protocol - SOAP

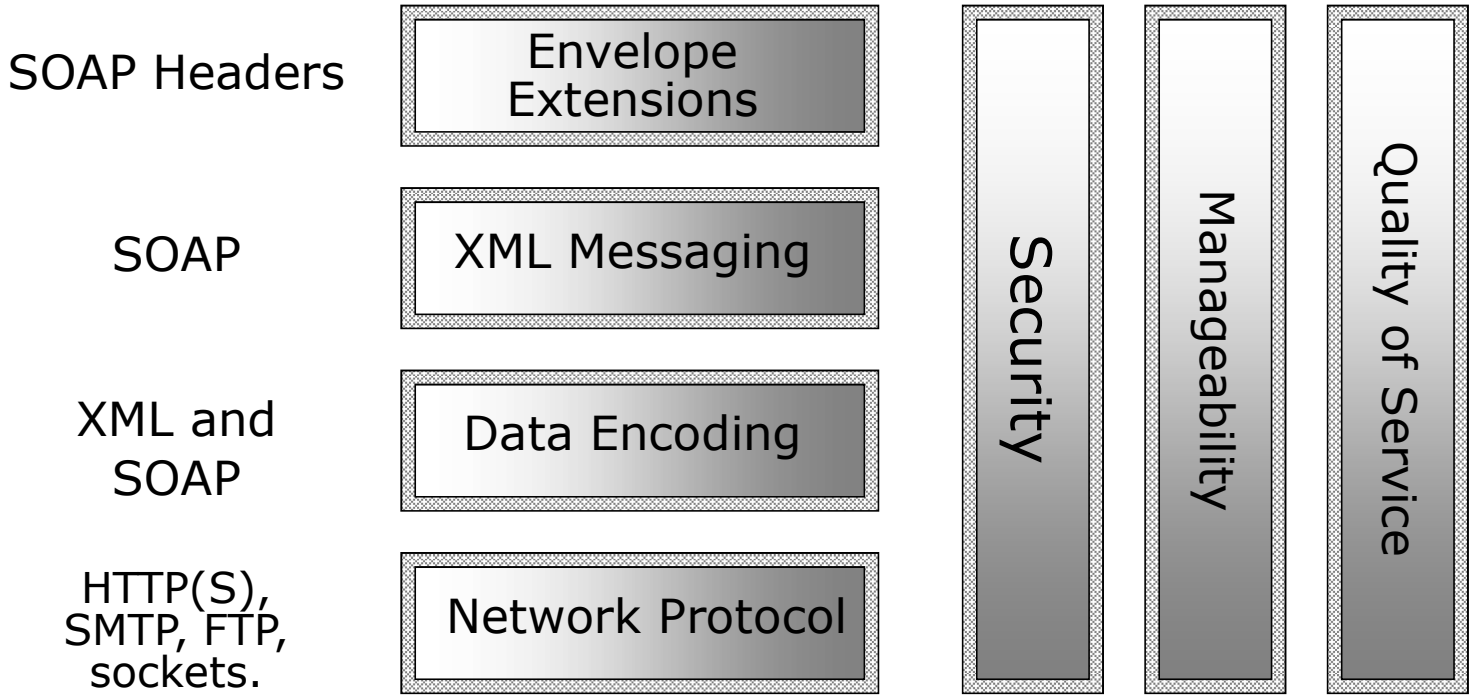
Objetivo

- ***Formato universal para mensagens XML***

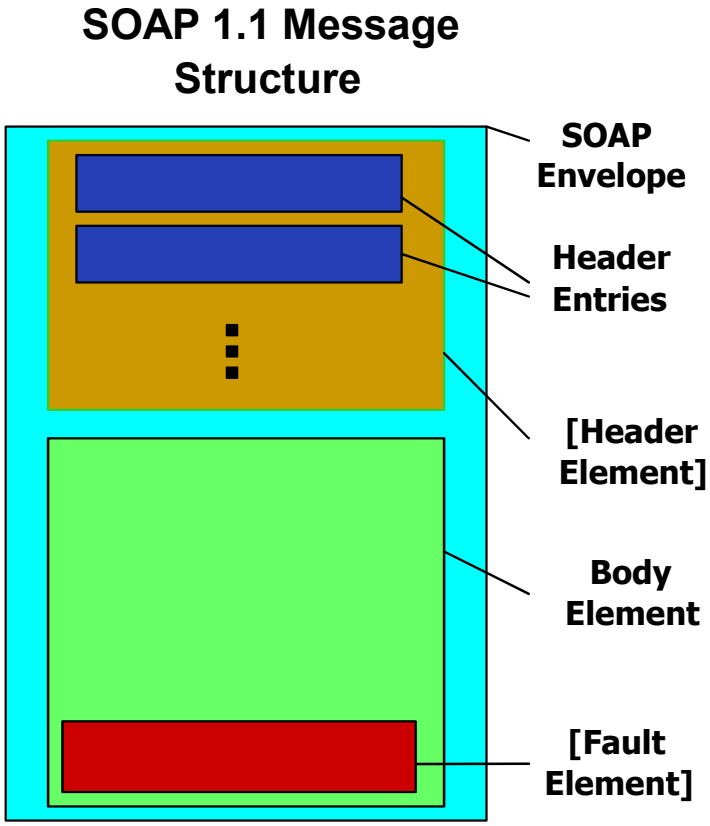
Características

- Protocolo de comunicação distribuído permitindo o envio de **qualquer tipo de informação** entre aplicações
- Define o protocolo de **pedido-resposta:** estrutura das mensagens e da interação entre cliente e servidor
- Protocolo de representação de dados baseado em **XML**
- Referências remotas baseadas em **URI**
- **Protocolo extensível** permitindo a incorporação de várias facetas: segurança, tolerância a faltas, através de cabeçalhos associados às mensagens

