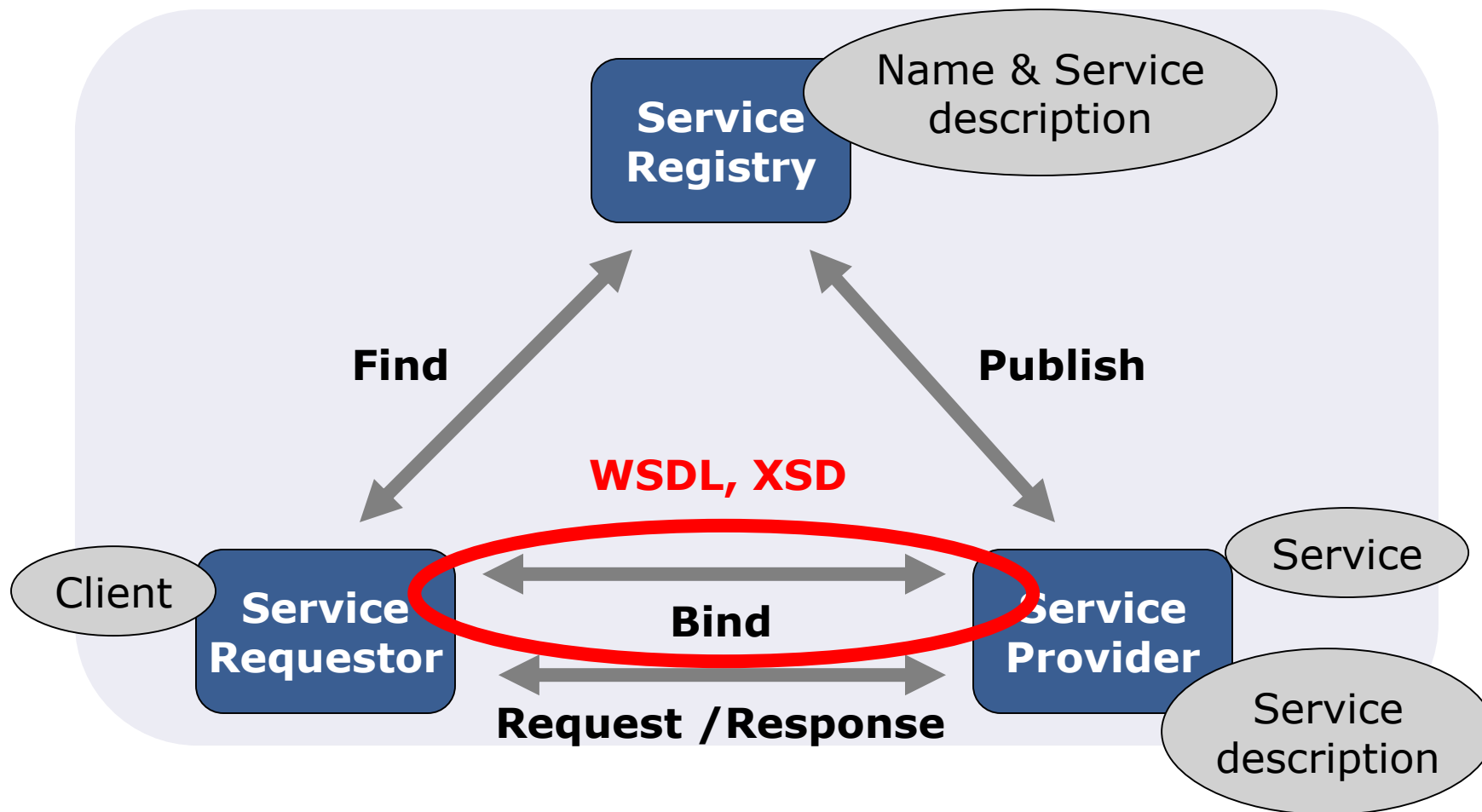


Web Services

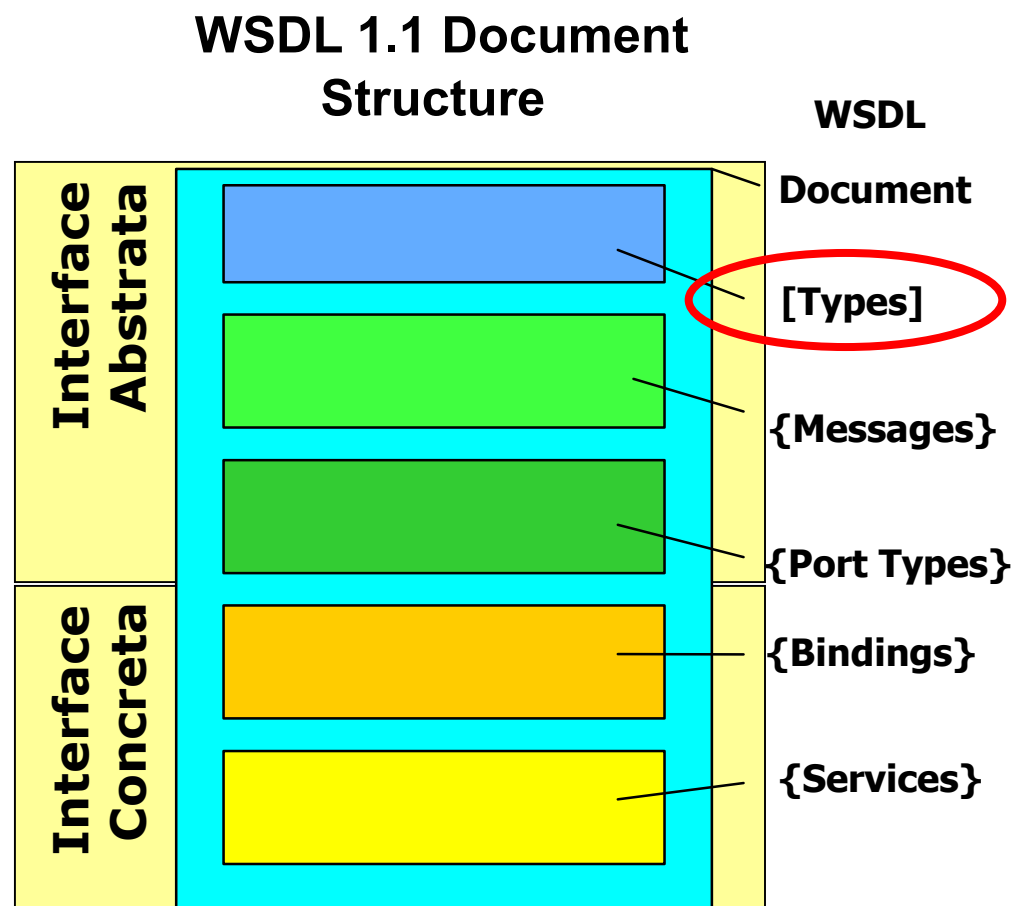
(continuação)

Modelo dos Web Services (arquitetura básica)



WSDL

Web Services Description Language



XML (eXtensible Markup Language)

Resolver a heterogeneidade
na comunicação e nos dados
de forma universal

Arquitetura de Integração da Informação

- Em muitos casos a integração tem de resolver o problema da troca de informação com múltiplas representações
- A solução mais simples para este problema é **representar os dados num formato canónico** que todos saibam utilizar
- Os dados originais de cada sistema têm de ser mapeados no formato canónico mas depois podem ser usados por todos
- Se o formato canónico tiver uma representação explícita as mensagens deixam de ter de ser construídas pela ordem estrita de invocação
 - Pode verificar-se se estão de acordo com o esperado
 - Podem ser analisadas por ferramentas independentes do RPC
- Esta é a razão da grande importância do XML

eXtensible Markup Language (XML)

- Deriva de uma linguagem muito mais antiga para definição do formato de impressão de documentos, o SGML
 - O SGML já tinha sido a tecnologia de base no desenvolvimento do HTML
- Essencialmente o XML permite através de etiquetas (*tags*) associar a descrição do formato aos dados de um documento
- Toda a descrição é textual
 - Resolve muito dos problemas de portabilidade
- O XML é uma proposta do W3C
 - Percepcionado como o substituto dos formatos anteriores de representação de dados como o EDI

Importância do XML

O XML trabalha sobre documentos

Os documentos podem ser usados para múltiplos fins:

- Representar/apresentar a informação numa forma visível: em papel ou transformada para HTML, PDF
- Comunicação de dados entre plataformas heterogéneas
- Para armazenamento da informação em ficheiros ou em bases de dados

Sintaxe XML

- Regras de construção simples, mas rígidas:
 - Existe um elemento raiz único
 - Todos os elementos têm que fechar
 - Fecham pela ordem inversa em que abrem
 - Os nomes são sensíveis à capitalização das letras (*case-sensitive*)
- Um documento que respeita as regras de construção diz-se:
 - Bem-formado (*well-formed*)
- Adicionalmente, pode-se definir uma gramática para documentos XML
 - Que sequências de elementos são válidas?
 - Que tipos de dados – string, int, date – contém o elemento?
- Um documento que respeita uma gramática diz-se:
 - Válido (*valid*)

