# Web Services e SOAP

Alexandre Zua Caldeira
Tecnologias de Middleware 2006/2007

Faculdade de Ciências da Universidade de Lisboa

20.10.2006

# Web Services - À Procura de Uma Definição

**Definição A**

Uma aplicação acessível a outras aplicações através da *Web*.

Problemas:

- Por este ponto de vista quase tudo é um *Web service*: Muito vago.

# Web Services - À Procura de Uma Definição

## Definição da UDDI Consortium

*A self-contained, modular business applications that have* **open, Internet-oriented, standards-based interfaces**.

obs: O acesso aos serviços é feito através da invocação de funcionalidades descritas numa interface; a comunicação é feita usando protocolos para a internet.

Problemas:

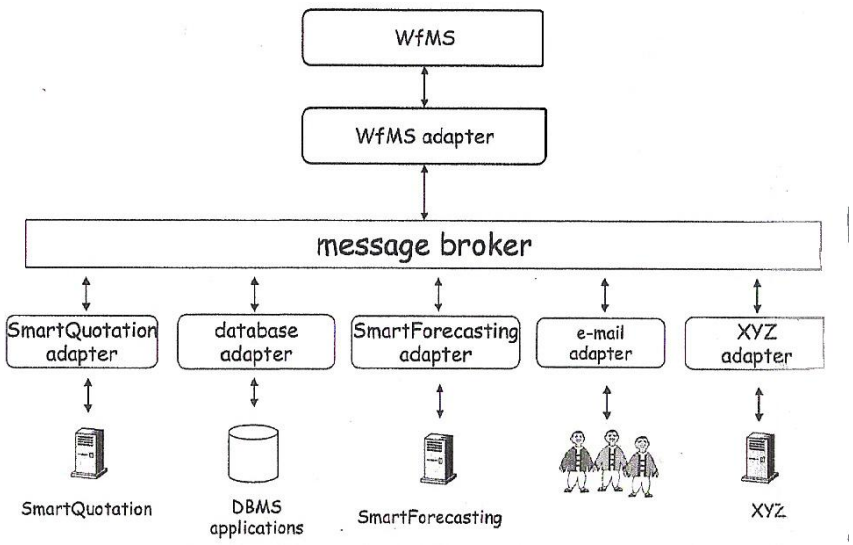
- O que é uma aplicação *self-contained, modular*?
- Quais os standards para a criação das interfaces?
- Onde estão as interfaces e como encontrá-las?
- Como invocar o serviço?

# Web Services - À Procura de Uma Definição

## Definição do W3C

*A software application identified by a URI (Universal Resource Identifier), whose interfaces and bindings are capable of being* **defined**, **described**, *and* **discovered** *as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols.*

# Motivação - Problema da Integração B2B




- As plataformas EAI (brokers) fornecem o mecanismo para a integração de sistemas autónomos e heterogéneos no contexto de uma organização.
- Os sistemas de Workflow implementam e executam os processos de negócio, distribuindo o trabalho pelas pessoas e aplicações, mas...
- O sistema pode estar **distribuído** por várias empresas/entidades distintas, relacionadas em algum processo de negócio partilhado.

# Motivação - Problema da Integração B2B

Tradicionalmente:

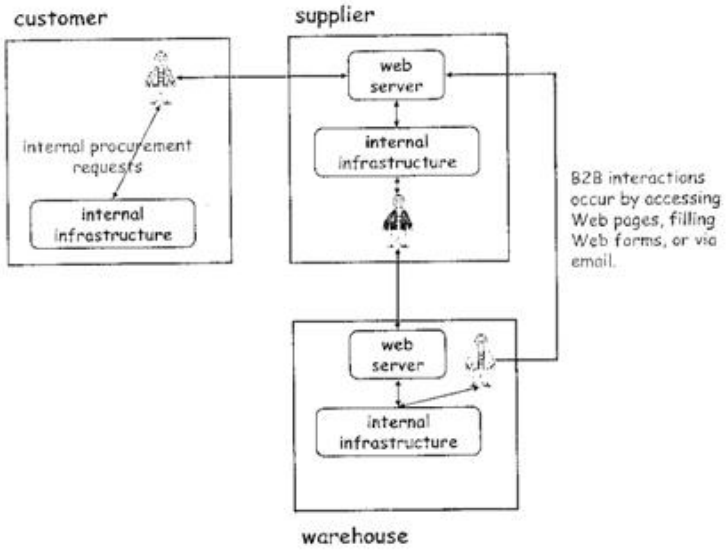



Fig. 5.1. Very often, integration across companies is still done manually

Questão Fundamental: **Onde colocar o middleware para a integração**?

- Numa das partes?
- Confiar em terceiros? Peer-2-Peer?