Wire stack – Visão dos Web Services

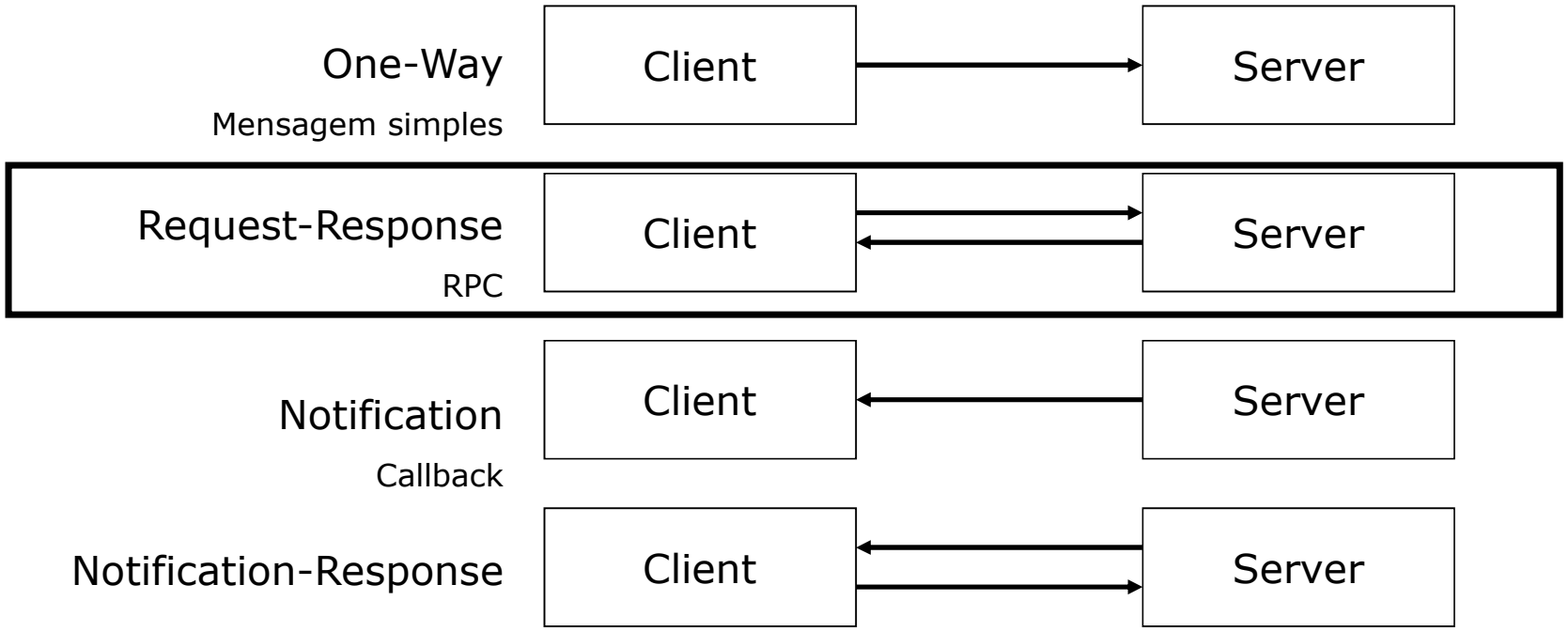


SOAP Simple Object Access Protocol



- Define:
  - Modelo de empacotamento: SOAP Envelope
- Baseado em XML
- Pode ser usado em praticamente todos os protocolos de transporte:
  - HTTP
  - SMTP
  - TCP/IP
  - JMS (Java Message Service)
  - ....

Interações previstas no SOAP



Execução simples em SOAP

- É necessário um URL de destino
- O nome de uma operação
- Os parâmetros:
  - Os parâmetros são passados por cópia (*in* e *out*)
  - Não existem referências para os objetos remotos
    - Criadas automaticamente no Java RMI
- Informação contextual, como a informação de segurança

Pedido SOAP

***Binding* de SOAP sobre HTTP**

O que há de específico deste transporte?