Exemplo de uma estrutura de dados em XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- este documento define uma lista de empregados -->
<employeeList xmlns="http://www.company.org/Emp">
  <employee type="contract">
    <employee_id>75868</employee_id>
    <name>
      <first_name>John</first_name>
      <last_name>Doe</last_name>
    </name>
    <extn>27304</extn>
    <dept>1104332089</dept>
    <email>john.doe@flute.com</email>
  </employee>
</employeeList>
```

employeeList

Benefícios do XML

- Tecnologia não proprietária
- Independente das plataformas: o formato é texto
- Compatível com o HTTP
 - O XML tem uma sintaxe mais restritiva que o HTML
 - Pode ser usado como o HTML, pelo que passa pelas *firewalls*
- Internacional
 - Formato texto que usa UTF-8 ou UTF-16 para representar os caracteres
 - **Unicode** Transformation Format
- Extensível
 - Novas *tags* podem ser adicionadas por necessidade de extensão
 - Para evitar duplicações existem *namespaces* a que um documento pode ser associado

Benefícios do XML (II)

- A interpretação das mensagens depende de *tags* e não da posição do campo na mensagem
- Permite a transformação automática dos dados
 - As transformações são especificadas usando XSLT (XML Stylesheet Language Transformation)
- Auto definida
 - A estrutura de um documento XML tem uma meta-descrição:
 - Document Type Definition (DTD) (originalmente)
 - **XML Schema Definition (XSD)** (atualmente)
- Ferramentas genéricas
 - Ferramentas *Open-Source* em Java e ferramentas já existentes para SGML

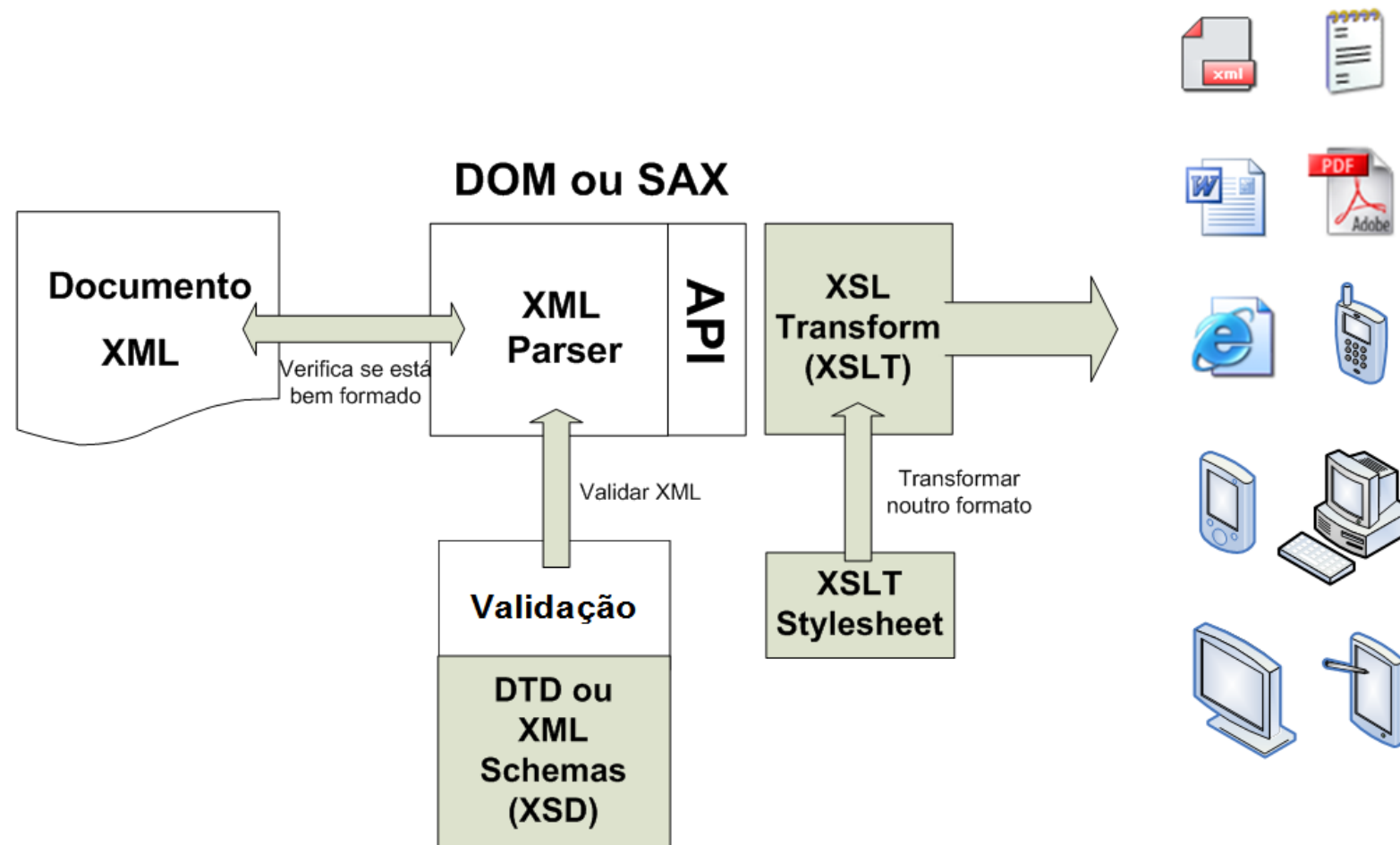
Tecnologia XML

- **Namespaces**
 - Permitem a unicidade de nomes e resolver colisões.
Os espaços de nomes são normalmente associados a um domínio de aplicação
 - Um nome torna-se único por associação a um URI
- **XPath**
 - Define um mecanismo de acesso a elementos dentro de um documento XML
- **XLink**
 - Mecanismo para indexar outros documentos semelhantes aos *hyperlinks* do HTML, mas mais robusto
- **XSL**
 - XML Style Sheet - documentos XML que definem transformações de documentos em XML
- **SAX - Simple API for XML**
 - O SAX é um *parser* que invoca funções à medida que encontra elementos no documento
- **DOM - Document Object Model**
 - O DOM define os métodos para aceder a um documento XML.
Pressupõe que o documento é carregado em memória e
que se acede aos diferentes elementos através de uma API

eXtensible Stylesheet Language (XSL)

- Permite:
 - Transformar XML em HTML
 - Transformar XML noutro XML
 - Filtrar e ordenar dados XML
 - Apresentar o mesmo documento de formas diferentes dependendo do dispositivo de destino (ecrã, impressão, telefone móvel, etc.)

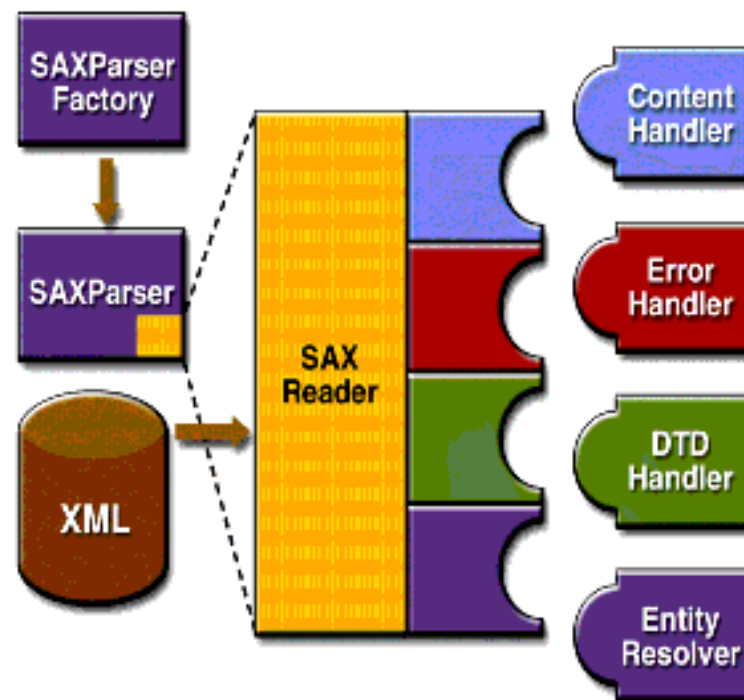
Tecnologia XML



JAX-P – Java API for XML Processing:

SAX – Simple API for XML

- Processamento em série
- Baseado no tratamento de eventos



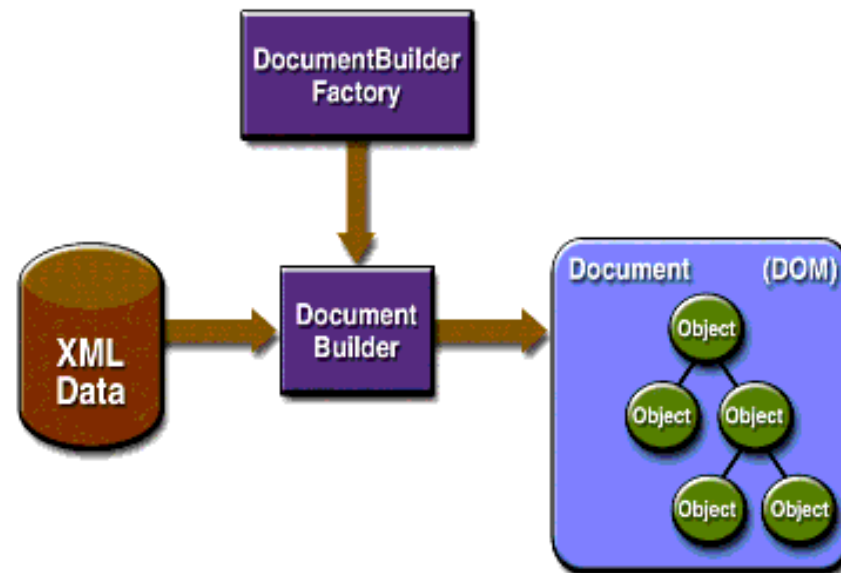
Quando usar SAX

- Quando pretendemos processar informação em série (*data stream*)
- A informação a processar não precisa de manter estado
 - Ex. importação/exportação de dados

JAX-P – Java API for XML Processing:

DOM – Document Object Model

- Manipulação de árvore em memória



Esquema do Documento

- A especificação de documentos era inicialmente efectuada em **DTD** (*Document Type Definition*)
- Em 2001 foi proposto o **XSD** (*XML Schema Definition*)
 - Estende as capacidades do DTD
 - Tipos de dados
 - Principal vantagem: também usa sintaxe XML
 - Um *parser* usando um DTD ou XSD pode verificar se o documento está sintaticamente correto e se está conforme com um determinado tipo
- A meta-descrição pode ser incluída no documento, mas a utilização mais interessante é poder aceder-lhe através de um URI partilhando-o entre aplicações

Comparação DTD e XML Schema

- DTD era anterior ao XML pelo que não suporta bem alguns aspetos da norma como por exemplo os *namespaces*
- O DTD tinha por objetivo descrever documentos legíveis por humanos e não documentos para representar dados, por conseguinte faltam-lhe alguns construções para exprimir diversas restrições simples,
 - Ex.: idade só pode ter um valor não negativo entre 0 e 150
- A sintaxe do DTD não é a do XML o que torna mais complexas as ferramentas

DTD Simples para o documento employeeList

