SISTEMAS DISTRIBUÍDOS

Capítulo 5

Web services e modelos alternativos de interacção cliente/servidor na internet

NOTA PRÉVIA

A apresentação utiliza algumas das figuras do livro de base do curso

G. Coulouris, J. Dollimore and T. Kindberg,
Distributed Systems - Concepts and Design,
Addison-Wesley, 5th Edition, 2005

Organização do capítulo

Web services SOAP

Web services REST

Invocação remotas assíncronas

- Ajax
- GWT

WEB SERVICES: MOTIVAÇÃO

Modelo para acesso a servidores: invocação remota

Desenhado para garantir inter-operabilidade

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using ***HTTP*** *with an* ***XML serialization*** *in conjunction with other Web-related standards*

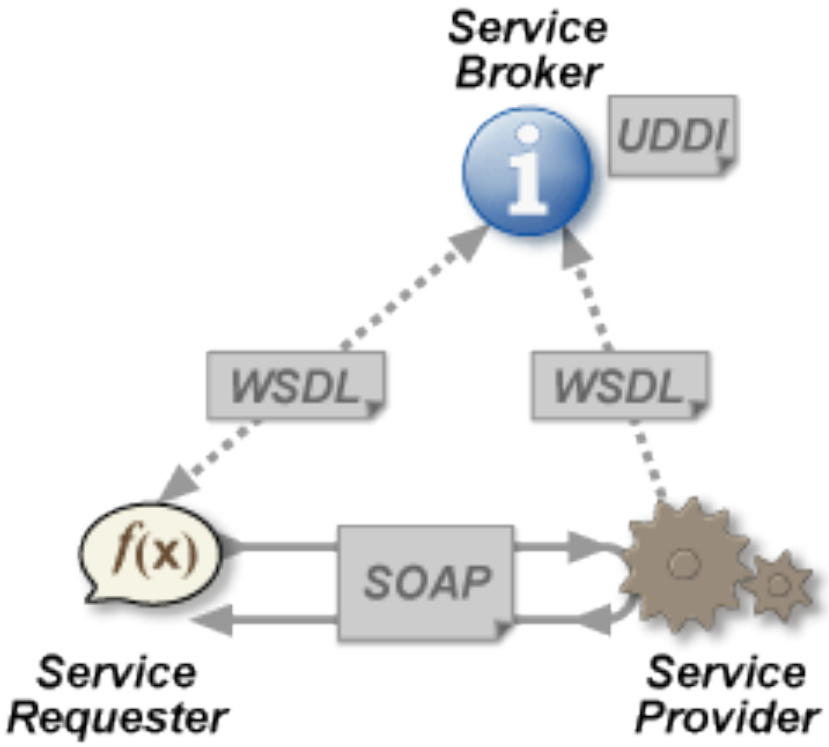
Protocolo: HTTP e HTTPS (eventualmente SMTP)
Referências: USL/URI
Representação dos dados: XML

COMPONENTES BÁSICOS

SOAP: protocolo de invocação remota

WSDL: linguagem de especificação de serviços

UDDI: serviço de registo



SOAP

Protocolo para trocar mensagens XML

- Inclui definição do formato das mensagens a trocar
- Inclui um mecanismo de ligação das mensagens SOAP com o protocolo de transporte usado: HTTP ou HTTPS (ou SMTP, …)
- Inclui mecanismo para tratar falhas

SOAP: INTERACÇÕES

Oneway: mensagem undireccional do cliente para o servidor

Pedido-resposta: interacção cliente-servidor-cliente

Notificação: interacção unidireccional servidor-cliente

E.g. callback, notificação

Notificação-resposta: interacção servidor-cliente-servidor

WSDL – IDL PARA *WEB SERVICES*

Definição da interface em XML

- WSDL permite definir a interface do serviço, indicando quais as operações disponíveis
- WSDL define as mensagens trocadas na interacção (e.g. na invocação duma operação, quais as mensagens trocadas)
- WSDL permite também definir a forma de representação dos dados e a forma de aceder ao serviço

- Especificação WSDL bastante verbosa – normalmente criada a partir de interface ou código do servidor
  - Em Java e .NET existem ferramentas para criar especificação a partir de interfaces Java
  - Sistemas de desenvolvimento possuem *wizards* que simplificam tarefa

WSDL: ELEMENTOS

No seu conjunto, um documento WSDL comporta os seguintes elementos:

Types - definições de tipos de dados, num dado sistema (e.g., XSD)

Messages - definição abstracta dos dados trocados nas operações

Operation - definição abstracta das acções suportadas pelo serviço

Port Type - conjunto abstracto das operações suportadas por um ou mais *endpoints*

Binding - uma especificação concreta do protocolo e formato de dados usados por um *port type.*

Port - um *endpoint* concreto que combina um endereço e um *binding* particulares.

Serviço - um conjunto de endpoints/ports relacionados.