Envelope SOAP com pedido

```
POST /hello-ws/endpoint HTTP/1.1
Host: www.server.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 322
SOAPAction: ""

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ns1="http://hello">

    <soapenv:Body>
       <ns1:sayHello>
          <ns1:name>friend</ns1:name>
       </ns1:sayHello>
    </soapenv:Body>

</soapenv:Envelope>
```

Resposta SOAP

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: 367

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:ns1="http://hello">

   <soapenv:Body>
      <ns1:sayHelloResponse>
         <ns1:return>Hello friend!</ns1:return>
      </ns1:sayHelloResponse>
   </soapenv:Body>

</soapenv:Envelope>
```

Exemplo

- Servidor que disponibiliza o último preço praticado para um produto
- Função Remota

```
float GetLastTradePrice(string symbol)
```

Pedido SOAP

POST /StockQuote HTTP/1.1
Host: **www.stockquoteserver.com**
Content-Type: **text/xml; charset="utf-8**"
Content-Length: nnnn
SOAPAction: "Some-URI"

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
     <SOAP-ENV:Body>
            <m:GetLastTradePrice xmlns:m="Some-URI">
                    <symbol>DIS</symbol>
            </m:GetLastTradePrice>
     </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Resposta SOAP

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

<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    <SOAP-ENV:Body>
            <m:GetLastTradePriceResponse xmlns:m="Some-URI">
                <Price>34.5</Price>
            </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Erro SOAP

HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset=”utf-8”
Content-Length: nnn

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=”http//schemas.xmlsoap.org/soap/envelope/”
  SOAP-
    ENV:encodingStyle=”http//schemas.xmlsoap.org/soap/encoding/”>
  <SOAP-ENV:Body>
      <SOAP-ENV:Fault>
         <faultcode>Client.AuthenticationFailure</faultcode>
         <faultstring>Failed to authenticate client</faultstring>
         <faultactor>urn:X-SkatesTown:PartnerGateway</faultactor>
      </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Binding do SOAP ao protocolo de Transporte

- O HTTP é um protocolo de pedido-resposta,
  o que torna o protocolo de RPC do SOAP muito simples
  - Para outros protocolos tem de se criar um protocolo de controlo da invocação remota
- O HTTP permite que o servidor RPC não tenha estado
- A confidencialidade da informação pode ser assegurada pelo HTTP/S

Physical (Communication Protocol) Message

```
POST /LookupCentral HTTP/1.1
Host: www.lookupcentraserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnn
SOAPAction: "Directory/LookupPerson"
```

Out-of-message context (target URI)

Out-of-message context (SOAPAction)

Logical SOAP Message

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle= "http://schemas.xmlsoap.org/soap/encoding/"/>

   <SOAP-ENV:Header>
      <a: AuthorizationLevel>
            xmls:a="some-URI">
      </a:AuthorizationLevel>
   </SOAP-ENV:Header>

   <SOAP-ENV:Body>
         <m:LookupPerson xmlns:m="Some-URI">
               <FirstName>Big<FirstName>
               <LastName>Boss</LastName>
         </m:LookupPerson>
   </SOAP-ENV:body>

</SOAP-ENV:Envelope>
```