```
<!DOCTYPE employeeList [
```

```
<!ELEMENT employeeList (employee*)>
```

```
<!ELEMENT employee (employee_id, name, extn, dept, email)>
```

```
<!ATTLIST employee type (perm|contract) #REQUIRED>
```

```
<!ELEMENT employee_id (#PCDATA)>
```

```
<!ELEMENT NAME (first_name, last_name)>
```

```
<!ELEMENT first_name (#PCDATA)>
```

```
<!ELEMENT last_name (#PCDATA)>
```

```
<!ELEMENT extn (#PCDATA)>
```

```
<!ELEMENT dept (#PCDATA)>
```

```
<!ELEMENT email (#PCDATA)>
```


Element *employeeList* consists of zero or more *employee* elements.

NOTE: "*" = zero or more, "?" = zero or one, "+" = one or more

Element *employee* contains elements *employee_id*, *name*, *dept*, and *email*.

Attribute *type* for element *employee* is required and must have either of these two values: *contract* or *perm*.

Sintaxe próxima das expressões regulares

Element *dept.* may contain only text (PCData).

XML Schema para o documento employeeList

```

</xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.flute.com"
  xmlns="http://www.flute.com">

  <xsd:element name="employeeList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="employee" minOccurs="1"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="employee"
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="employee_id" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="name" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="extn" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="dept" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="email" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="employeeAttribute"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="name"
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="first_name" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="last_name" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

```

Namespace declaration

<!ELEMENT employeeList (employee*)>

<!ELEMENT employee (employee_id, name, extn, dept, email)>

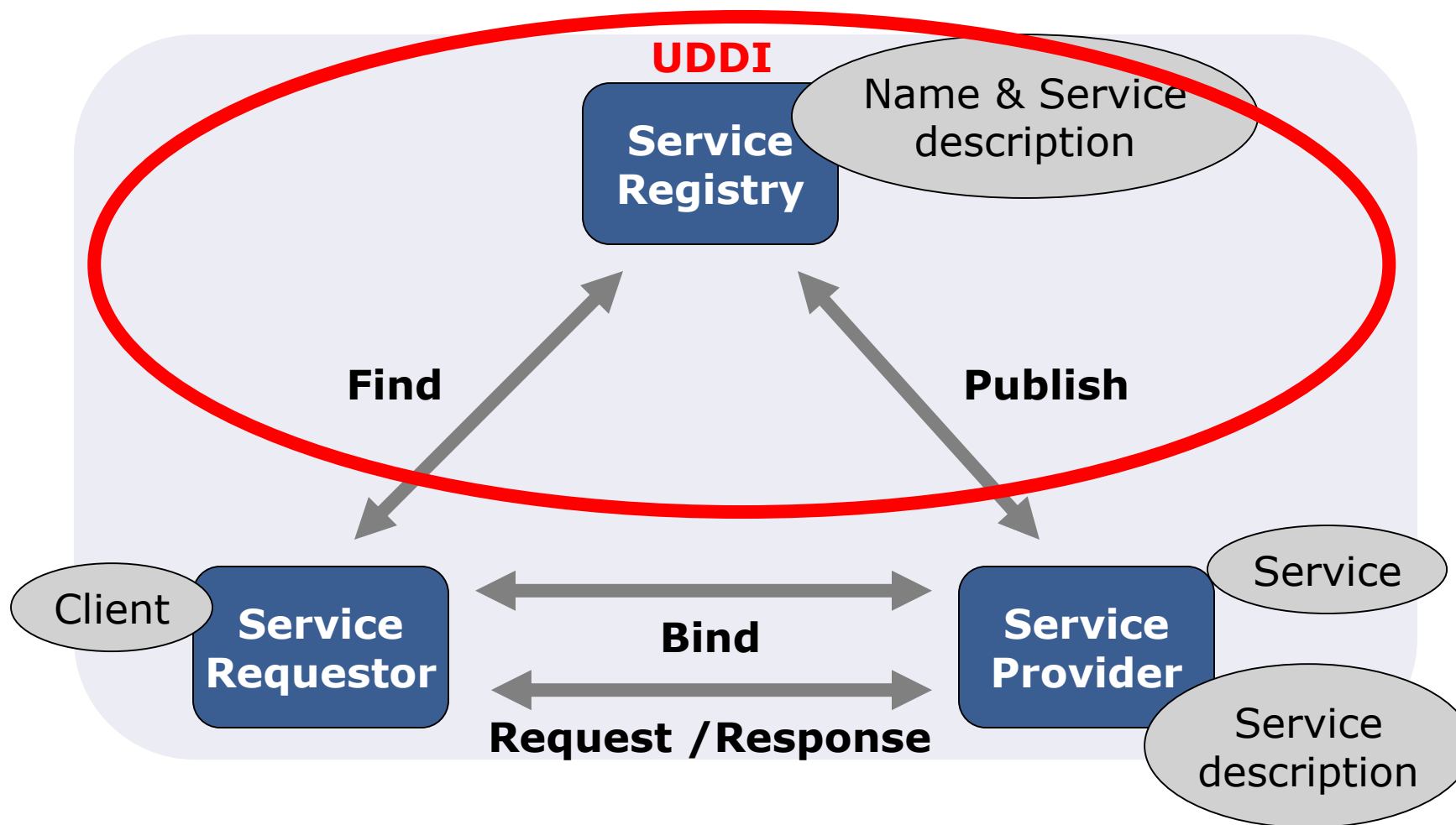
<!ELEMENT name (first_name, last_name)>

<pre> <xsd:element name="employee_id"> <xsd:simpleType> <xsd:restriction base="xsd:int"> <xsd:minInclusive value="1"/> <xsd:maxInclusive value="100000"/> </xsd:restriction> </xsd:simpleType> </xsd:element> </pre>	<p>← employee:_id must be an integer between 1 and 100,000.</p>
<pre> <xsd:element name="first_name" type="xsd:string"/> <xsd:element name="last_name" type="xsd:string"/> <xsd:element name="email" type="xsd:string"/> <xsd:element name="dept" > <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:pattern value="[0-9]{3}-[0-9]{3}-[0-9]{4}"/> </xsd:restriction> </xsd:simpleType> </xsd:element> </pre>	<p>← <!ELEMENT first_name (PCDATA)></p> <p>← <!ELEMENT email (PCDATA)></p> <p>← Department must be specified in the form: "999-999-9999".</p>
<pre> <xsd:element name="extn"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:pattern value="[0-9]{5}"/> </xsd:restriction> </xsd:simpleType> </xsd:element> </pre>	<p>← Extension must be specified in the form "99999" (String consisting of five digits).</p>
<pre> <xsd:attributeGroup name="employeeAttribute"> <xsd:attribute name="type" use="required"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:enumerator value="contract"/> </xsd:restriction> </xsd:simpleType> </xsd:attribute> </xsd:attributeGroup> </pre>	<p>← <!ATTLIST employee type (permicontract) #REQUIRED></p>

XML Schema

- Define
 - Elementos obrigatórios e proibidos