# Motivação - Problema da Integração B2B

Tradicionalmente:

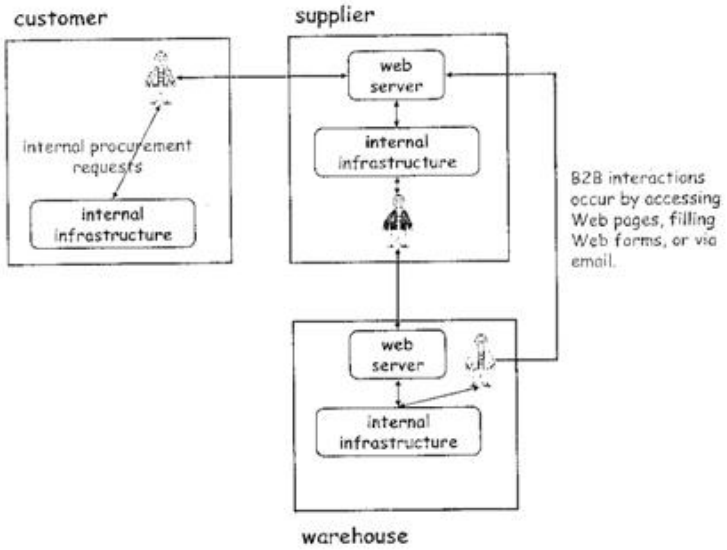



Fig. 5.1. Very often, integration across companies is still done manually

Questão Fundamental: **Onde colocar o middleware para a integração**?

- Numa das partes? **autonomia** e **modularidade**.
- Confiar em terceiros? Peer-2-Peer?

# Soluções com Middleware Convencional

1. (Solução 1) B2B - versão EAI

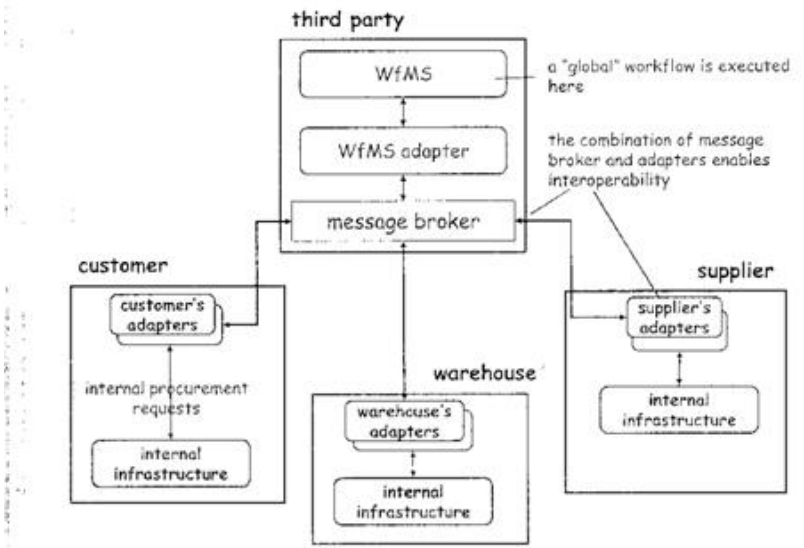



**Fig. 5.2.** B2B integration performed in the same way EAI is done. While this is conceptually possible, it rarely happens in practice due to lack of trust, autonomy, and confidentiality reasons

# Soluções com Middleware Convencional

1. (Solução 1) B2B - versão EAI

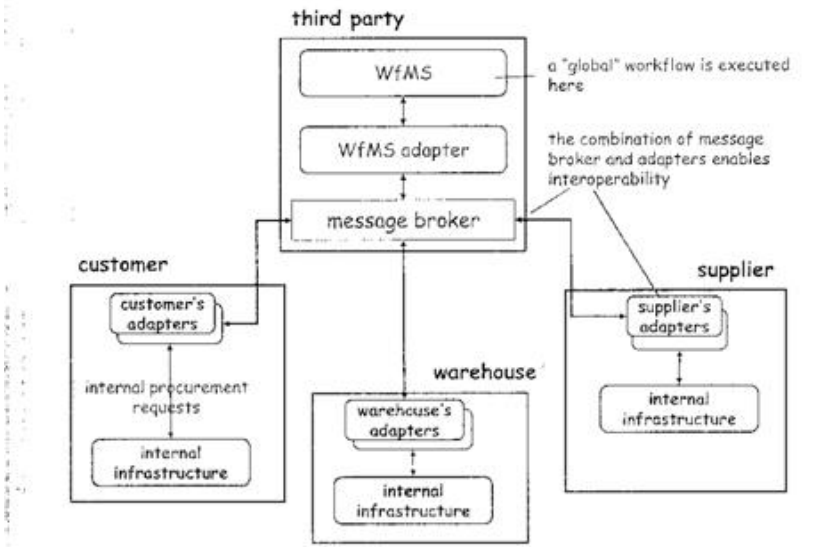



**Fig. 5.2.** B2B integration performed in the same way EAI is done. While this is conceptually possible, it rarely happens in practice due to lack of trust, autonomy, and confidentiality reasons