WSDL a partir do Java

Anota-se uma classe usando: WebService e WebMethod

```
import javax.jws.*;
import java.util.*;

@WebService()
public class SimpleWSServer {
        private Map<String, MyInfo> infos;

        public SimpleWSServer() {
                infos = new HashMap<String, MyInfo>();
        }
        @WebMethod()
        public void addInfo( String name, int age) {
                infos.put( name, new MyInfo( name, age));
        }
        @WebMethod()
        public MyInfo getInfo( String name) throws InfoNotFoundException {
                if( infos.containsKey( name))
                        return infos.get( name);
                else
                        throw new InfoNotFoundException();
        }
```

Para gerar WSDL:
wsgen -cp . aulaWS.SimpleWSServer -wsdl -servicename {http://basename}/MyServer

CÓDIGO GERADO: SIMPLEWSSERVER.WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://aulaWS/" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://
aulaWS/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://aulaWS/" schemaLocation="SimpleWSServer_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="getInfo">
    <part name="parameters" element="tns:getInfo"/>
  </message>
  <message name="getInfoResponse">
    <part name="parameters" element="tns:getInfoResponse"/>
  </message>
  <message name="InfoNotFoundException">
    <part name="fault" element="tns:InfoNotFoundException"/>
  </message>
  ...
  <portType name="SimpleWSServer">
    <operation name="getInfo">
      <input message="tns:getInfo"/>
      <output message="tns:getInfoResponse"/>
      <fault name="InfoNotFoundException" message="tns:InfoNotFoundException"/>
    </operation>
  ...
  </portType>
</definitions>
```

CÓDIGO GERADO: SIMPLEWSSERVER.WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://aulaWS/" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://
aulaWS/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://aulaWS/" schemaLocation="SimpleWSServer_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="getInfo">
    <part name="parameters" element="tns:getInfo"/>
  </message>
  <message name="getInfoResponse">
    <part name="parameters" element="tns:getInfoRespons
  </message>
  <message name="InfoNotFoundException">
    <part name="fault" element="tns:InfoNotFoundException"/>
  </message>
  ...
  <portType name="SimpleWSServer">
    <operation name="getInfo">
      <input message="tns:getInfo"/>
      <output message="tns:getInfoResponse"/>
      <fault name="InfoNotFoundException" message="tns:InfoNotFoundException"/>
    </operation>
  ...
  </portType>
</definitions>
```

portType

Define os serviços do Web Service

Mensagens permitem assinatura dos serviços

CÓDIGO GERADO: MYSERVER.WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://a
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="InfoNotFoundException" type="t
 ...
  <xs:element name="getInfo" type="tns:getInfo"/>
  <xs:element name="getInfoResponse" type="tns:getInfoResponse"/>
  <xs:complexType name="getInfo">
    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="getInfoResponse">
    <xs:sequence>
      <xs:element name="return" type="tns:myInfo" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="myInfo"/>
  <xs:complexType name="InfoNotFoundException">
    <xs:sequence>
      <xs:element name="message" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

tipos

Tipos usados nas mensagens são definidos em XML Schema

CÓDIGO GERADO: SIMPLEWSSERVER.WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://aulaWS/" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://
aulaWS/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://aulaWS/" schemaLocation="SimpleWSServer_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="getInfo">
    <part name="parameters" element="tns:getInfo"
  </message>
  <message name="getInfoResponse">
    <part name="parameters" element="tns:getInfoResponse"/>
  </message>
  <message name="InfoNotFoundException">
    <part name="fault" element="tns:InfoNotFoundException"/>
  </message>
  ...
  <portType name="SimpleWSServer">
    <operation name="getInfo">
      <input message="tns:getInfo"/>
      <output message="tns:getInfoResponse"/>
      <fault name="InfoNotFoundException" message="tns:InfoNotFoundException"/>
    </operation>
  ...
  </portType>
</definitions>
```

message

As mensagens podem ser de input, output ou assinalar erros

CÓDIGO GERADO: MYSERVER.WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://localhost/WS/" name="MyServer" ... xmlns:tns="http://localhost/WS/" ... >
  <import namespace="http://aulaWS/" location="SimpleWSServer.wsdl"/>
  <binding name="SimpleWSServerPortBinding" type="ns1:SimpleWSServer" xmlns:ns1="http://aulaWS/">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="getInfo">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="InfoNotFoundException">
        <soap:fault name="InfoNotFoundException" use="literal"/>
      </fault>
    </operation>
    ...
  </binding>
  <service name="MyServer">
    <port name="SimpleWSServerPort" binding="tns:SimpleWSServerPortBinding">
      <soap:address location="http://webserver/WS/MyServer"/>
    </port>
  </service>
</definitions>
```

binding

Concretiza o serviço definindo como se processam as interacções

CÓDIGO GERADO: MYSERVER.WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://localhost/WS/" name="MyServer" ... xmlns:tns="http://localhost/WS/" ... >
  <import namespace="http://aulaWS/" location="SimpleWSServer.wsdl"/>
  <binding name="SimpleWSServerPortBinding" type="ns1:SimpleWSServer" xmlns:ns1="http://aulaWS/">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="getInfo">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="InfoNotFoundException">
        <soap:fault name="InfoNotFoundException" use="literal"/>
      </fault>
    </operation>
    ...
  </binding>
  <service name="MyServer">
    <port name="SimpleWSServerPort" binding="tns:SimpleWSServerPortBinding">
      <soap:address location="http://webserver/WS/MyServer"/>
    </port>
  </service>
</definitions>
```

port

Define o endereço de rede onde o web service é disponibilizado

WSDL a partir do .NET