 - Estrutura hierárquica dos elementos
 - Atributos necessários ou opcionais dos elementos e gama de valores permitidos
 - Referências de parte de um documento a outros elementos do documento

Modelo dos Web Services (arquitetura básica)



Discovery Stack

- Protocolo de ligação ou *binding* entre o cliente e o servidor.
- Os fornecedores dos serviços publicam a respetiva interface
- O protocolo de inspeção permite verificar se um dado serviço existe baseado na sua identificação
- O UDDI responde às questões
 - Onde é que o Web Service está localizado?
 - Qual o processo de negócio que o serviço disponibiliza?
- O UDDI permite encontrar o serviço baseado na sua definição
 - *Capability lookup*

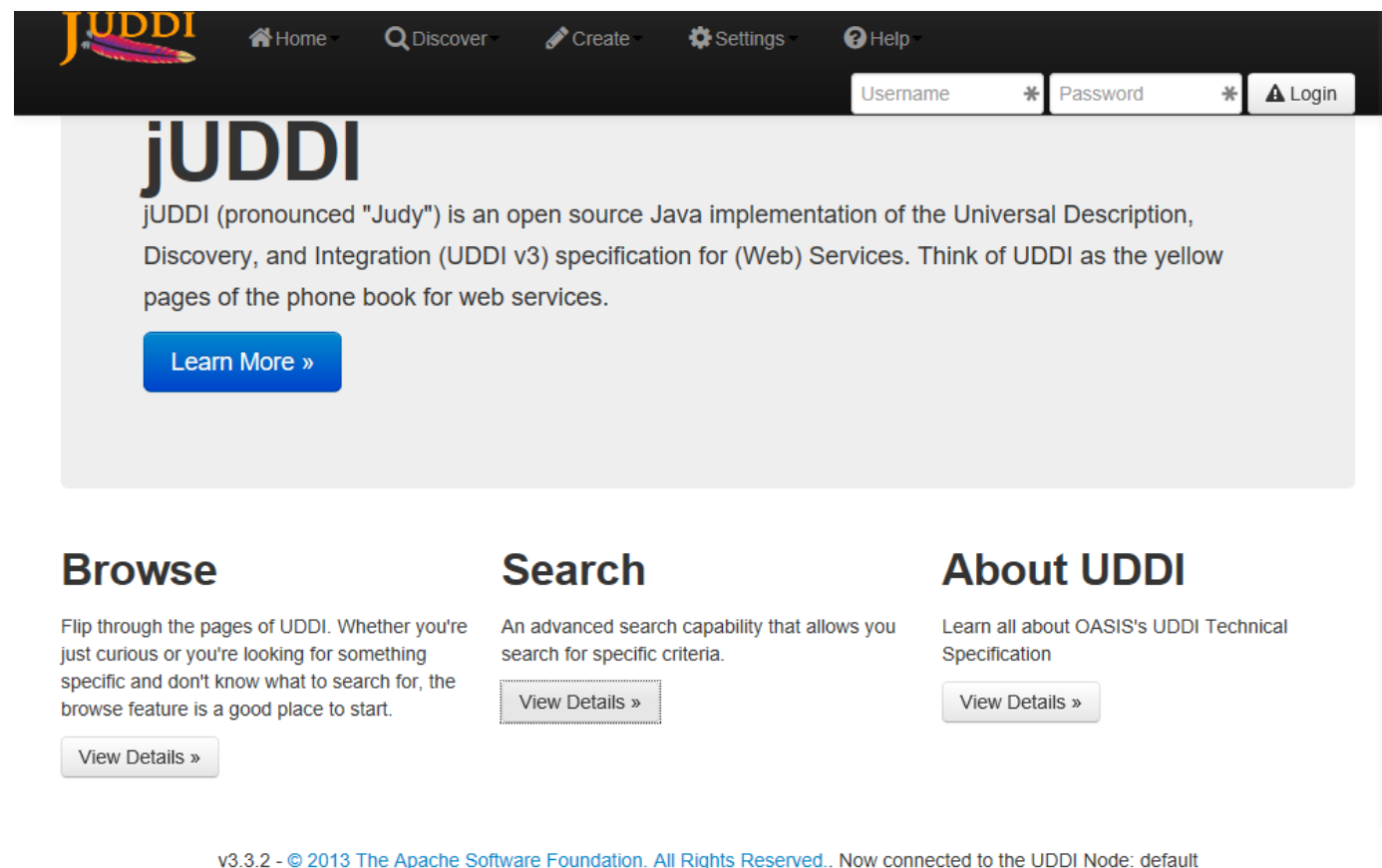
Universal Description Discovery & Integration (UDDI)

- Definição de um conjunto de serviços que suportam a descrição e a localização de:
 - Entidades que disponibilizam Web Services (empresas, organizações)
 - Os Web Services disponibilizados
 - As interfaces que devem ser utilizadas para aceder aos Web Services
- Baseada em standards Web: HTTP, XML, XSD, WSDL, SOAP

Informação representada na UDDI

- A UDDI permite pesquisar informação muito variada sobre os Web Services.
Ex:
 - Procurar Web Services que obedecem a uma determinada interface abstracta
 - Procurar Web Services que estejam classificados de acordo com um esquema conhecido de classificação
 - Determinar os protocolos de transporte e segurança suportados por um determinado Web Service
 - Procurar Web Services classificados com uma palavra-chave
- O acesso é feito via as API definidas
 - Mas os operadores também disponibilizam *sites* para acesso via web

jUDDI Graphical User Interface



The screenshot shows the jUDDI web application interface. At the top is a dark navigation bar with the jUDDI logo, links for Home, Discover, Create, Settings, and Help, and a login section with Username and Password fields and a Login button. Below the navigation bar is a large white box containing the jUDDI logo and a description: "jUDDI (pronounced 'Judy') is an open source Java implementation of the Universal Description, Discovery, and Integration (UDDI v3) specification for (Web) Services. Think of UDDI as the yellow pages of the phone book for web services." A blue "Learn More »" button is positioned below the text. Below this box are three columns: "Browse", "Search", and "About UDDI". Each column contains a brief description and a "View Details »" button. At the bottom, a footer line reads: "v3.3.2 - © 2013 The Apache Software Foundation. All Rights Reserved.. Now connected to the UDDI Node: default".

jUDDI

jUDDI (pronounced "Judy") is an open source Java implementation of the Universal Description, Discovery, and Integration (UDDI v3) specification for (Web) Services. Think of UDDI as the yellow pages of the phone book for web services.

[Learn More »](#)

Browse

Flip through the pages of UDDI. Whether you're just curious or you're looking for something specific and don't know what to search for, the browse feature is a good place to start.

[View Details »](#)

Search

An advanced search capability that allows you search for specific criteria.

[View Details »](#)

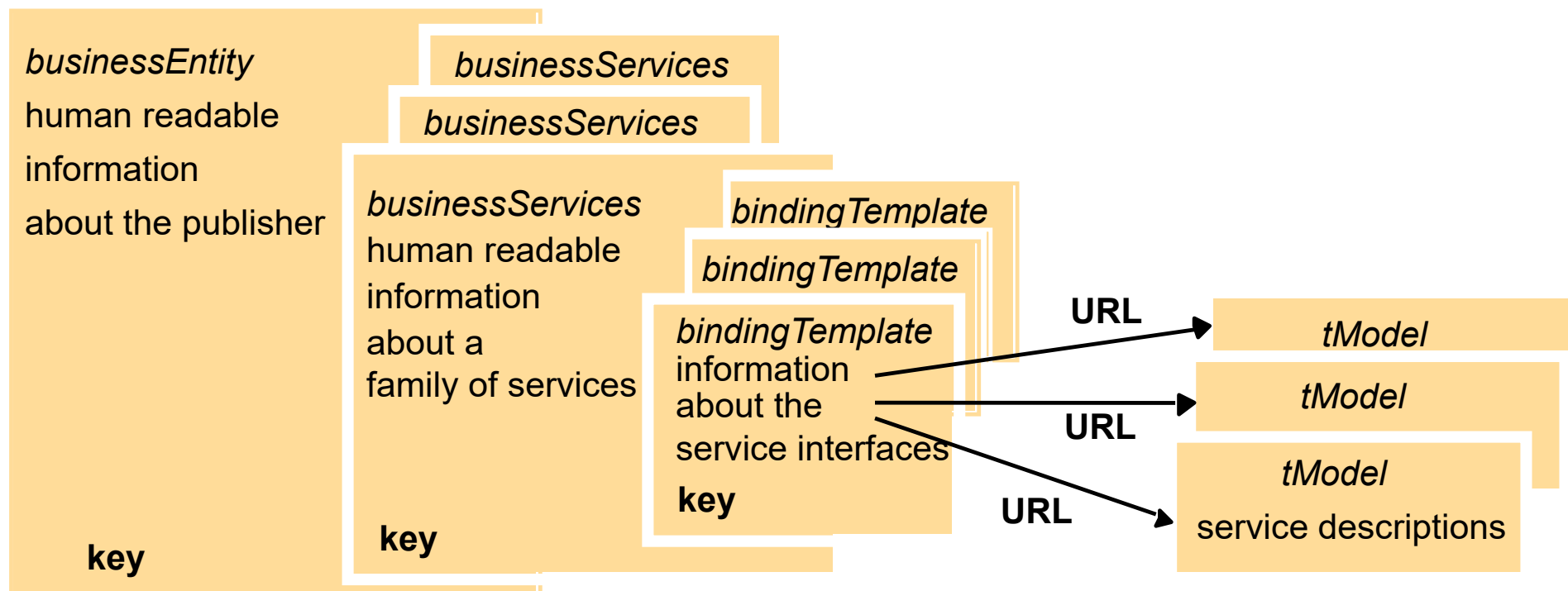
About UDDI

Learn all about OASIS's UDDI Technical Specification

[View Details »](#)

v3.3.2 - © 2013 The Apache Software Foundation. All Rights Reserved.. Now connected to the UDDI Node: default