Problema: Segurança, Fiabilidade

# Soluções com Middleware Convencional

1. (Solução 2) B2B - versão peer-to-peer

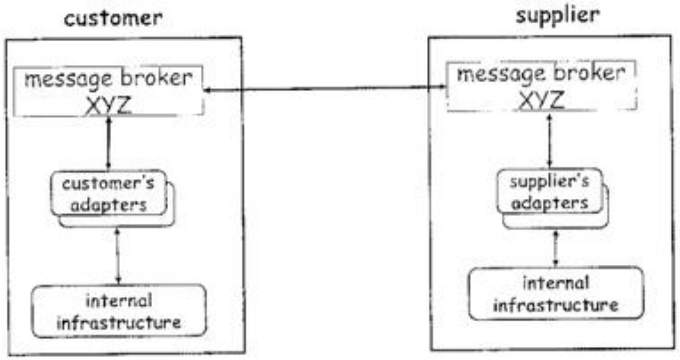



Fig. 5.3. Point-to-point integration across companies

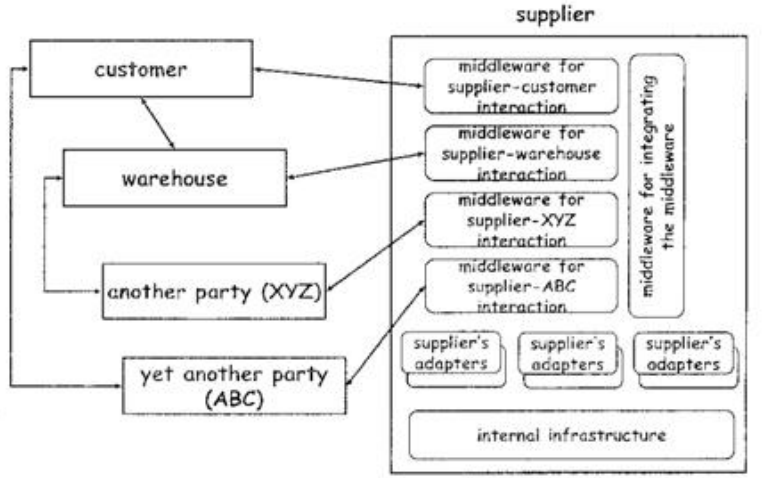



Fig. 5.4. The lack of a central middleware platform means that interactions are managed in a point-to-point manner, possibly using different middleware platforms to communicate with different parties

# Soluções com Middleware Convencional

1. (Solução 2) B2B - versão peer-to-peer

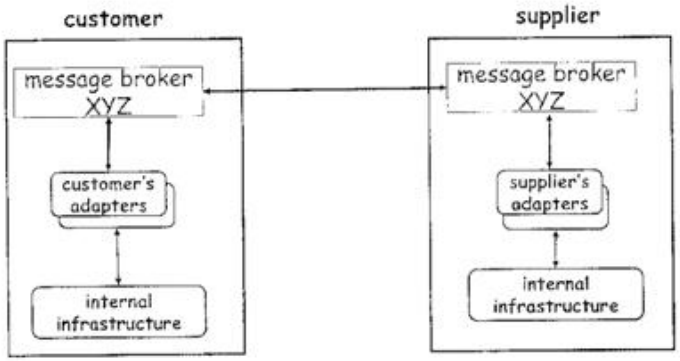



Fig. 5.3. Point-to-point integration across companies

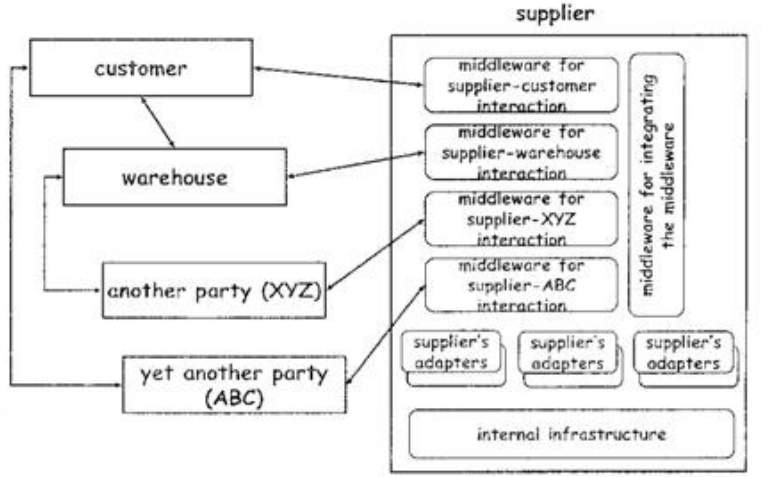



Fig. 5.4. The lack of a central middleware platform means that interactions are managed in a point-to-point manner, possibly using different middleware platforms to communicate with different parties

Problema: Escalabilidade

# Contribuição dos Web Services para a Integração B2B

1. Arquitecturas orientadas a serviços
   - Um serviço é um procedimento, método ou objecto com uma interface publicada que pode ser invocada por aplicações clientes.
   - Web services são como os serviços de middleware. Funcionam como *wrappers* das funcionalidades internas, expondo-as através de uma interface publicada.
2. Protocolos de Middleware Peer-to-Peer
   - Redesenhar os protocolos de middleware para funcionamento em modo Peer-to-Peer.
3. Standards
   - Linguagens e Protocolos standards necessários para a escalabilidade. A não utilização de standards levaria à profusão de formatos.