```
using System;
using System.WebServices;
using System.Xml.Serialization;

[WebService(Namespace="http://127.0.0.1:8088/PingImpl”)]
public class PingImpl : System.WebServices.WebService
{
    [WebMethod]
    public string ping( string v )
    {
            return v.ToUpper() ;
    }
}
```

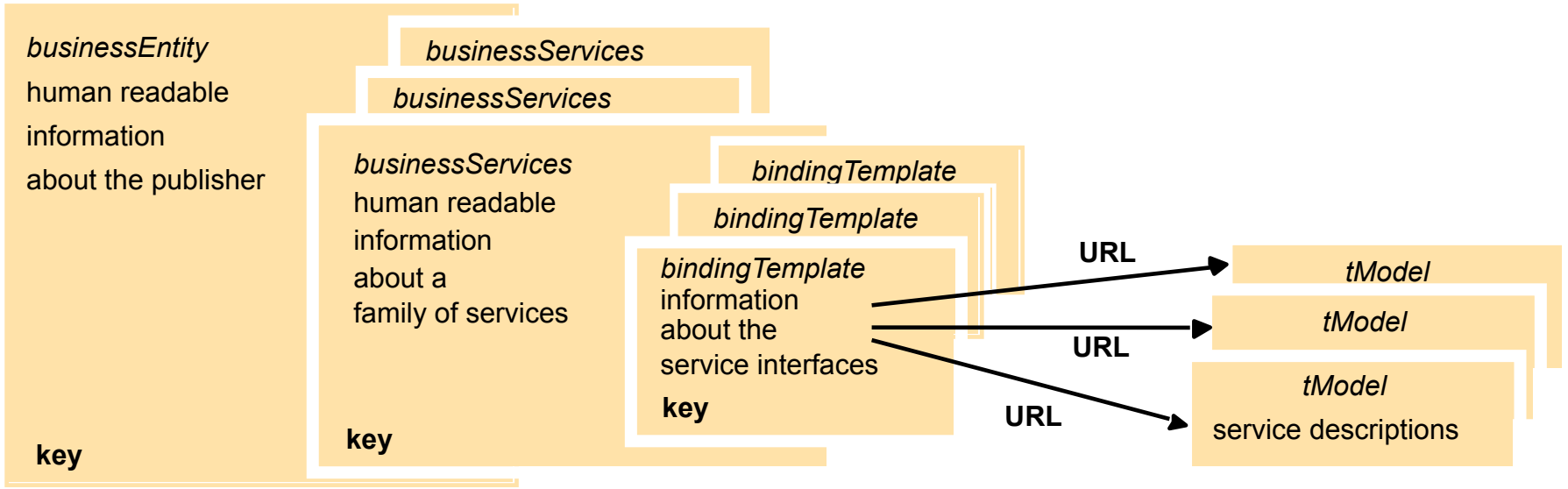
UDDI

Protocolo de ligação ou binding entre o cliente e o servidor

Os fornecedores dos serviços publicam a respectiva interface

O protocolo de inspecção permite verificar se uma dado serviço existe baseado na sua identificação

O UDDI permite encontrar o serviço baseado na sua definição – capability lookup



CLIENTES

O cliente pode ter um **static proxy** criado antes da execução (static stub) que é compilado a partir do WSDL

O cliente pode também usar um **dynamic proxy** uma classe que é criada durante a execução a partir do WSDL

Cliente em Java

```
import javax.xml.ws.WebServiceRef;

public class SimpleClient {
  @WebServiceRef(wsdlLocation="http://localhost:8080/WS/SimpleServer?wsdl")
  static MyServer service;

  public static void main(String[] args) {
     new SimpleClient().doTest(args);
   }

  public void doTest(String[] args) {
    try {
      SimpleWSServer port = service.getSimpleWSServerPort();

       MyInfo response = port.getInfo(args[0]);
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

WS-*

WS-Reliability: especificação que permite introduzir fiabilidade nas invocações remotas (com as diferentes semânticas)