Modelo estrutural da informação



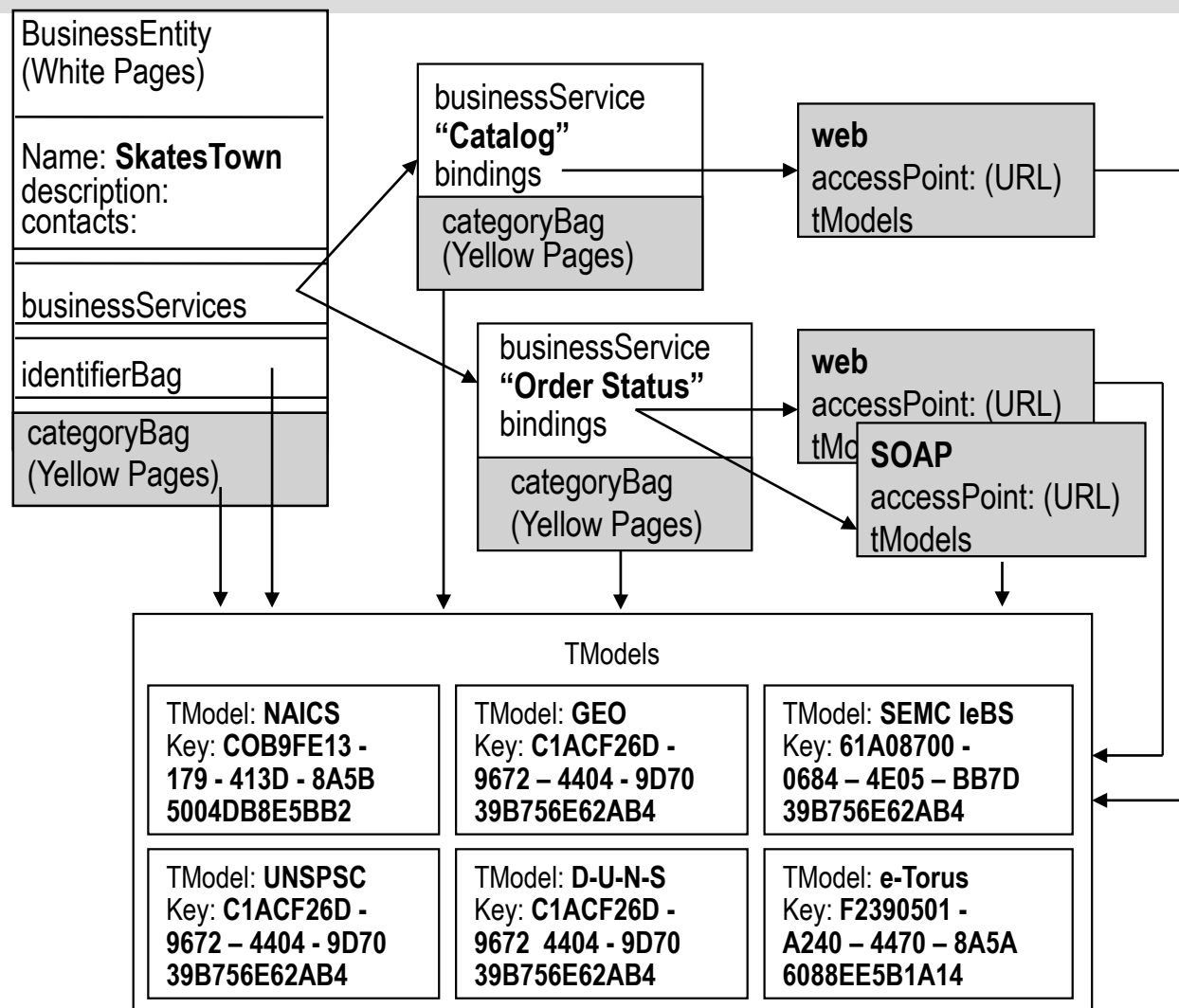
Modelo estrutural da informação

- *businessEntity*: descreve uma empresa ou organização que exporta Web Services
 - A informação encontra-se conceptualmente dividida em:
 - Páginas brancas – informação geral de contacto
 - Páginas amarelas – Classificação do tipo de serviço e localização
 - Páginas verdes – Detalhes sobre a invocação do serviço
- *businessService*: descreve um conjunto de Web Services exportado por uma *businessEntity*
- *bindingTemplate*: descreve a informação técnica necessária para usar um determinado serviço
- *tModel*: descreve o “modelo técnico” de uma entidade reutilizável, como um tipo de Web Service, o *binding* a um protocolo usado por um Web Service, etc.

Taxonomia

- Os serviços registados devem ser categorizados em **taxonomias** que os permitam pesquisar
- Existem várias taxonomias normalizadas, ex.:
 - UN/SPSC – Produtos e serviços – ONU
 - ISO 3166 – Geografias
 - D-U-N-S – Data Universal Numbering System – Dun&Bradstreet
- O UDDI permite que todas as entidades sejam classificadas
- As pesquisas podem usar múltiplas classificações
- Para uso interno das organizações podem ser definidos os seus esquemas de classificação
 - Ex.: qualidade de serviço do fornecedor do Web Service

Exemplo: Modelo estrutural da informação



API UDDI

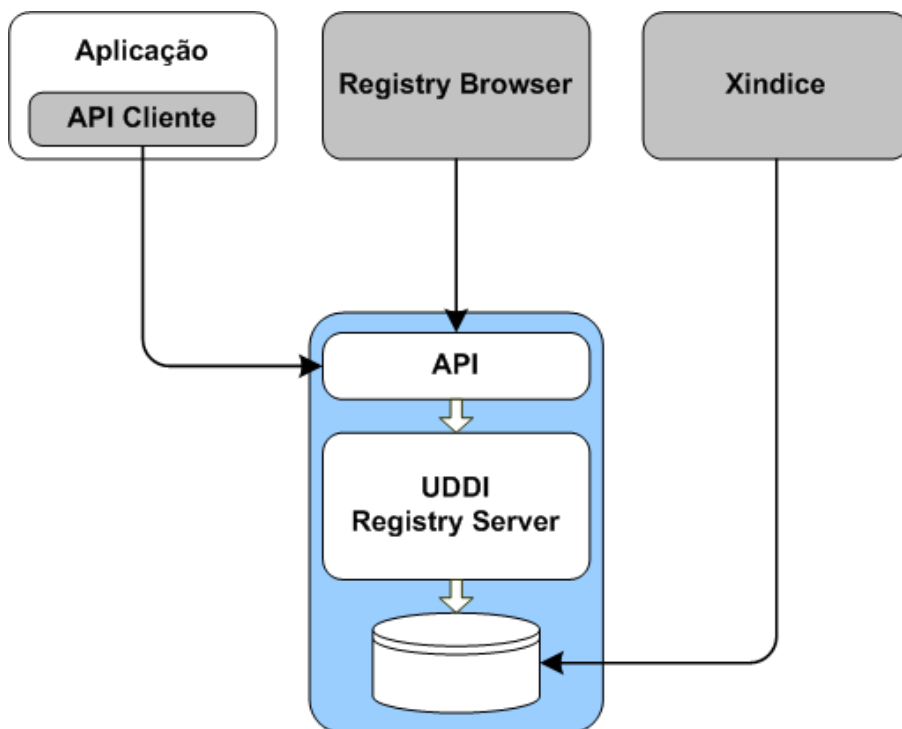
- API de Mensagens para as funções CRUD (*Create, Retrieve, Update, Delete*) definidas pelas UDDI Spec

	<i>Business</i>	<i>Service</i>	<i>Binding</i>	<i>tModel</i>
Save/Update	save_business	save_service	save_binding	save_tModel
Delete	delete_business	delete_service	delete_binding	delete_tModel
Find	find_business	find_service	find_binding	find_tModel
GetDetail	get_businessDetail	get_serviceDetail	get_bindingDetail	get_tModelDetail

Arquitetura UDDI - *Registries*

- Um *Registry* é composto por um ou mais nós UDDI
- Os nós de um *Registry* gerem coletivamente um conjunto bem definido de dados UDDI. Tipicamente, isto é suportado com replicação entre os nós do *Registry*
- A representação física de um *Registry* é deixada à escolha das implementações

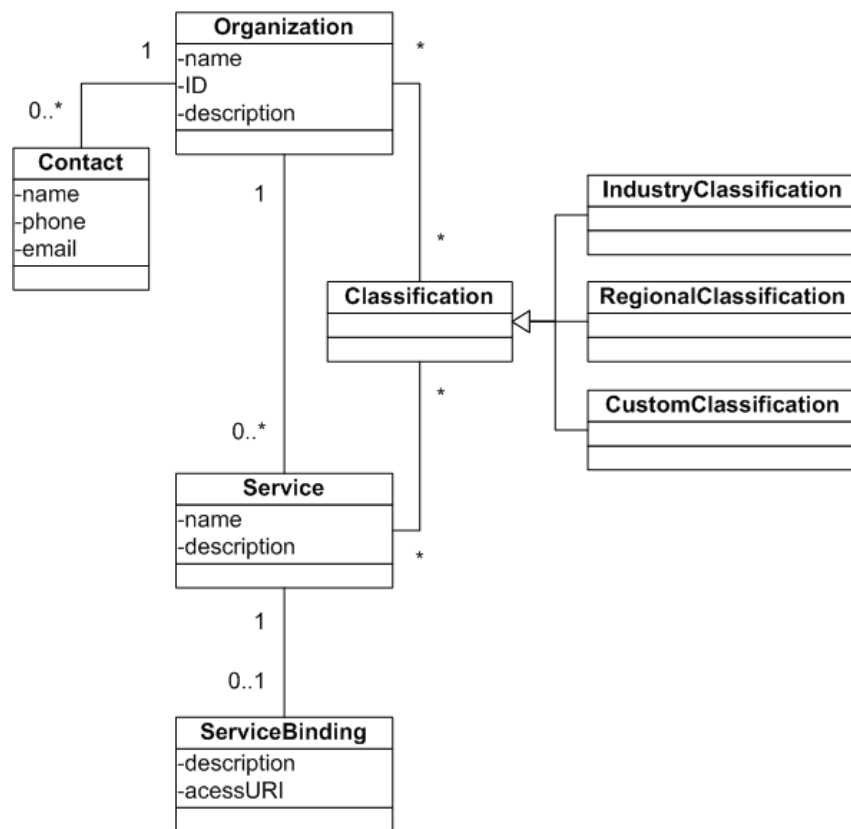
Registry Server privado



- Gere a base de dados com os registos UDDI
- Modos de acesso
 - Aplicações JAX-R
 - *Registry Browser*
 - *XIndice*
- Interfaces
 - API
 - Acesso direto aos registos

JAX-R API

Estruturas de dados



- *Java API for XML Registries*
- O JAX-R tem um modelo de dados muito próximo do UDDI
 - O mapeamento de um para outro é quase direto
 - Só mudam alguns nomes
- Pacote de classes que implementam o modelo de dados:
 - `javax.xml.registry.infomodel`

UDDI Naming

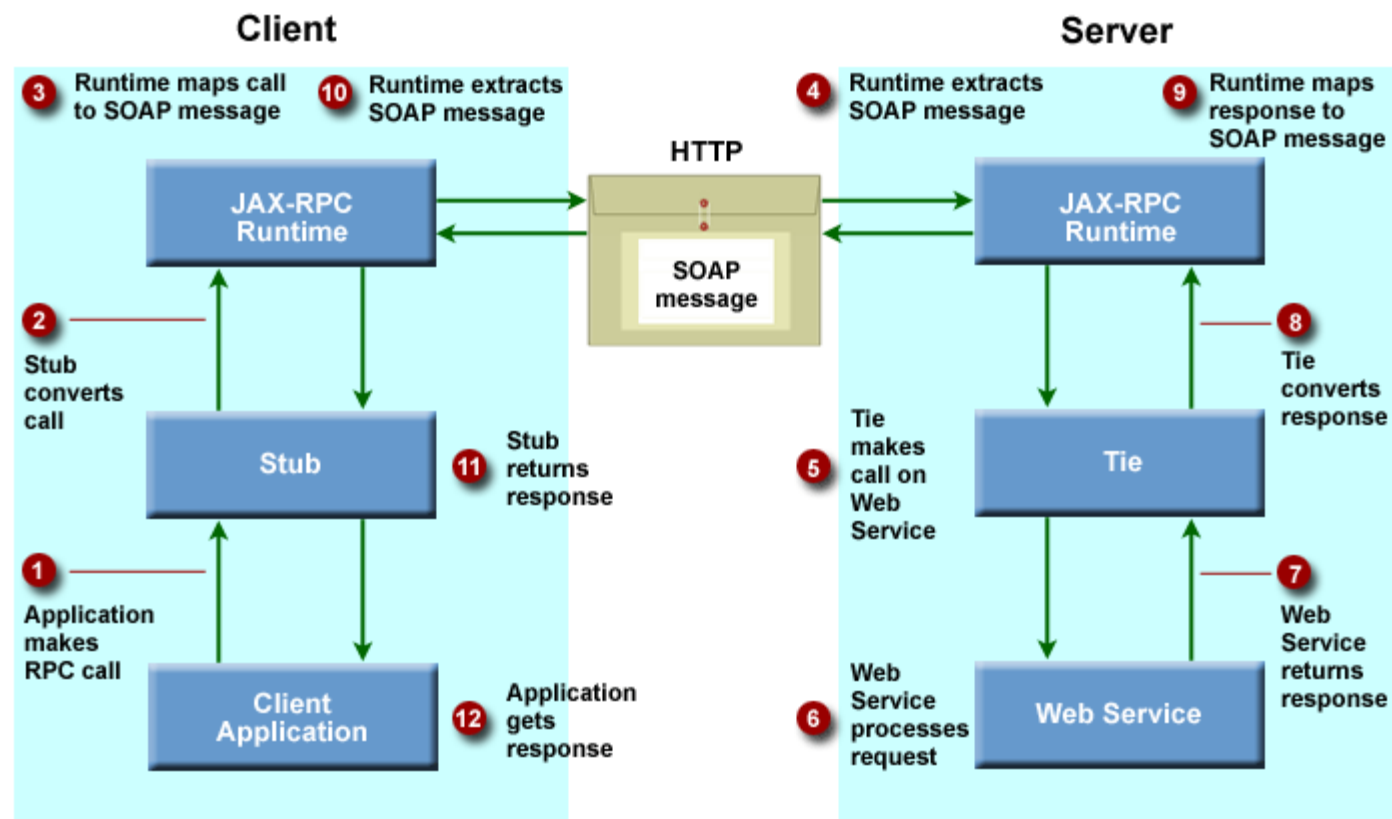
- Biblioteca para simplificar a utilização do UDDI
 - Inspirada no JNDI (RMI) Naming
 - `bind()`, `lookup()`
 - Limitação: apenas suporta relações 1-para-1
- 1 Organização
 - 1 Serviço
 - 1 Endereço
- Pacote:
 - `pt.ulisboa.tecnico.sdis.ws.uddi`
 - Código-fonte disponível, podem modificar no projeto

JAX-WS

(sucessor do JAX-RPC)

Integração dos Web Services com o ambiente Java

JAX-WS - Passos de Execução



JAX-WS

- Server-side
 - Definir o WSDL
 - Gerar os *ties* a partir do WSDL
 - Empacotar aplicação
 - Executar
 - Isoladamente (*stand alone*) ou
 - Instalar (*deploy*) num servidor aplicacional
- Client-side
 - Gerar os *stubs* a partir do WSDL
 - Compilar e executar a aplicação

Cliente com um *static proxy*