# Contribuição dos Web Services para a Integração B2B

## All Together

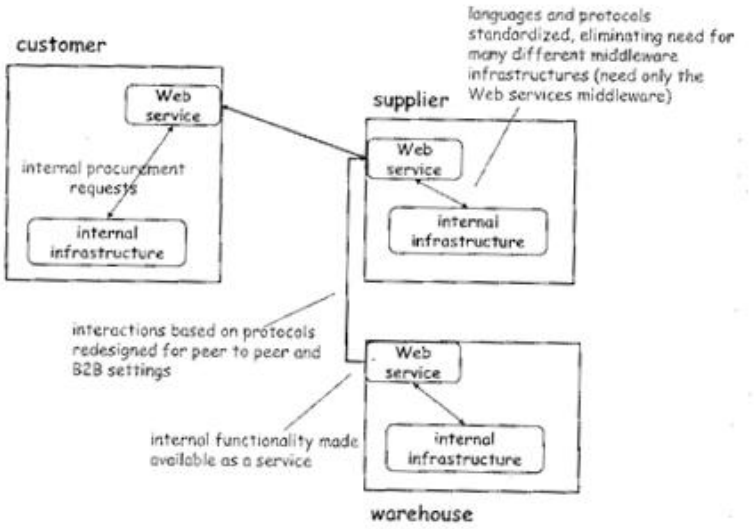



Fig. 5.5. Service-oriented architectures, redesigned (peer-to-peer) middleware protocols, and standardization are the main ingredients of the solution offered by Web services

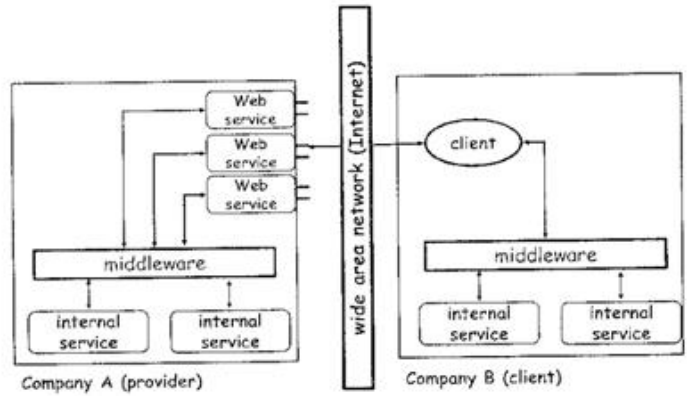



Fig. 5.6. Web services provide an entry point for accessing local services

# Contribuição dos Web Services para a Integração EAI

All Together

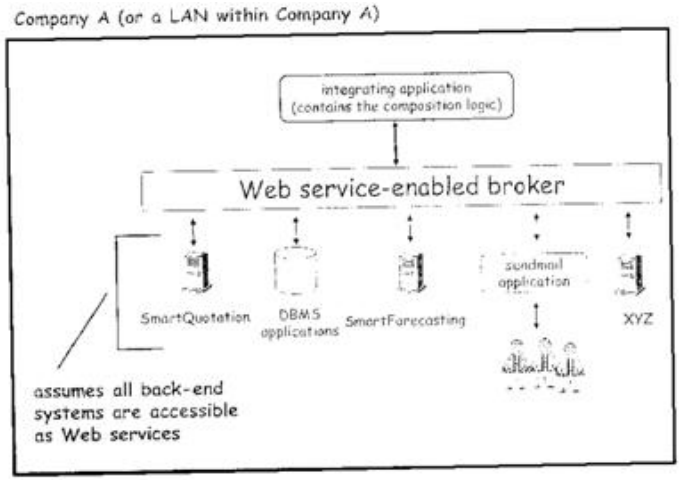



**Fig. 5.7.** Web services can be also used within the enterprise or even a LAN, to integrate enterprise applications

# Service Description, Discovery and Interactions

- Descrição de Serviços (WSDL)
- Descoberta de Serviços (UDDI)
  - Descrições dos Serviços são guardados num *service directory*
  - Os clientes utilizam o *Service Directory* para localizar os serviços
  - A especificação do UDDI define uma API para o *publishing* e *discovery* de informação sobre serviços no *service directory*
- Interacções entre Web Services (SOAP)
  - Transporte
  - Basic and Secure messaging

# Arquitectura

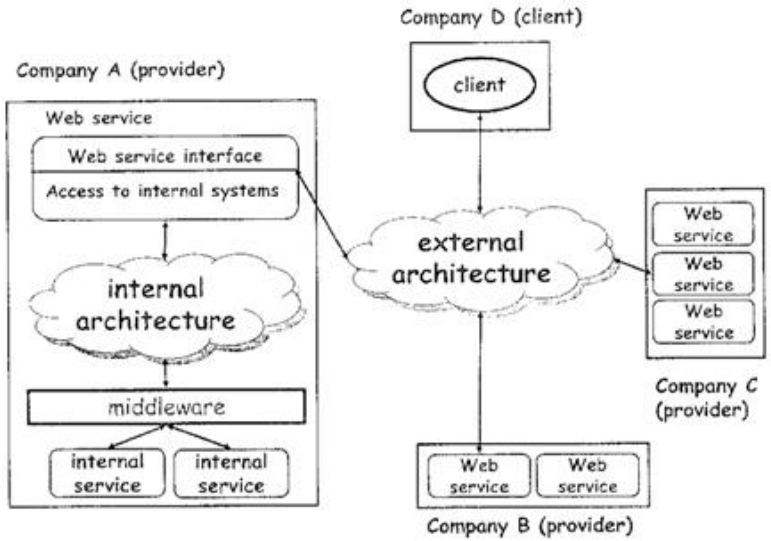
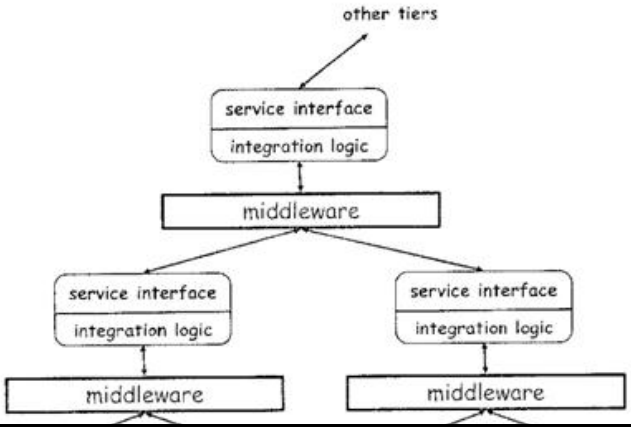
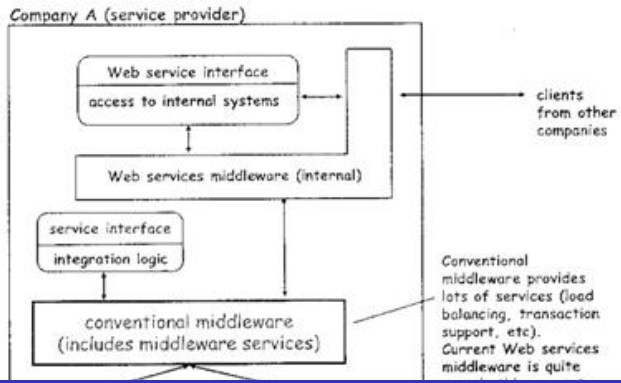