SOAP Headers

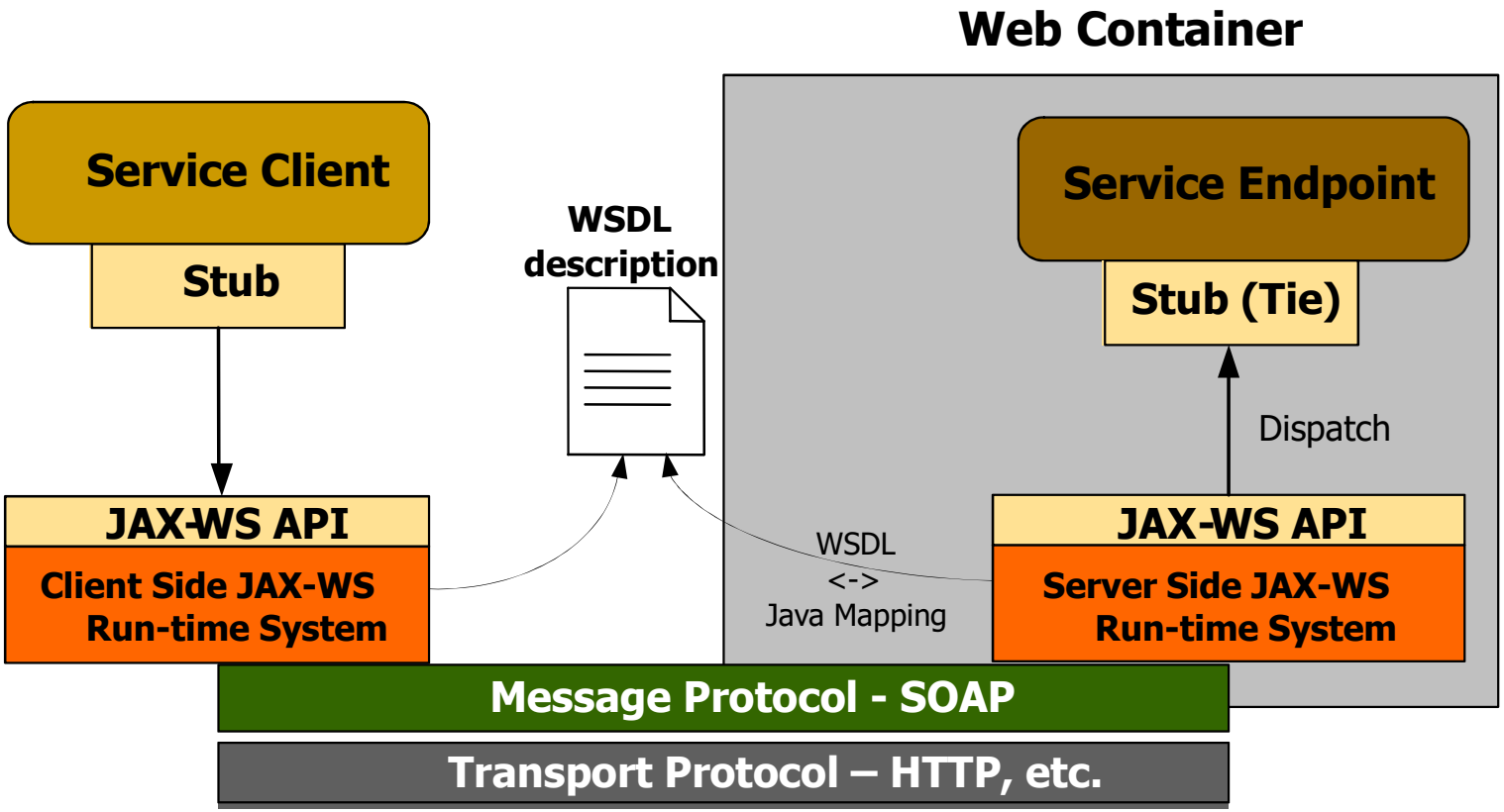
In-message context

SOAP Body

JAX-WS
(sucessor do JAX-RPC)

Integração dos Web Services com o ambiente Java

JAX-WS: Arquitetura



Quando usar cada abordagem?

Implementation-First

- Oferecer por Web Service operações já implementadas
- Construir novo Web Service de raiz sem ter de aprender WSDL

Contract-First

- Programador conhecedor de WSDL que pretende ter elevado controlo sobre o que é especificado no contrato
- Substituir a implementação de um Web Service existente por outro mantendo compatibilidade com clientes
- Aderir a contrato definido por terceiros (e.g. protocolo definido por consórcio de organizações)

Exemplo Implementation-First interface do Web Service

```
@WebService
public interface Hello {
        @WebMethod
        String sayHello(String name);
}
```

Não herda de Remote!
Tem uma anotação para indicar que é um Web Service

Annotations, **a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.**
Annotations have a number of uses, among them:

- **Information for the compiler — Annotations can be used by the compiler to detect errors or suppress warnings.**
- **Compile-time and deployment-time processing — Software tools can process annotation information to generate code, XML files, and so forth.**
- **Runtime processing — Some annotations are available to be examined at runtime.**

Exemplo Implementation-First
Implementação do Web Service

```
@WebService(endpointInterface="example.ws.Hello")
@WebServlet("/endpoint")
public class HelloImpl implements Hello {
    public String sayHello(String name) {
        return "Hello" + name + "!";
    }
}
```

Implementation-First: Java com Anotações

```
@WebService(targetNamespace = "http://duke.org", name="AddNumbers")
@SOAPBinding(style=SOAPBinding.Style.RPC, use=SOAPBinding.Use.LITERAL)
public interface AddNumbersIF extends Remote {
    @WebMethod(operationName="add", action="urn:addNumbers")
    @WebResult(name="return")
    public int addNumbers(
        @WebParam(name="num1")int number1,
        @WebParam(name="num2")int number2) throws AddNumbersException;
}
```

Cliente

- O cliente pode ter um *static proxy* criado antes da execução (*static stub*)
  - Compilado a partir do WSDL pelo `wsimport`
- O cliente pode também usar um *dynamic proxy*
  - Classe que é criada durante a execução a partir do WSDL