```
public static void main(String args[]) {  
    if (args.length != 1) {  
        System.err.println("usage: BillPayClient <cableProvider>");  
        System.exit(1);  
    }  
  
    BillPayService service = new BillPayService();  
    BillPay port = service.getBillPayPort();  
    System.out.print("Server said: ");  
    System.out.println(port.getLastPayment(args[0]));  
}
```

Cliente: invocação dinâmica

- Invocação “semi-dinâmica” (*dynamic proxy*)
 - Classe criada durante a execução a partir do WSDL que se obtém na altura
 - Antes da execução, apenas a interface abstracta do serviço é conhecida (correspondendo a uma interface Java)
- Invocação dinâmica (*dynamic invocation interface, DII*)
 - O cliente em tempo de execução utiliza o WSDL para construir a invocação
 - Antes da execução, nem interface abstracta nem concreta do serviço são conhecidas

Exemplo:

Cliente com *dynamic proxy*

```
public static void main(String[] args) throws Exception{
    String namespace = "http://www.flutebank.com/xml";
    String wsldport = "BillPayPort";
    String wsdlservice = "Billpayservice";
    String wsdllocation="http://127.0.0.1:8080/billpayservice/billpayservice.wsdl";
    URL wsldurl = new URL(wsdllocation);

    ServiceFactory factory = ServiceFactory.newInstance();

    Service service = factory.createService(
        wsldurl, new QName(namespace, wsdlservice));
    //make the call to get the stub corresponding to this service and interface
    BillPay stub = (BillPay) service.getPort(
        new QName(namespace,wsldport), BillPay.class);
    // invoke methods on the service
    double lastpaid= stub.getLastPayment
    ("my cable tv provider");
    System.out.println("Last payment was" + lastpaid);
}
}
```

```
public class DIIClient_WSDL{

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

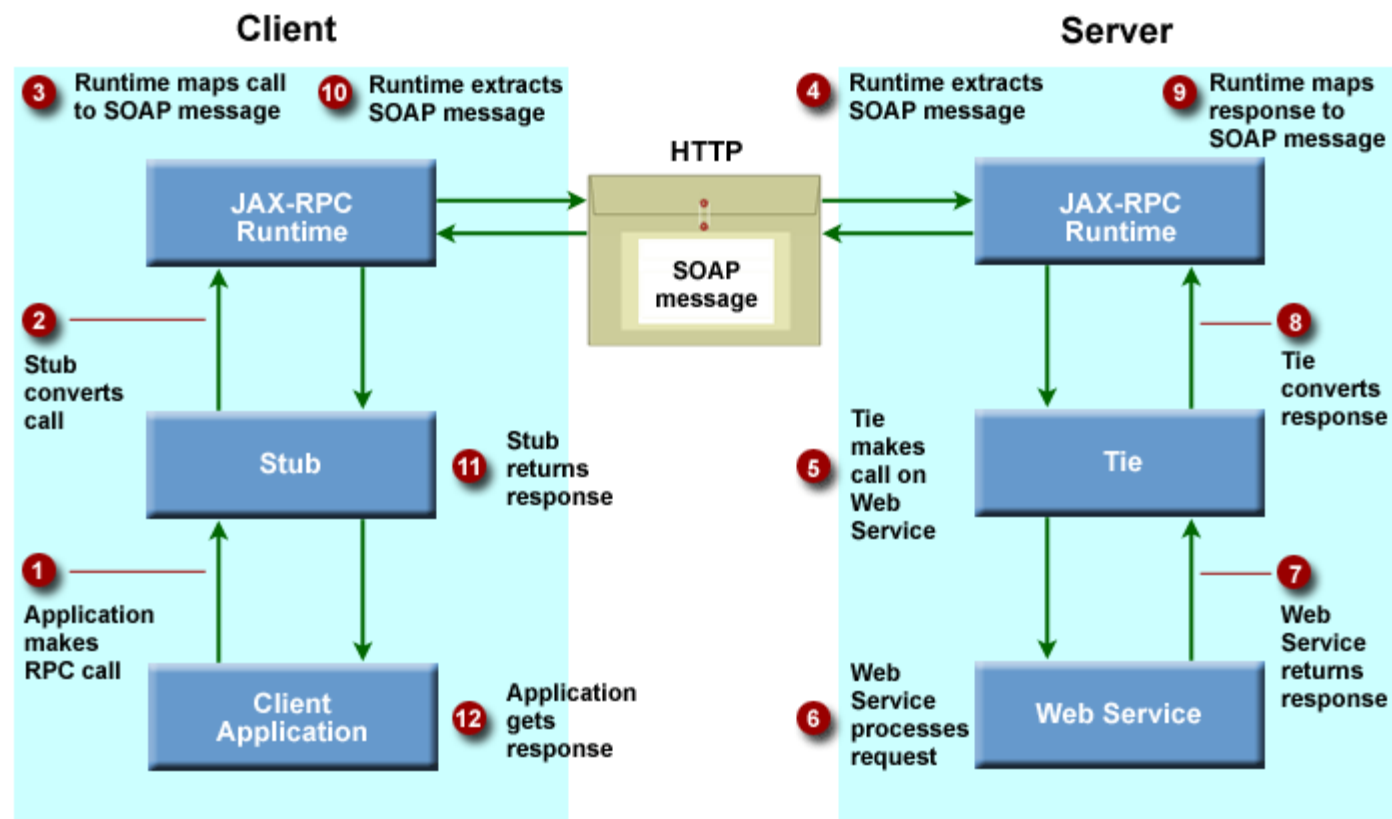
        String wsdllocation = http://127.0.0.1:9090/billpayservice/billpayservice.wsdl";

        String namespace = "http://www.flutebank.com/xml";
        String serviceName = "Billpayservice";
        ServiceFactory factory = ServiceFactory.newInstanceQ;