Fig. 5.10. Web services require an internal and an external architecture, along with corresponding middleware support

# Arquitectura Interna

- Arquitectura por **camadas**. Os Web services são mais uma camada por cima do middleware convencional.
- A implementação está no middleware convencional. *Web services* funcionam como **wrappers** dos serviços existentes, servindo de interface entre as camadas de nível superior e inferior.
- Através da **composição** pode-se contruir novos serviços (flexibilidade, escalabilidade). Web Services de **ordem superior** são Web Services resultantes da composição de Web Services.
- Uma nova camada e o empacotamento de mensagens entre camadas causam um *overhead* no processamento das mensagens.



# Arquitectura Externa

- Os fornecedores de serviços criam Web services e uma interface para a invocação dos serviços.
- Geram descrições dos serviços e publica-as no *service registry*.
- Os clientes consultam o *service registry* por um determinado serviço.
- O *service registry* responde com a localização do serviço e como o invocar.
- O *binding* entre o cliente e o fornecedor é feito na invocação do serviço.

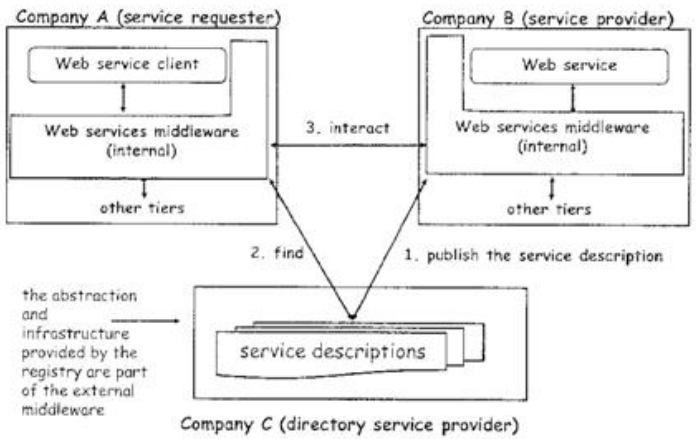



Fig. 5.13. External architecture of Web services

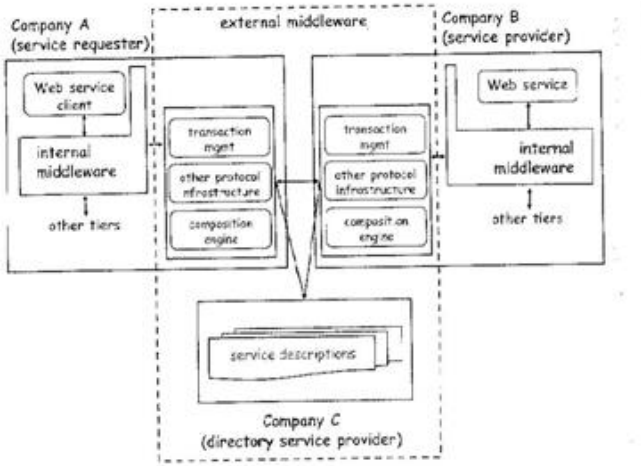



Fig. 5.14. External architecture of a Web service augmented with peer-to-peer protocol execution capabilities and with service composition

