

J500: Java 2 Enterprise Edition

10 Java e XML

Helder da Rocha
www.argonavis.com.br

- *Parte I: XML e tecnologias relacionadas*
 - *Documentos XML; namespaces; processadores*
 - *DTD e XSchema*
 - *DOM e SAX*
 - *XSLT e XPath*
- *Parte II: APIs Java para XML e Web Services*
 - *Introdução a Web Services: SOAP, WSDL, UDDI*
 - *JAXP - Java API for XML Processing, JDOM, SAX e XSLT*
 - *JAXB - Java API for XML Binding*
 - *JAXM - Java API for XML Messaging*
 - *JAXR - Java API for XML Registry*
 - *JAXRPC - como criar um Web Service com JAX-RPC*
 - *Java Web Service Development Kit*

Parte I

Introdução a XML e tecnologias relacionadas

Parte I - Introdução a XML

Objetivos

- Responder às questões



- **Como implementar** soluções de gestão de informações usando XML?

- **Quando e como usar** as tecnologias e linguagens que viabilizam o compartilhamento de informações?

- Apresentar



- Breve **introdução ao XML** e tecnologias relacionadas.



- Recursos para manipular informações representadas em XML: **ferramentas, linguagens e tecnologias**

Parte I - Programa

Por que XML? Onde usar XML?

Como produzir documentos XML

Documentos válidos: DTD e XML Schema

Manipulação via programação em DOM e SAX

Transformação: XSLT e XPath

Localização e extração: XLink, XQuery e XPointer

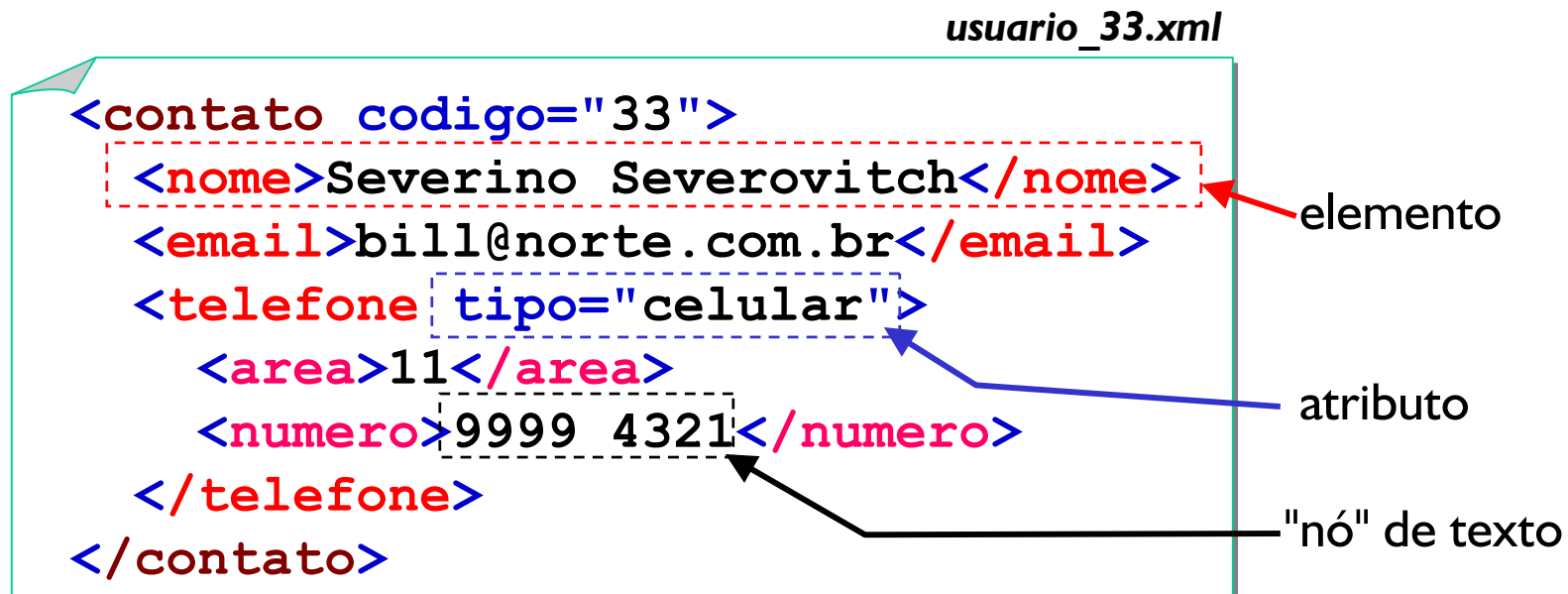
Visualização: XSL-FO e SVG

Demonstração: geração de HTML, RTF e PDF

Ferramentas e Conclusões

O que é XML?