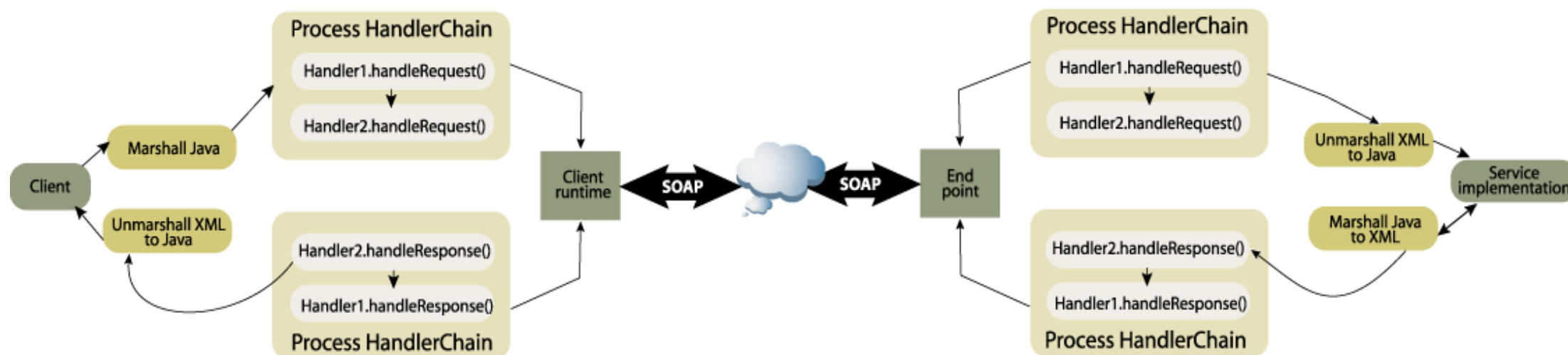
        Service service = (Service) factory.createService(new URL(wsdllocation),
                new QName(namespace, serviceName));
        QName portName = new QName(namespace, "BillPayPort");
        QName operationName = new QName(namespace, "getLastPayment");
        Call call = service.createCall(portName, operationName);
        Object[] params = {"my cable tv provider"};

        Object lastpaid = (Double) call.invoke(params);
        System.out.println("Last payment was" + lastpaid);
    }
}
```

JAX-WS - Passos de Execução



JAX-WS Handlers



- *Handler*
 - Implementa a interface
 - **javax.xml.ws.handler.Handler**
 - Métodos mais relevantes
 - `handleMessage(MessageContext context)`
 - `handleFault(MessageContext context)`

Exemplo de *SOAP Handler*

```
public boolean handleMessage(SOAPMessageContext smc) {  
  
    Boolean isOutbound = (Boolean) smc.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);  
  
    String bound = (isOutbound ? "Outbound" : "Inbound");  
    out.println(bound + " message:");  
  
    SOAPMessage soapMessage = smc.getMessage();  
  
    // Use SAAJ API to manipulate the SOAP Message  
    soapMessage.writeTo(out);  
    out.println("");  
  
}
```

Handler chains: cadeia de processamento

- **Configuração** (uma no cliente, outra no servidor)

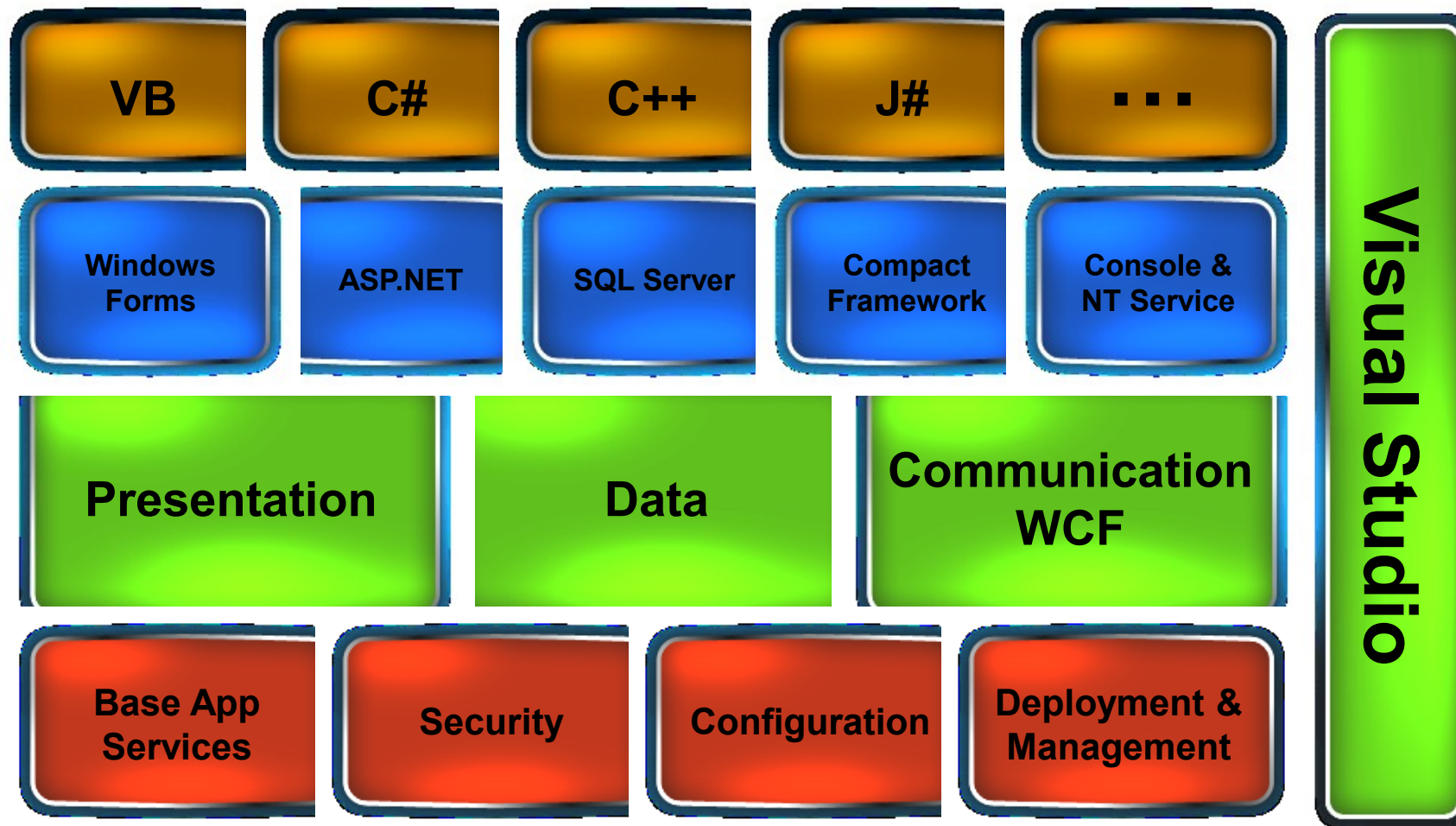
```
...
<jws:handler-chains>
  <jws:handler-chain>
    <jws:handler>
      <jws:handler-class>util.LogHandler</jws:handler-class>
    </jws:handler>
    <jws:handler>
      <jws:handler-class>util.CipherHandler</jws:handler-class>
    </jws:handler>
  </jws:handler-chain>
</jws:handler-chains>
...
```

- Esta configuração especifica que, para cada mensagem SOAP que é recebida ou enviada pelo Web Service, os *handlers* são invocados na seguinte ordem:
 - À saída (*outbound*): Log, Cipher
 - À chegada (*inbound*): Cipher, Log

Windows Communication Foundation (WCF)

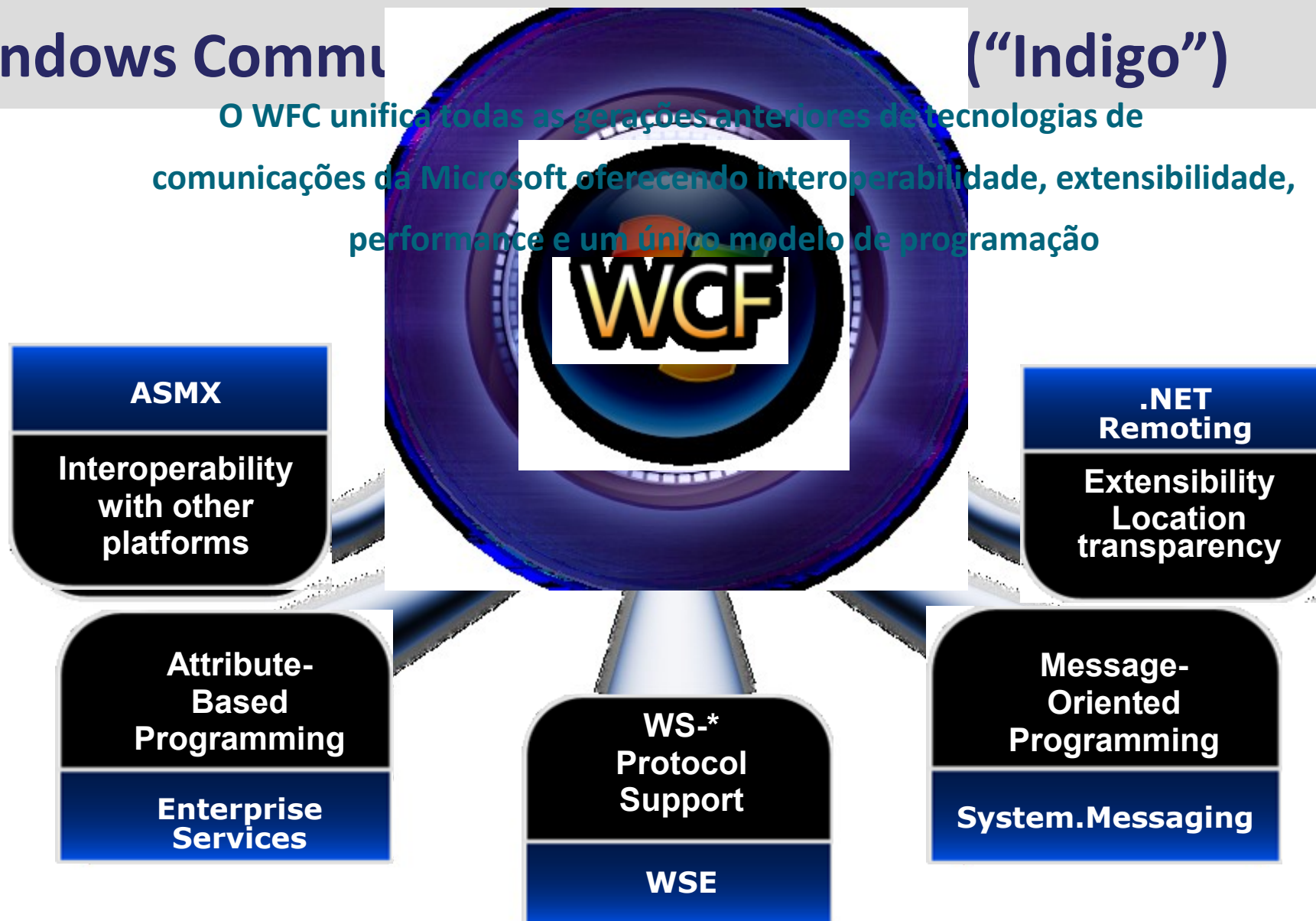
Integração dos *Web Services* com o ambiente .Net 3.0

Microsoft .NET Framework



Windows Communication Foundation (“Indigo”)

O WCF unifica todas as gerações anteriores de tecnologias de comunicações da Microsoft oferecendo interoperabilidade, extensibilidade, performance e um único modelo de programação



Windows Communication Foundation (WCF)

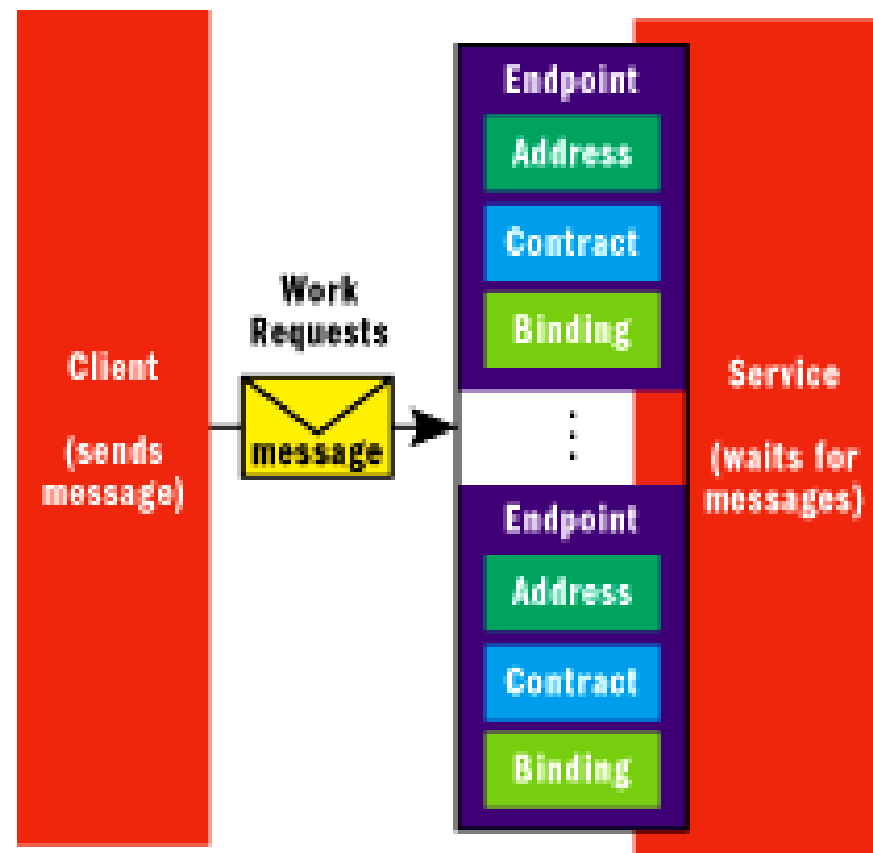
- Unifica de forma consistente o anterior suporte na plataforma .Net a:
 - Web Services
 - Invocação remota de objectos (.Net Remoting)
 - Transações distribuídas (MS Transaction Server)
 - Message Queues (MSMQ)
- Assegurando execução protegida (*managed*) desse suporte
- Através de modelo de programação orientado a serviços

(As Muitas) Semelhanças de Base com JAX-WS

- Baseado em SOAP para comunicação entre processos
- API simples, alivia programador da complexidade de SOAP, WSDL, etc.
 - No servidor, o programador especifica os procedimentos remotos definindo uma interface C# (ou Java, ou outra...) e criando uma classe que a implemente
 - No cliente, o programador cria um objecto proxy para utilizar os serviços remotos

Definição dos Serviços: *Services* e *Endpoints*

- *Service* serve múltiplas *actions*
 - Cada action associada a um método no servidor
- Cada *service* associado a múltiplos *endpoints*
 - No servidor, o programador expõe os seus Endpoints
 - Mensagens enviadas para endpoints
- Endpoint define:
 - **Para onde** as mensagens devem ser enviadas (endereço)
 - **Como** as mensagens devem ser enviadas (*Binding*)
 - **O que** as mensagens devem conter (*Contract*)



Definição dos Serviços: *Contracts*

- *Service Contracts*
 - Mapeia um serviço remoto a uma interface (e.g. C#)
- *Operation Contracts*
 - Mapeia uma (ou mais) ações a cada método na interface
- *Message Contracts*
 - Permitem definir cabeçalhos específicos para as mensagens SOAP