# Enquadramento

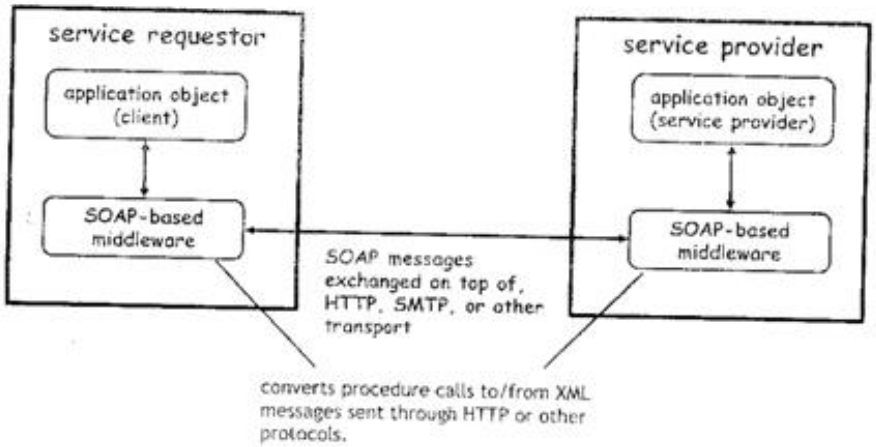



Fig. 6.1. Clients can invoke Web services by exchanging SOAP messages

# Enquadramento

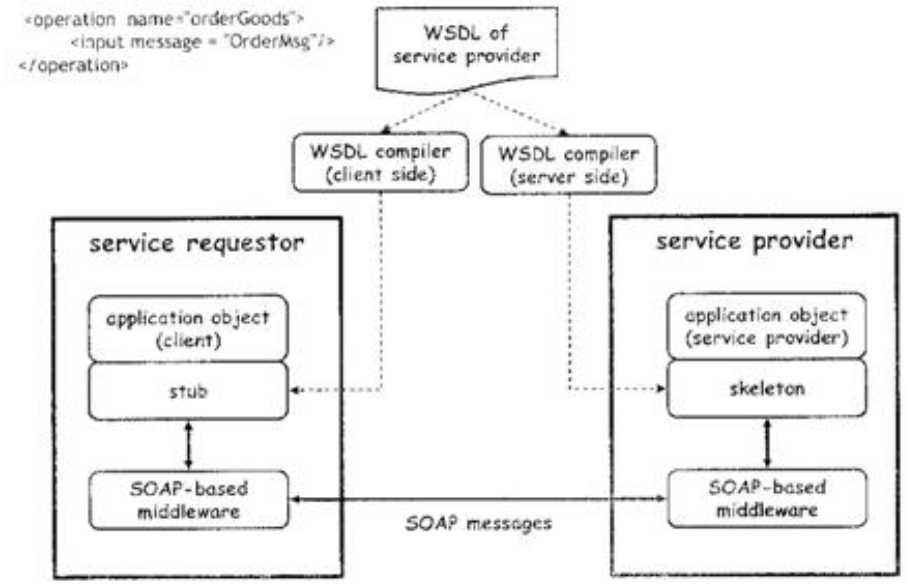



Fig. 6.2. WSDL specifications can be compiled into stubs and skeletons, analogously to IDL in conventional middleware. Dashed lines denote steps performed at development time, solid lines refer to run-time

# Enquadramento

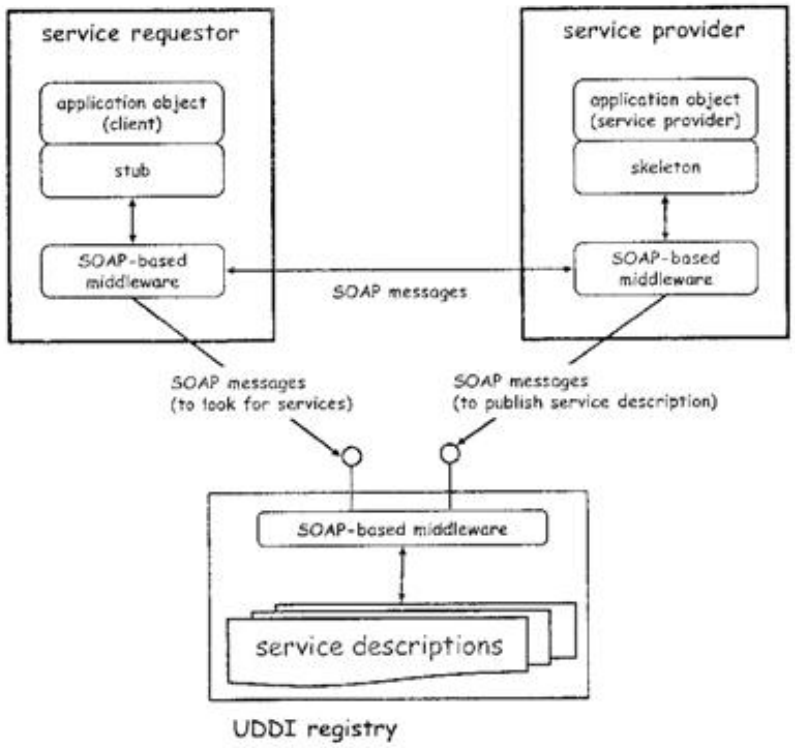



**Fig. 6.3.** Providers advertise their services in a UDDI registry. Clients (at development time or run-time) look for services in the registry, thereby statically or dynamically binding to a service. Then, clients can invoke the service

# Objectivos

Protocolo para a comunicação entre *Web services* que especifica:

- Formato genérico das mensagens
- Convenções para a implementação de comunicação estilo RPC
- Regras e acções para os nós que processam as mensagens SOAP.