- WS-ReliableMessaging

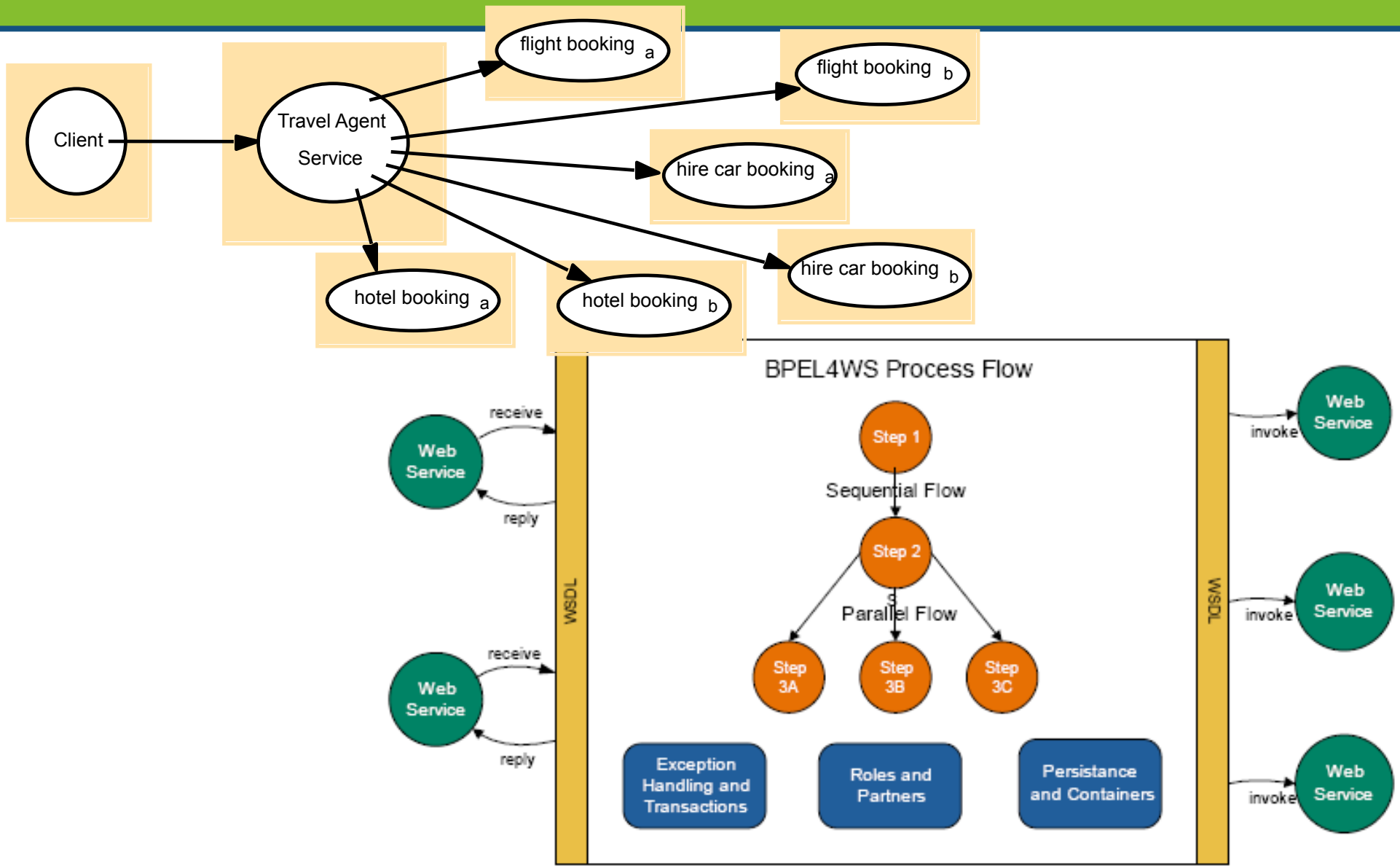
WS-Security: define como efectuar invocações seguras

WS-Coordination: fornece enquadramento para coordenar as acções de aplicações distribuídas (coreografia de web services)

- WS-AtomicTransaction: forncece coordenação transaccional (distribuída) entre múltiplos web services

E.g.: BPEL4WS, WSBPEL linguagem de orquestração

WS-COORDINATION



Web Services: resumo

Standard na indústria porque apresenta uma solução para integrar diferentes aplicações

- RMI – apenas Java
- Corba/IIOP – suporte mais complexo

Permite:

- Utilização de standards
  - HTTP, XML
- Reutilização de serviços (arquitecturas orientadas para os serviços)
  - WS-Coordination
- Modificação parcial/evolução independente dos serviços
  - WSDL
- Disponibilidade, assincronismo
  - WS-Eventing