- e**X**tensible **M**arkup **L**anguage: padrão W3C
- Uma maneira de **representar** informação
 - não é uma linguagem específica
 - não define vocabulário de comandos
 - não define uma gramática, apenas regras mínimas
- Exemplo: documento XML



XML versus HTML

HTML mostra
como
apresentar

```
<h1>Severino Severovitch</h1>  
<h2>bill@norte.com.br</h2>  
<p>  
  <b>11</b>  
  <i>9999 4321</i>  
</p>
```

XML mostra
o que
significa

```
<nome>Severino Severovitch</nome>  
<email>bill@norte.com.br</email>  
<telefone>  
  <ddd>11</ddd>  
  <numero>9999 4321</numero>  
</telefone>
```

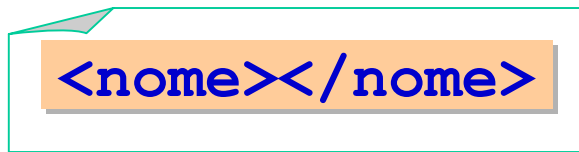
Anatomia de um documento XML

- Documentos XML são documentos de texto Unicode
 - É uma hierarquia de **elementos** a partir de uma **raiz**
 - Menor documento tem um elemento (vazio ou não):



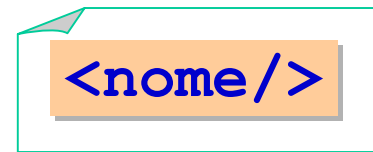
A diagram showing a root element in XML. It consists of a blue opening tag `<nome>`, the text "Северино Северович", and a blue closing tag `</nome>`. The entire structure is enclosed in a dashed-line box with a light blue shadow, representing the root element.

- Menor documento contendo elemento vazio



A diagram showing an empty XML element. It consists of a blue opening tag `<nome>` and a blue closing tag `</nome>`. The entire structure is enclosed in a light blue box with a shadow.

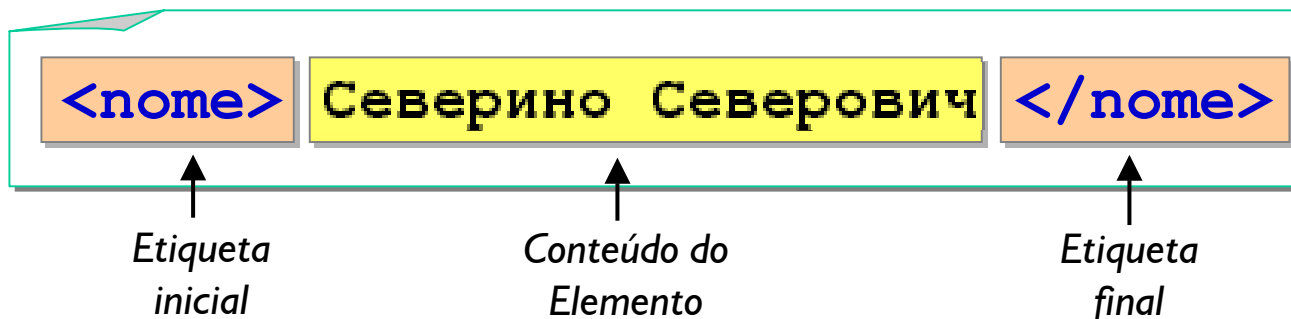
=



A diagram showing a self-closing XML element. It consists of a blue opening tag `<nome />`. The entire structure is enclosed in a light blue box with a shadow.

Elemento raiz

- Menor documento contendo elemento e conteúdo texto



A diagram showing an XML element with content and labels. It consists of a blue opening tag `<nome>`, the text "Северино Северович", and a blue closing tag `</nome>`. The opening and closing tags are highlighted in light blue boxes. Below the diagram, three arrows point to the respective parts: "Etiqueta inicial" points to the opening tag, "Conteúdo do Elemento" points to the text, and "Etiqueta final" points to the closing tag.

Partes de um documento

elemento raiz

declaração XML

nó raiz (/)

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

atributos

```
<cartao-simples>
```

```
<logotipo href="/imagens/logo14bis.gif" />
```

```
<nome>Alberto Santos Dumont</nome>
```

```
<endereco>Rua do Encanto, 22 - 2o. andar -  
Centro - 25600-000 - Petrópolis - RJ</endereco>
```

```
<email>dumont@14bis.com.br</email>
```

```
<telefone tipo="residencial">
```

```
<ddd>21</ddd>
```