# Formato Genérico das Mensagens SOAP

- SOAP Envelope
- SOAP Header
- SOAP Body

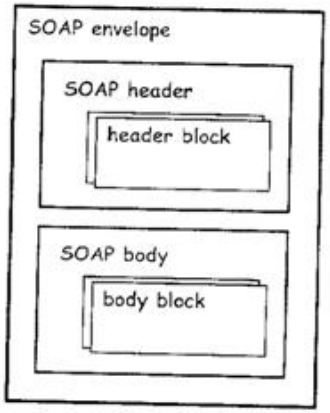



Fig. 6.4. Schematic representation of a SOAP message

# Tipos de Interacção

- Document-style
- RPC-style

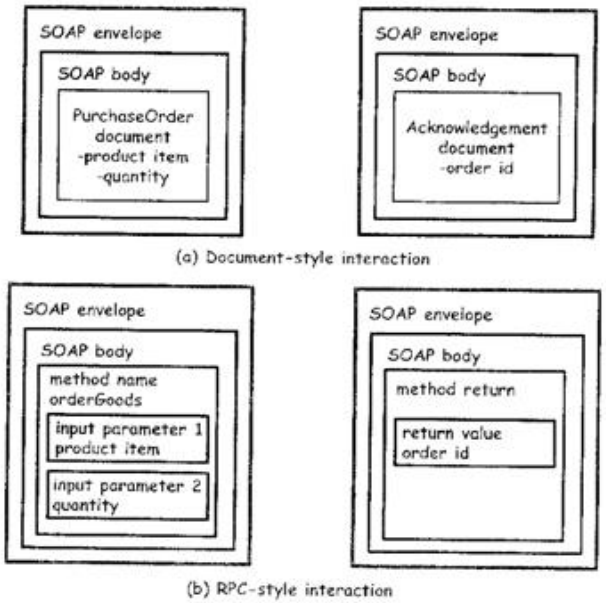



**Fig. 6.5.** The document-style interaction involves the client sending a *PurchaseOrder* document and the supplier responding with an *Acknowledgement* document. The RPC-style interaction involves the client making a *orderGoods* method call using a SOAP message and the supplier responding with a order id as the return value in another SOAP message

# Codificação da informação em XML

Exemplos de codificação para uma mesma mensagem

```
<ProductItem>
  <name>...</name>
  <type>...</type>
  <make>...</make>
</ProductItem>
```

```
<ProductItem
  name="..."
  type="..."
  make="..."
/>
```

```
<ProductItem name="...">
  <type>...</type>
  <make>...</make>
</ProductItem>
```

Fig. 6.6. Different encodings result in different XML structures in a SOAP message

Cliente e servidor devem por isso concordar na representação dos dados

# Processamento da Mensagem

- Cliente e Servidor estão separados por nós de processamento.
- O nós podem desempenhar vários papéis: *none*, *next* e *ultimateReceiver*.
- Os blocos do cabeçalho podem definir quais os papéis que os devem processar:
  - none: o bloco não deve ser processado por nenhum nó. Pode ser lido para obter informação necessária para processar outros blocos.
  - next: todos os nós devem processar o bloco.
  - ultimateReceiver: o bloco deve ser processado apenas pelo destinatário da mensagem.

# SOAP e Protocolos de Transporte

- SOAP define como as mensagens devem ser incorporadas num protocolo de transporte.
- Tipicamente são usados os protocolos HTTP e SMTP (SOAP Assíncrono).