WEB SERVICES: UTILIZAÇÕES POSSÍVEIS

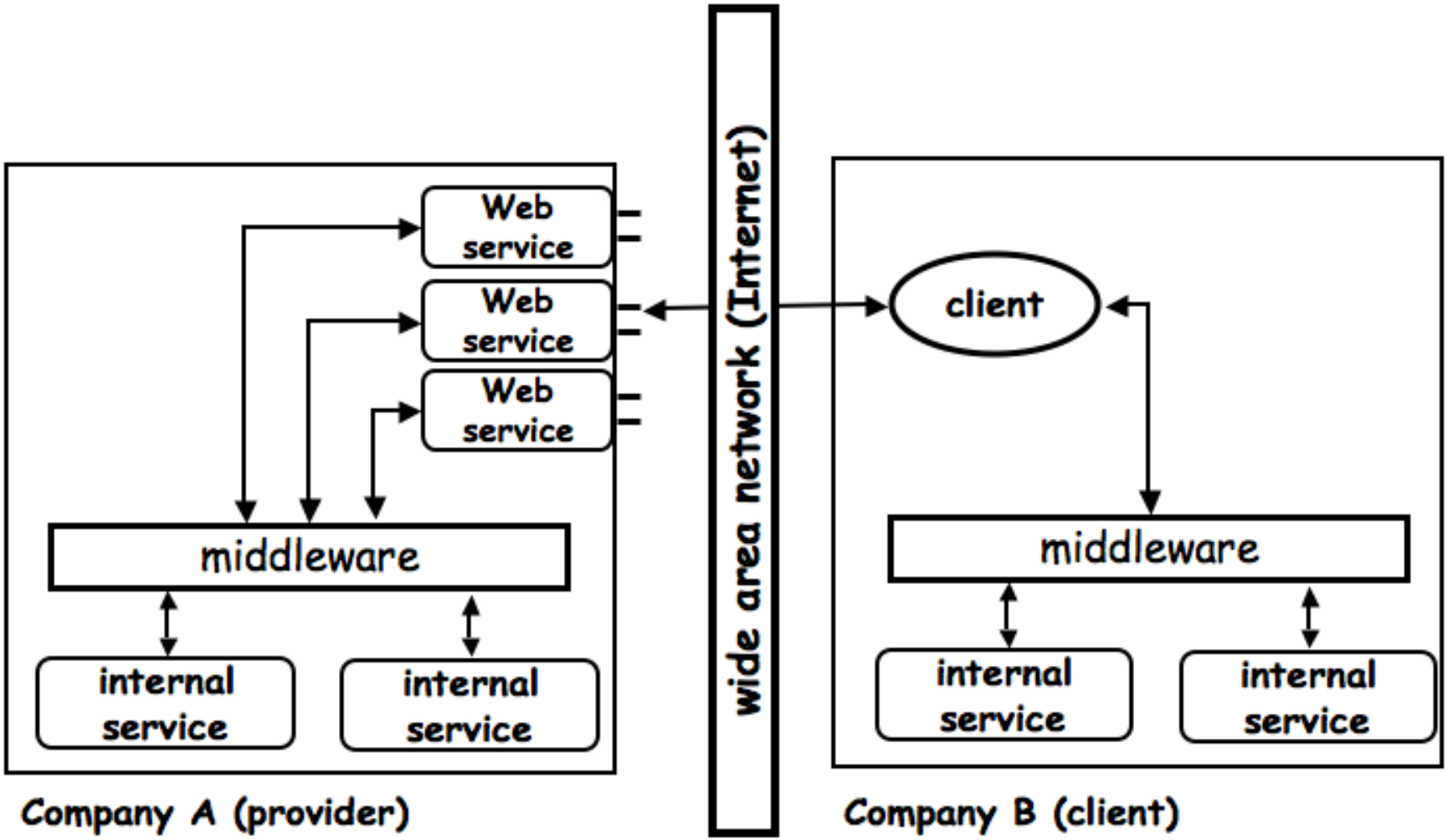
Inter-operação entre instituições

- Utilização tende a ser limitada
- Promessas de todos os serviços disponíveis para todos acabaram por não se concretizar