Cliente com Stub estático: Exemplo

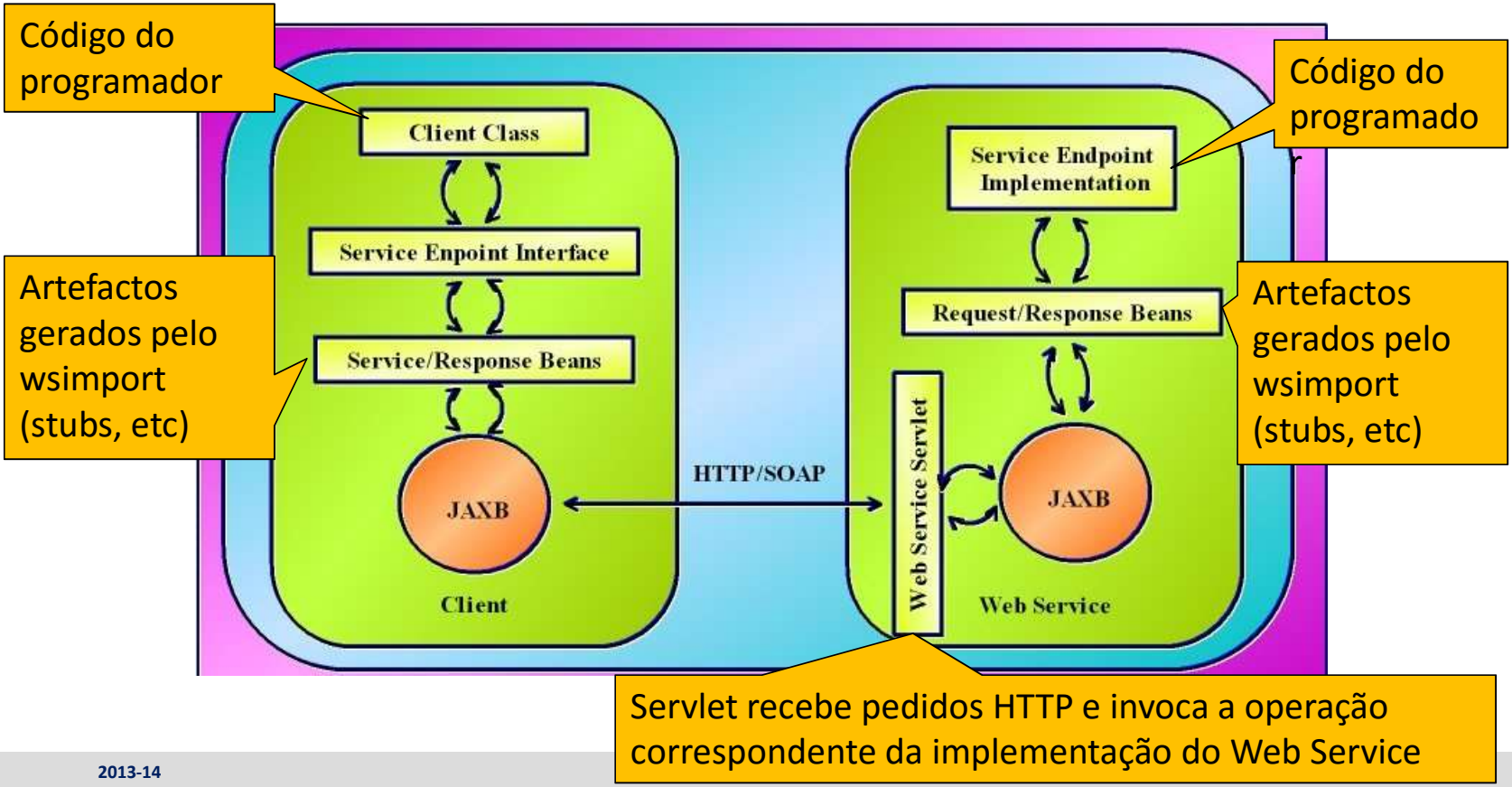
```
public static void main(String[] args) throws Exception {

    HelloWorld_Impl proxy = new HelloWorld_Impl();
    HelloWorldSoap soapProxy = proxy.getHelloWorldSoap();
    System.out.println(soapProxy.sayHello());

}
```