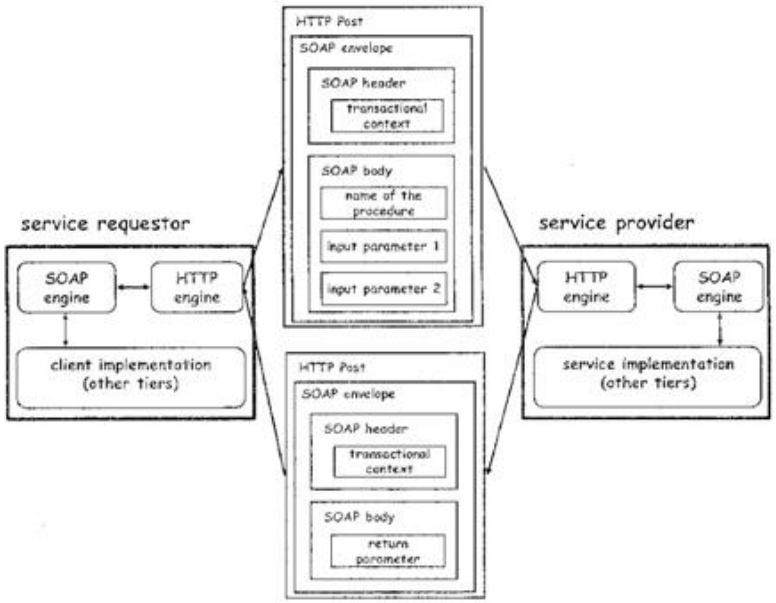



Fig. 6.8. An RPC call using SOAP over HTTP

# Implementação do SOAP

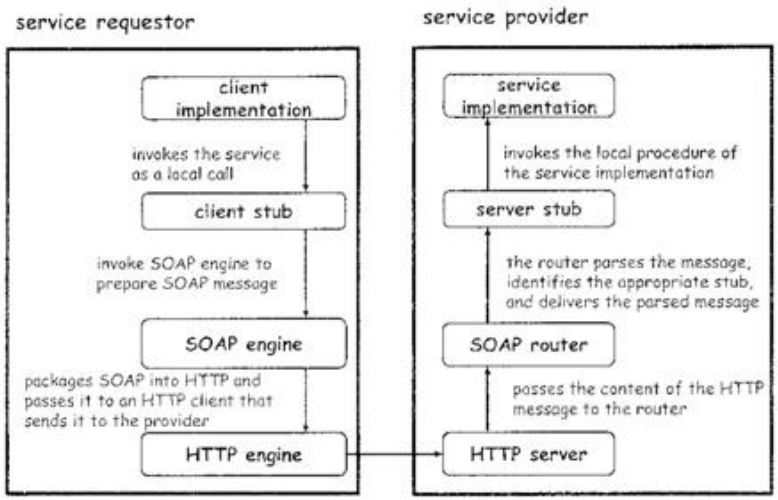



Fig. 6.9. A simple implementation of SOAP

# Exemplos

- Programa Cliente (Java)
- API
- Mensagens SOAP
- Implementação do Serviço (Java)

```
/** Hello client in Java */

import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;

public class Example1_client {

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

  System.out.println("\n\nCalling the SOAP Server to say hello\n\n");
  URL url = new URL (args[0]);
  String name = args[1];

  Call call = new Call ( );
  call.setTargetObjectURI("urn:Example1");
  call.setMethodName("sayHello");
  call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC;);
  Vector params = new Vector ( );
  params.addElement (new Parameter("name", String.class, name, null));
  call.setParams (params);

  System.out.print("The SOAP Server says: ");

  Response resp = call.invoke(url, "");

  if (resp.generatedFault ( )) {
   Fault fault = resp.getFault ( );
   System.out.println ("\nOuch, the call failed: ");
   System.out.println ("  Fault Code   = " + fault.getFaultCode ( ));
   System.out.println ("  Fault String = " + fault.getFaultString ( ));
  } else {
   Parameter result = resp.getReturnValue ( );
   System.out.print(result.getValue ( ));
   System.out.println( );
  }
 }
}
```

# Exemplos

- Programa Cliente (Java)
- API
- Mensagens SOAP
- Implementação do Serviço (Java)

Soap Packages
**org.apache.soap**
**org.apache.soap.encoding**
**org.apache.soap.encoding.literalxml**
**org.apache.soap.encoding.soapenc**
**org.apache.soap.encoding.xmi**
**org.apache.soap.messaging**
**org.apache.soap.providers**
**org.apache.soap.providers.com**
**org.apache.soap.rpc**
**org.apache.soap.server**
**org.apache.soap.server.http**
**org.apache.soap.transport**
**org.apache.soap.transport.http**
**org.apache.soap.transport.smtp**
**org.apache.soap.util**
**org.apache.soap.util.mime**
**org.apache.soap.util.net**
**org.apache.soap.util.xml**

Package org.apache.soap.rpc
Class Summary

| | |
|---|---|
| **Call** | A Call object represents an RPC call. |
| **Parameter** | A Parameter represents an argument to an RPC call. |
| **Response** | A Response object represents an RPC response. |
| **RPCConstants** | SOAP constants used for RPC. |
| **RPCMessage** | An RPCMessage is the base class that Call and Response extend from. |
| **SOAPContext** | SOAP context for a message. |

Package org.apache.soap

Class Summary

| | |
|---|---|
| **Body** | A Body object represents the contents and semantics of a <SOAP-ENV:Body> element. |
| **Constants** | SOAP constants. |
| **Envelope** | An Envelope object represents the contents and semantics of an <SOAP-ENV:Envelope> element. |
| **Fault** | A Fault object represents the contents and semantics of a <SOAP-ENV:Fault> element. |
| **Header** | A Header object represents the contents and semantics of a <SOAP-ENV:Header> element. |
| **Utils** | SOAP utilities. |

Exception Summary
**SOAPException** SOAP Exceptions.

# Exemplos

- Programa Cliente (Java)
- API
- Mensagens SOAP
- Implementação do Serviço (Java)

**Hello request**

```
<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <s:Body>
  <m:sayHello xmlns:m='urn:Example1'>
   <name xsi:type='xsd:string'>James</name>
  </m:sayHello>
 </s:Body>
</s:Envelope>
```

**Hello response**

```
<s:Envelope
 xmlns:s="http://www.w3.org/2001/06/soap-envelope"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <s:Body>
  <n:sayHelloResponse xmlns:n="urn:Example1">
   <return xsi:type="xsd:string">
    Hello James
   </return>
  </n:sayHelloResponse>
 </s:Body>
</s:Envelope>
```

# Exemplos

- Programa Cliente (Java)
- API
- Mensagens SOAP
- Implementação do Serviço (Java)

```
/** A classe que implementa o Serviço */
public class Hello {
  public String sayHello(String name) {
    return "Hello " + name;
  }
}
```

**Apache SOAP configuration file**

```
<root>
 <dd:service xmlns:dd="http://xml.apache.org/xml-soap/deployment"
         id="urn:Example1">
  <dd:provider type="java"
           scope="Application"
           methods="sayHello">
    <dd:java class="samples.Hello"
          static="false" />
  </dd:provider>
  <dd:faultListener>
   org.apache.soap.server.DOMFaultListener
  </dd:faultListener>
  <dd:mappings />
 </dd:service>
</root>
```

```
> java samples.Hello http://localhost/soap/servlet/rpcrouter James

Calling the SOAP Server to say hello
The SOAP Server says: Hello James

>
```

[1]G. Alonso, F. Casati, H. Kuno, V. Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer Verlag 2004

[2]`http://www.oreilly.com/catalog/progwebsoap/chapter/ch03.html`