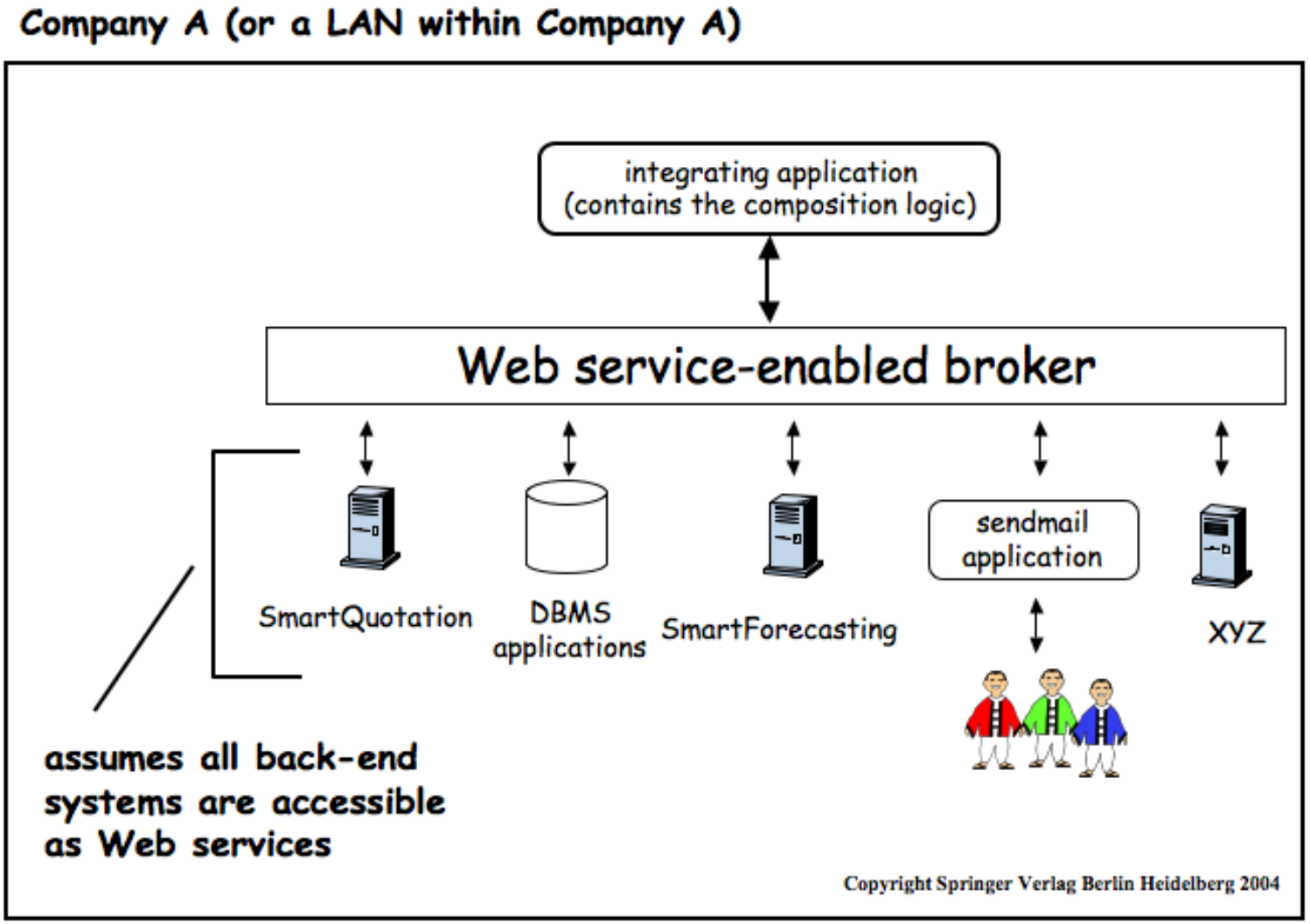
Inter-operação entre sistemas duma mesma instituição

- Utilização com forte implantação

WEB SERVICES: ENTRE ORGANIZAÇÕES



WEB SERVICES: DENTRO DE UMA ORGANIZAÇÃO



WEB SERVICES: PROBLEMAS

Desempenho:

- Complexidade do XML
- Difícil fazer *caching*: noção de operação + estado

Necessário cuidado para fazer evoluir o servidor

Complexidade

- Necessário utilização de ferramentas de suporte

REST: Representational State Transfer

Aproximação: vê uma aplicação como uma colecção de recursos

- Um recuso é identificado por um URI/URL
- Um URL devolve um documento com a representação do recurso
- Podem-se fazer referências a outros recursos usando *ligações (links)*

Estilo arquitectural, não um sistema de desenvolvimento

Aproximação proposta por Roy Fielding na sua tese de doutroamento

- Não como uma alternativa aos web services, mas como uma forma de aceder a informação

REST: PRINCÍPIOS DE DESENHO

Protocolo cliente/servidor stateless: cada pedido contém toda a informação necessária para ser processado – objectivo: tornar o sistema simples.

Recursos – no sistema tudo são recursos, identificados por um URI/URL.

- Recursos tipicamente armazenados num formato estruturado que suporta hiper-ligações (e.g. XML)

Interface uniforme: todos os recursos são acedidos por um conjunto de operações bem-definidas. Em HTTP: POST, GET, PUT, DELETE (equiv. criar, ler, actualizar, remover).

Cache: para melhorar desempenho, respostas podem ser etiquetadas como permitindo ou não *caching*.