Stub, gerado previamente com ferramenta *wsimport*

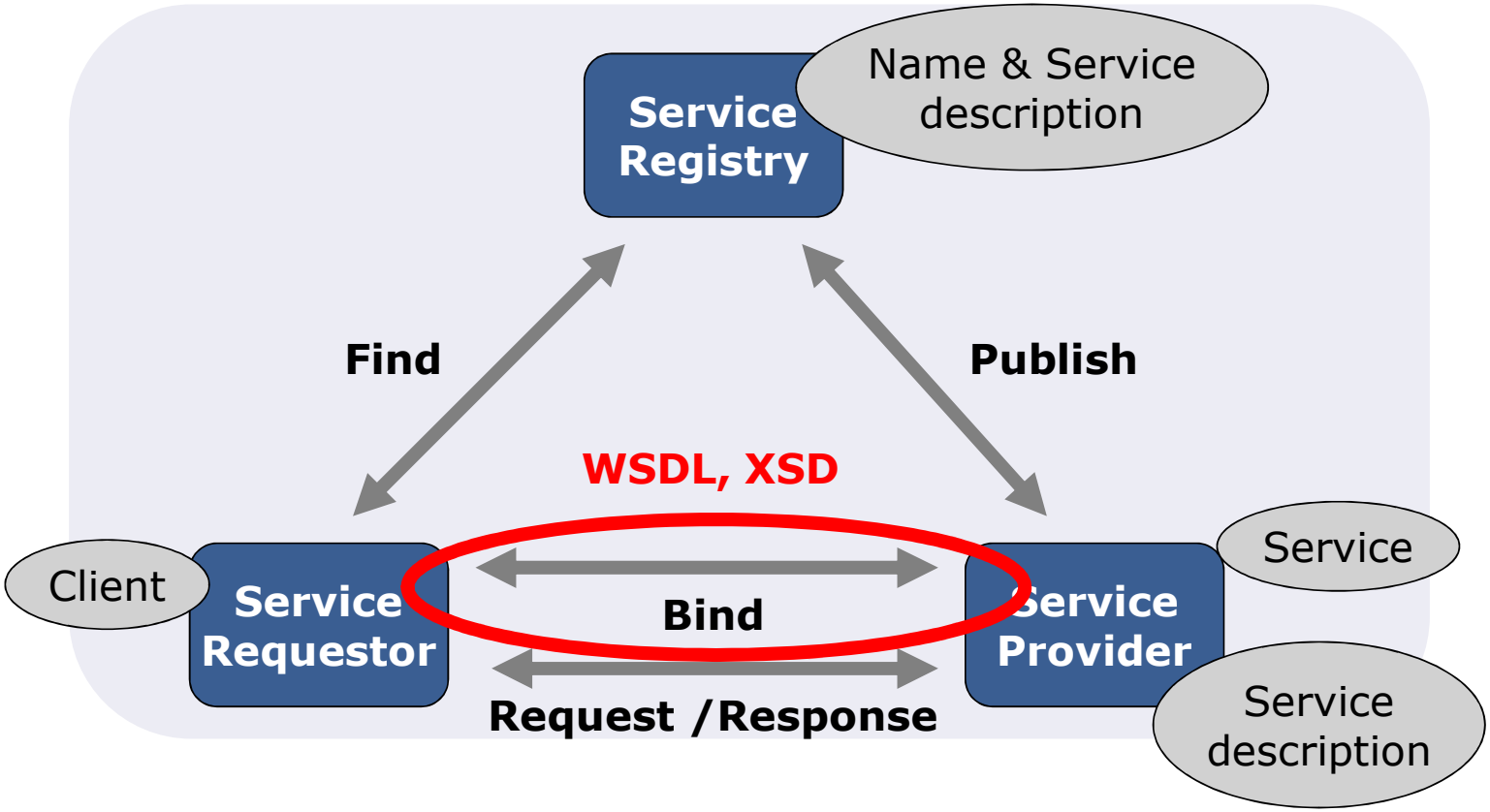
Cria uma instância da classe *proxy*

Obtém o *proxy* especifico para Soap

JAX-WS: Arquitectura



Modelo dos Web Services
(arquitetura básica)



WSDL - Web Service Definition Language

Definição do contrato do Serviço

WSDL - Web Service Definition Language

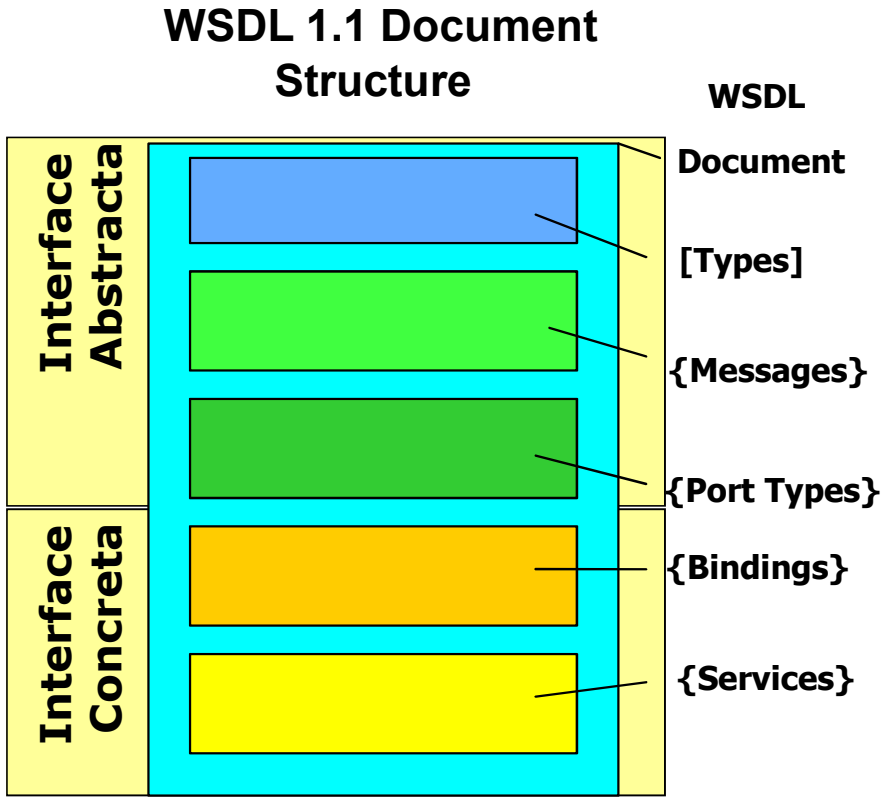
- A IDL para descrição dos contratos em Web Services
- Define o contrato a que o serviço se obriga

- A definição permite descrever
  - Qual o serviço
  - Que mensagens devem ser enviadas e qual a sua estrutura
  - Como usar os vários protocolos de transporte
  - Onde o serviço está localizado, mais precisamente para que rede a mensagem deve ser enviada