```
using System.ServiceModel;
namespace ServiceLibrary {
    [ServiceContract(Namespace="http://example.org/echo/")]
    public interface IEchoService {
        [OperationContract]
        string Echo(string msg);
    }
}
```

Implementação do Serviço

- Classe que implementa a interface do serviço

```
using System.ServiceModel;

namespace ServiceLibrary {
    [ServiceBehavior(
        InstanceContextMode=InstanceContextMode.Single,
        ConcurrencyMode=ConcurrencyMode.Multiple)]
    public class EchoService : IEchoService {
        public string Echo(string msg) {
            return msg;
        }
        ...
    }
}
```

Behaviors permitem configurar aspectos do processamento local

Uma única instância da classe serve as chamadas remotas

Múltiplas chamadas podem acontecer concorrentemente

Existe também operationBehavior

Definição dos Serviços: Bindings

- Binding define como o cliente deve enviar mensagens e como o servidor as deve processar
- WCF já inclui largo conjunto de bindings pré-definidos
 - Mas programador também pode criar novos bindings

Class Name	Element Name	Transport	Encoding	WS-* Protocols
BasicHttpBinding	basicHttpBinding	HTTP	XML 1.0	WS-I Basic Profile 1.1
WSHttpBinding	wsHttpBinding	HTTP	XML 1.0	Message security, reliable sessions, and transactions
WSDualHttpBinding	wsDualHttpBinding	HTTP	XML 1.0	Message security, reliable sessions, and transactions
NetTcpBinding	netTcpBinding	TCP	Binary	Transport security, reliable sessions, and transactions
NetNamedPipeBinding	netNamedPipeBinding	Named Pipes	Binary	Transport security, reliable sessions, and transactions
NetMsmqBinding	netMsmqBinding	MSMQ	Binary	Transport security and queue transactions

Definição de Serviços: Exemplo

```
using System;
using System.ServiceModel;
using ServiceLibrary;

class Program {
    static void Main(string[] args) {
        using (ServiceHost host = new ServiceHost(
            typeof(EchoService), new Uri("http://localhost:8080/echo")))
        {
            host.AddServiceEndpoint(typeof(IEchoService),
                new BasicHttpBinding(), "svc");
            host.AddServiceEndpoint(typeof(IEchoService),
                new NetTcpBinding(),
                "net.tcp://localhost:8081/echo/svc");
            host.Open() ;
            ...
        }
    }
}
```

1. Cria serviço, indicando classe que o implementa e endereço

2. Cria (múltiplos) endpoints

3. Inicia o serviço

Cliente: Exemplo

```
using System;
using System.ServiceModel;
using ServiceLibrary;

class Program {
    static void Main(string[] args) {
        try {
            // define service endpoints on client
            ServiceEndpoint httpEndpoint = new ServiceEndpoint(
                ContractDescription.GetContract( typeof(IEchoService)),
                new BasicHttpBinding(),
                new EndpointAddress("http://localhost:8080/echo/svc"));

            IEchoService svc = null;
            // create channel factory based on HTTP endpoint
            using (ChannelFactory<IEchoService> httpFactory =
                new ChannelFactory<IEchoService>(httpEndpoint)) {
                // create channel proxy for endpoint
                svc = httpFactory.CreateChannel();
                // invoke service operation
                Console.WriteLine("Invoking HTTP endpoint: {0}",
                    svc.Echo("Hello, world"));
            }
        }
    }
}
```

1. Cria endpoint do lado do cliente, especificando contract, binding e endereço do endpoint do servidor

2. Cria proxy
(equivalente a dynamic proxy do JAX-WS)

3. Faz chamada remota



binding		Interoperability	Mode of Security (Default)	Session (Default)	Transactions	Duplex
BasicHttpBinding		Basic Profile 1.1	(None), Transport, Message, Mixed	None, (None)	(None)	n/a
WSHttpBinding		WS	None, Transport, (Message), Mixed	(None), Transport, Reliable Session	(None), Yes	n/a
WS2007HttpBinding		WS-Security, WS-Trust, WS-SecureConversation, WS-SecurityPolicy	None, Transport, (Message), Mixed	(None), Transport, Reliable Session	(None), Yes	n/a
WSDualHttpBinding		WS	None, (Message)	(Reliable Session)	(None), Yes	Yes
WSFederationHttpBinding		WS-Federation	None, (Message), Mixed	(None), Reliable Session	(None), Yes	No
WS2007FederationHttpBinding		WS-Federation	None, (Message), Mixed	(None), Reliable Session	(None), Yes	No
NetTcpBinding		.NET	None, (Transport), Message, Mixed	Reliable Session, (Transport)	(None), Yes	Yes
NetNamedPipeBinding		.NET	None, (Transport)	None, (Transport)	(None), Yes	Yes
NetMsmqBinding		.NET	None, Message, (Transport), Both	(None)	(None), Yes	No
NetPeerTcpBinding		Peer	None, Message, (Transport), Mixed	(None)	(None)	Yes
MsmqIntegrationBinding		MSMQ	None, (Transport)	(None)	(None), Yes	n/a