Mais detalhes e exemplo

- Usa HTTP GET, POST, etc., mas
  - Usa dados bem-tipados (usando schemas XML, json)
- Um exemplo (de Roger L. Costello)
  - Uma empresa pretende disponibilizar web services REST para permitir aos seus clientes:
    - Obter uma lista de peças
    - Obter informação detalhada sobre uma peça
    - Submeter uma ordem de compra

EXEMPLO

Tornar uma lista de peças disponível como um recurso

http://www.parts-depot.com/parts

Enviar um pedido HTTP GET para o recurso devolve

```
<?xml version="1.0"?>
   <p:Parts xmlns:p="http://www.parts-depot.com"
            xmlns:xlink="http://www.w3.org/1999/xlink">
                <Part id="00345" xlink:href="http://www.parts-depot.com/parts/00345"/>
                <Part id="00346" xlink:href="http://www.parts-depot.com/parts/00346"/>
                <Part id="00348" xlink:href="http://www.parts-depot.com/parts/00348"/>
        </p:Parts>
```

EXEMPLO (CONT.)

Tornar uma lista de peças disponível como um recurso

http://www.parts-depot.com/parts?query=hammer

Ou http://www.parts-depot.com/parts/hammer

Enviar um pedido HTTP GET para o recurso devolve

```
<?xml version="1.0"?>
   <p:Parts xmlns:p="http://www.parts-depot.com"
            xmlns:xlink="http://www.w3.org/1999/xlink">
                <Part id="00345" xlink:href="http://www.parts-depot.com/parts/00345"/>
        </p:Parts>
```

EXEMPLO (CONT.)

Obter detalhes sobre uma peça

http://www.parts-depot.com/parts/00345?flavor=xml

Enviar um pedido HTTP GET para o recurso devolve

```
<?xml version="1.0"?>
    <p:Part xmlns:p="http://www.parts-depot.com"
            xmlns:xlink="http://www.w3.org/1999/xlink">
          <Part-ID>00345</Part-ID>
          <Name>Widget-A</Name>
          <Description>This part is used within the frap assembly</Description>
          <Specification xlink:href="http://www.parts-depot.com/parts/00345/
specification"/>
          <UnitCost currency="USD">0.10</UnitCost>
          <Quantity>10</Quantity>
    </p:Part>
```

EXEMPLO (CONT.)

Submeter uma ordem de compra

Cliente deve saber ou obter informação de como representar o pedido num documento

A seguir efectua um HTTP POST http://www.parts-depot.com/order

```
<?xml version="1.0"?>
    <p:Order xmlns:p="http://www.parts-depot.com"
            xmlns:xlink="http://www.w3.org/1999/xlink">
          <Part-ID>00345</Part-ID>
                  <UnitCost currency="USD">0.10</UnitCost>
          <Quantity>10</Quantity>
    </p:Part>
```

OUTRO EXEMPLO: SISTEMA SOBRE UTILIZADORES

(baseado em documento REST na wikipedia)

Ideia: sistema que mantém informação sobre utilizadores

RPCs:

- getUser()
- addUser()
- removeUser()
- updateUser()
- listUsers()
- findUser()

REST:

http://example.com/users/{user}
(GET, POST, DELETE, PUT)

http://example.com/users/
http://example.com/find/Nuno+Preguica

REST vs. RPCs/Web Services

Nos sistemas de RPCs/Web Services a ênfase é nas operações que podem ser invocadas

Nos sistemas REST, a ênfase é nos recusos, na sua representação e em como estes são afectados por um conjunto de métodos standard

CODIFICAÇÃO DOS DADOS: XML VS. JSON

A informação transmitida nos pedidos e nas respostas é codificada tipicamente em:

- XML
- Json

CODIFICAÇÃO DOS DADOS: XML VS. JSON

```
<Person firstName='John'
lastName='Smith' age='25'>
  <Address streetAddress='21 2nd
Street' city='New York' state='NY'
postalCode='10021' />
  <PhoneNumbers home='212
555-1234' fax='646 555-4567' />
</Person>
```

(Example from wikipedia)

```
{  "Person”:  {
     "firstName": "John",
     "lastName": "Smith",
     "age": 25,
     "Address”:  {
        "streetAddress":"21 2nd Street",
        "city":"New York",
        "state":"NY",
        "postalCode":"10021"
     },
     "PhoneNumbers”: {
        "home":"212 555-1234",
        "fax":"646 555-4567"
     }
   }
}
```

Codificação dos dados: XML vs. Json

A informação transmitida nos pedidos e nas respostas é codificada tipicamente em:

- XML
- Json

Json tem ganho popularidade porque permite manipulação simples em Javascript

REST: PROTOCOLOS

Baseado em protocolos standard

Comunicação

- HTTP (tipicamente)

Identifcação de recursos

- URL/URI

Representação dos recursos

- XML, JSon

REST: NOTAS SOLTAS

Tem ganho popularidade

- Muitos serviços web disponibilizados segundo este modelo
- Grandes vantagens: simplicidade e desempenho

Serviços disponibilizados num modelo REST nem sempre aderem completamente aos princípios REST

- Difícil disponibilizar número elevado de métodos

  http://api.flickr.com/services/rest/?
  method=flickr.photos.search&appid=sdfjkshf

REST: SUPORTE JAVA

Definido em JAX-RS (JSR 311)