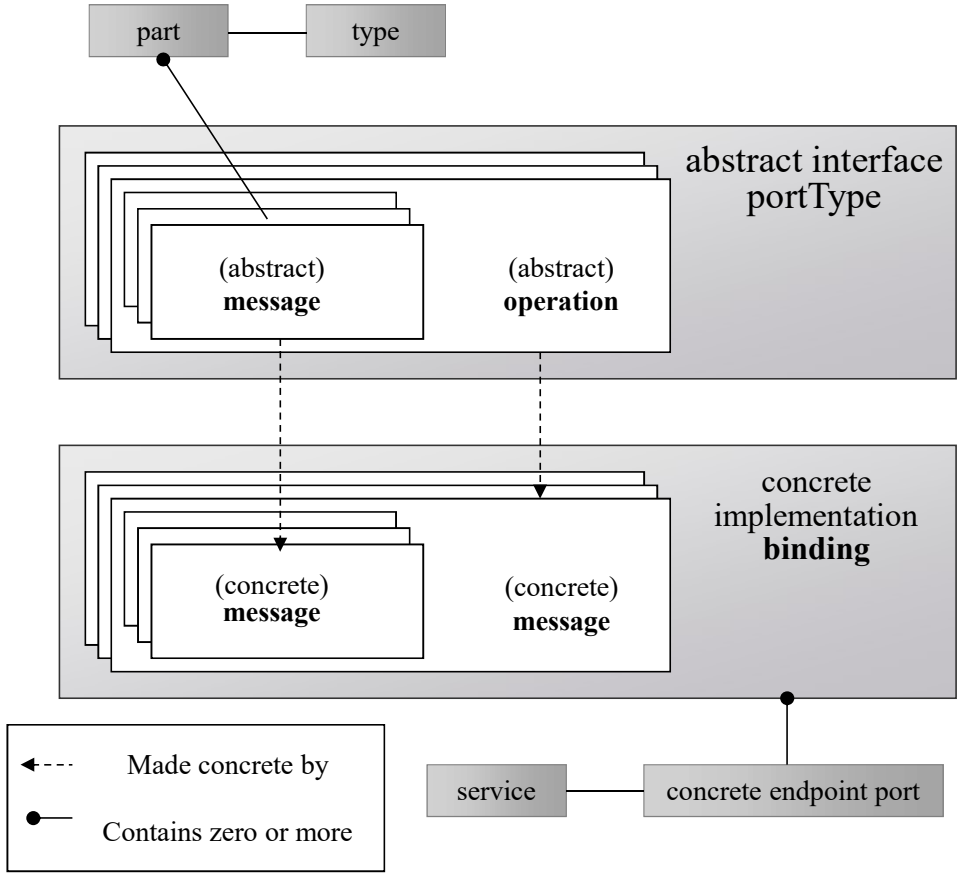
Diferenças vs IDL de RPC?

WSDL

Web Services Description Language

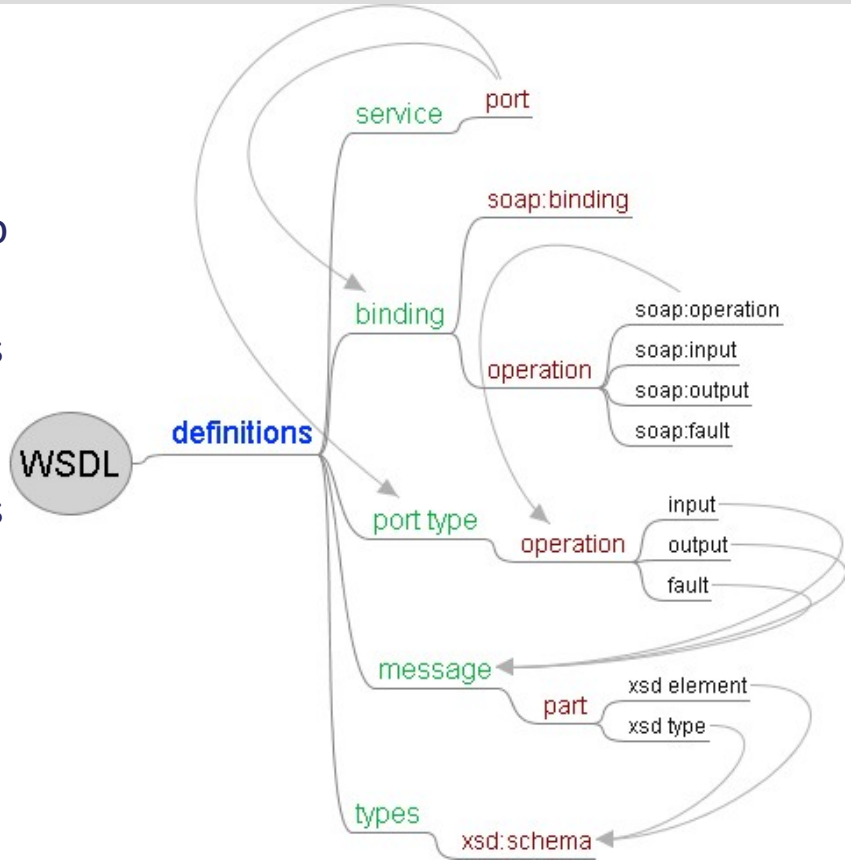


WSDL information model



WSDL
Web Services Description Language

- O contrato WSDL indica quais as operações disponibilizadas pelo Web Service aos seus clientes
- O conjunto das operações é designado po interface
- Para cada operação, especifica os argumentos (*inputs*), os resultados (*outputs*) e os erros (*faults*).
- Os tipos de dados dos argumentos, resultados erros são descritos com esquemas XSD
- O contrato WSDL não é fácil de ler
  - É extenso
  - A sua organização está invertida
  - Tem múltiplas dependências