Suporte linguístico baseado na utilização de anotações

- Permite definir que um dado URL leva à execução dum dado método
- Permite definir o modo de codificação da resposta
  - XML – usando mecanismo standard de codificação de cobjectos java em XML fornecido pelo JAXB
  - JSON – mecanismo leve de codificação de tipos (RFC 4627)

REST: SUPORTE JAVA (EXEMPLO)

```
@Path("/customerservice/")
@ProduceMime("application/xml")
public class CustomerService {
   @GET
   public Customers getCustomers() {   ......   }

   @GET
   @Path("/customers/{id}")
   @Produces("application/json")
   public Customer getCustomer(@PathParam("id") String id) {......   }

   @PUT
   @Path("/customers/{id}")
   @Consumes("application/xml")
   public Response updateCustomer(@PathParam("id") Long id, Customer customer) {   ......   }

   @POST
   @Path("/customers")
   public Response addCustomer(Customer customer) {   ......   }

   @DELETE
   @Path("/customers/{id}/")
   public Response deleteCustomer(@PathParam("id") String id) {   ......   }

   @Path("/orders/{orderId}/")
   public Order getOrder(@PathParam("orderId") String orderId) {   ......   }
}
```

WEB SERVICES OU REST?

Questão em aberto

REST:

- Mais simples
- Mais eficiente
- Complicado implementar serviços complexos

Web services

- Mais complexo
- Grande suporte da indústria

E.g.:

- http://www.oreillynet.com/pub/wlg/3005

AJAX: Asynchronous JavaScript with XML

Ideias chave:

- Apresentação baseada em standards: XHTML e CSS;
- Apresentação e interação dinâmica usando o Document Object Model;
- Troca e manipulação de dados usando XML e XSLT;
- Obtenção assíncrona de dados usando XMLHttpRequest;
- Utilização de JavaScript na implementação destes princípios.

Objectivo: permitir ao cliente duma aplicação, a executar num browser, apresentar uma interface *rica* e interactiva

- Riqueza: Muitos widgets disponíveis
- Interactividade
  - Possibilidade de modificar a página actual: modificando a árvore DOM
  - Possibilidade de enviar/receber pedidos sem se bloquear: operações XMLHttpRequest não bloqueante

AJAX VS. WEB CLÁSSICO

Web:

- Utilizador clica em links ou forms
- Servidor obtém dados dum form e computa nova página
- Browser apresenta nova página completa

Ajax:

- Acção do utilizador desencadeia modificação rápida da interface
- E.g. Gmail, Google Maps, etc.

AJAX (CONT.)

- Baseado em Javascript
  - Javascript é uma linguagem de scripting suportada pela generalidade dos browsers
  - Algumas características:
    - Não é Java !!!
    - Não tipada
    - Sem threads
    - Suporte para eventos. Porquê?
    - XMLHttpRequest permite efectuar pedido HTTP, de forma assíncrona
- Desenvolvimento
  - Usando frameworks Ajax
    - E.g. Prototype, Dojo, Script.aculo.us, GWT

GWT: Google Web Toolkit

Permite desenvolver aplicações Ajax de forma simples

Desenvolvimento em Java

- Compilador transforma Java em Javascript

E.g. Gmail, Google Maps, etc.

Suporte para invocação remota de procedimentos (semelhante ao RMI), mas suportando RPCs assíncronos

- Mecanismo de serialização própria, permite serializar grafos de objectos

GWT: INTERFACE COM MÉTODO ASSÍNCRONO

```
public interface MyService extends RemoteService {
        Shape[] getShapes(String databaseName) throws ShapeException;
}
public interface MyServiceAsync {
   void getShapes(String databaseName, AsyncCallback callback);
}

public class MyServiceImpl extends RemoteServiceServlet implements MyService {
        Shape[] getShapes(String databaseName) throws ShapeException {
                ...
        }
}
```

GWT: INTERFACE COM MÉTODO ASSÍNCRONO

```
service.getShapes(dbName, new AsyncCallback() {
   public void onSuccess(Object result) {
     Shape[] shapes = (Shape[]) result;
     controller.processShapes(shapes);
   }

   public void onFailure(Throwable caught) {
     try {
       throw caught;
     } catch (IncompatibleRemoteServiceException e) {
       // this client is not compatible with the server; cleanup and refresh the browser
     } catch (InvocationException e) {
       // the call didn't complete cleanly
     } catch (ShapeException e) {
       // one of the 'throws' from the original method
             } catch (Throwable e) {
       // last resort -- a very unexpected exception
     }
   }
 });
```

Para saber mais

G. Coulouris, J. Dollimore and T. Kindberg, Gordon Blair, Distributed Systems - Concepts and Design, Addison-Wesley, 5th Edition

Web services – capítulo 9.1-9.4

REST: http://en.wikipedia.org/wiki/Representational_State_Transfer

GWT: http://code.google.com/webtoolkit/