Definições

- *Port Type* – Descreve a interface abstracta de um Web service.
  - Atenção porque o termo "port" é usado com um sentido totalmente diferente dos sockets. Um *port type* de um Web Service é semelhante a uma interface Java
- *Message* – assinatura das operações descrevendo o nome e os parâmetros da operação
- *Types* – coleção de todos os tipos de dados usados na especificação.
- Estes elementos são reutilizáveis porque definem entidades abstratas e não a concretização de um serviço

portType

```
<portType name="PriceCheckPortType">
        <operation name="checkPrice">
            <input message="pc:PriceCheckRequest"/>
            <output message="pc:PriceCheckResponse"/>
        </operation>
</portType>
```

- Descreve o que o Serviço faz
- As mensagens permitem saber a assinatura dos métodos
- Normalmente um documento WSDL contem apenas um port type por razões de reutilização

Mensagens

```
<! - - Message definitions - ->
<! - - A PriceCheckRequest is simply an item code (sku)
<message name="PriceCheckRequest">
    <part name="sku" type="xsd:string"/>
</message>

<! - - A PriceCheckResponse consists of an availability structure
<! - - defined above.
<message name="PriceCheckResponse">
    <part name="result" type="avail:availabilityType"/>
</message>
```

- As mensagens podem ser de **input**, **output** ou assinalar **faltas**
- Podem ser reutilizadas em diferentes operações

Tipos de dados

```
<types>= http://www.skatestown.com/ns/availability
   <xsd:schema targetNamespace
   xmlns:xsd = http://www.w3.org/2001/XMLSchema>
      <xsd:complexType name="availabilityType">
        <xsd:sequence>
           <xsd:element name="sku" type="xsd:string"/>
           <xsd:element name="price" type="xsd:double"/>
        </xsd:sequence>
      </xsd:complexType>
   </xsd:schema>
</types>
```

- Tipos utilizados no documento WSDL
- Os tipos são declarados num XML schema
- Podem estar num ficheiro externo que contenha o xsd

Binding

- A função do *binding* é tornar concreto o serviço definindo a forma como funciona
- Exemplos:
  - Protocolo de transporte: SOAP; HTTP; SMTP
  - Valor do *SOAP Action*
  - Formatação da mensagem

- Um *portType* pode ter um ou mais *bindings* associados.

Exemplo de *binding* para HTTP

```
<binding name="PriceCheckSOAPBinding" type="pc:PriceCheckPortType">
   <soap:binding style="rpc"/>
   <operation name="checkPrice" transport= http://schemas.xmlsoap.org/soap/http/>

   <soap:operation soapAction=" "/>
     <input>
        <soap:body use="encoded"
               namespace= http://www.skatestown.com/services/PriceCheck
               encodingStyle= http://schemas.xmlsoap.org./soap/encoding//>
     </input>
     <output>
         <soap:body use="encoded"
               namespace= http://www.skatestown.com/services/PriceCheck
               encodingStyle= http://schemas.xmlsoap.org/soap/encoding//>
     </output>
  </operation>
</binding>
```

Exemplo de *binding* para SMTP

```
<!- - Binding definitions - ->
<binding name="PriceCheckSMTPBinding"
type="pc:PriceCheckPortType">
  <soap:binding style= "document"
    Transport= "http://schemas.xmlsoap.org/soap/smtp"/>
  <operation name="checkPrice">
    <input>
      <soap:body use="literal"/>
    </ input>
    <output>
      <soap:body use="literal"/>
      </ output>
    </ operation>
  </ binding>
```

Exemplo de opções no *binding*

```
public void myMethod(int x, float y);
```

RPC/encoded

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

RPC/literal

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

document/literal **Mais utilizado**

```
<soap:envelope>
    <soap:body>
        <xElement>5</xElement>
        <yElement>5.0</yElement>
    </soap:body>
</soap:envelope>
```

document/encoded

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

Não é utilizado

Service e Port

```
<! - - Service definition - ->
<service name="PriceCheckService">
     <port name="PriceCheck" binding="pc:PriceCheckSOAPBinding">
        <soap:address location = http://localhost:8080/axis/services/PriceCheck />
     </port>

</service

<! - - Service definition - - >
<service name="PriceCheckSMTPService">
   <port name="PriceCheckSMTP"  binding="PriceCheckSMTPBinding">
        <soap:address location = mailto:priceCheck@skstestown.com />
   </port>
</ service>
```

- Define o endereço da rede da rede onde o Web service é disponibilizado.
- Se existirem vários bindings são definidos vários ports
  - exemplo para http ou o endereço de email para SMTP