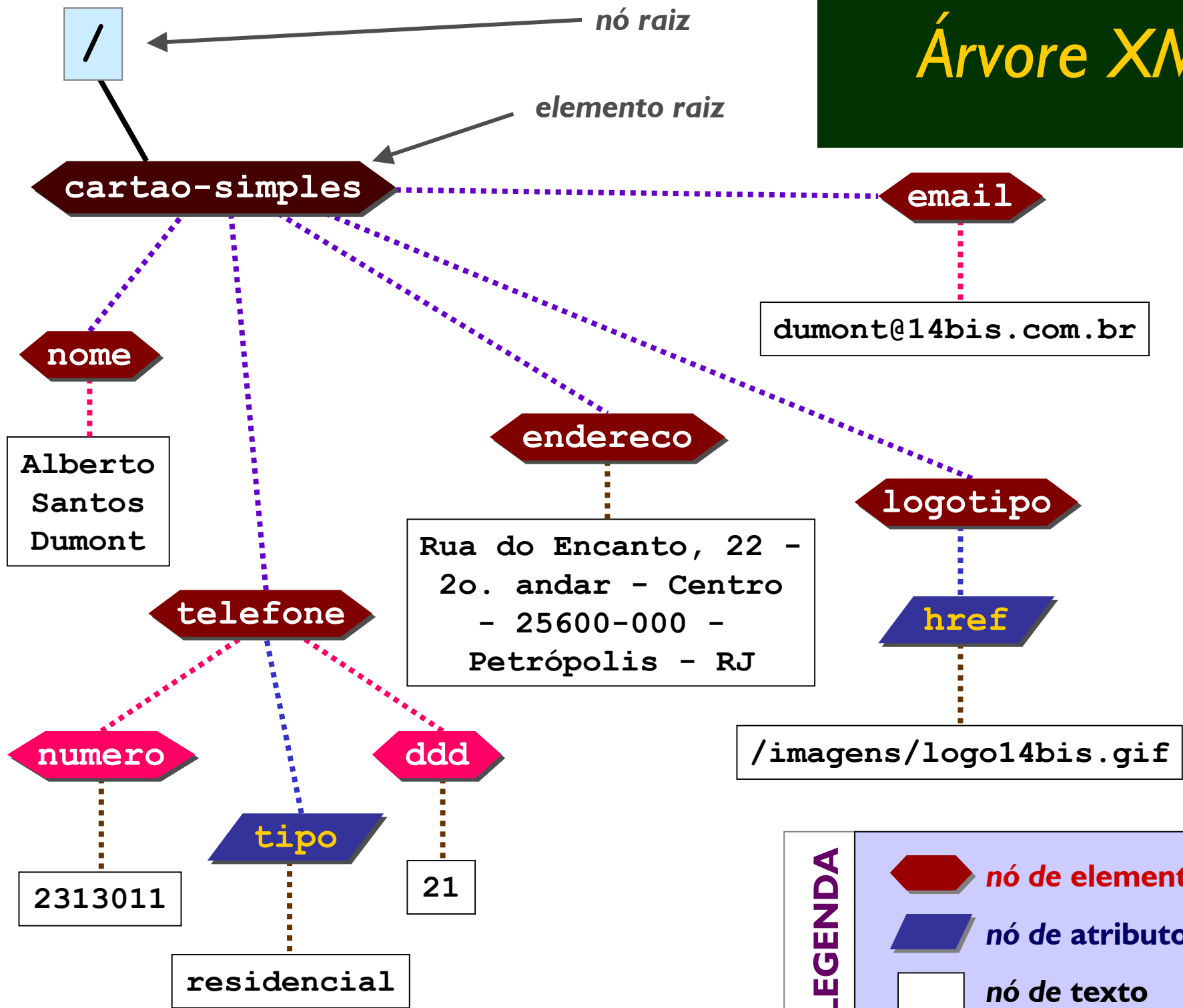
```
<numero>2313011</numero>
```

```
</telefone>
```

```
</cartao-simples>
```

elementos

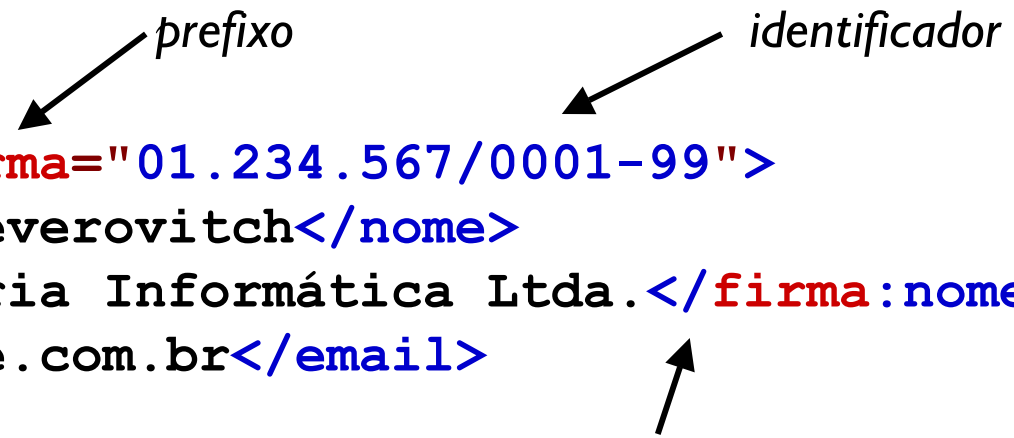
Árvore XML



XML Namespaces

- Limita o escopo de elementos
 - Evita conflitos quando duas linguagens se cruzam no mesmo documento
- Consiste da associação de um **identificador** a cada elemento/atributo da linguagem, que pode ser
 - **herdado** através do escopo de uma sub-árvore
 - atribuído explicitamente através de um **prefixo**
- Exemplo

```
<cadastro xmlns:firma="01.234.567/0001-99">  
  <nome>Severino Severovitch</nome>  
  <firma:nome>Sibéria Informática Ltda.</firma:nome>  
  <email>bill@norte.com.br</email>  
</cadastro>
```



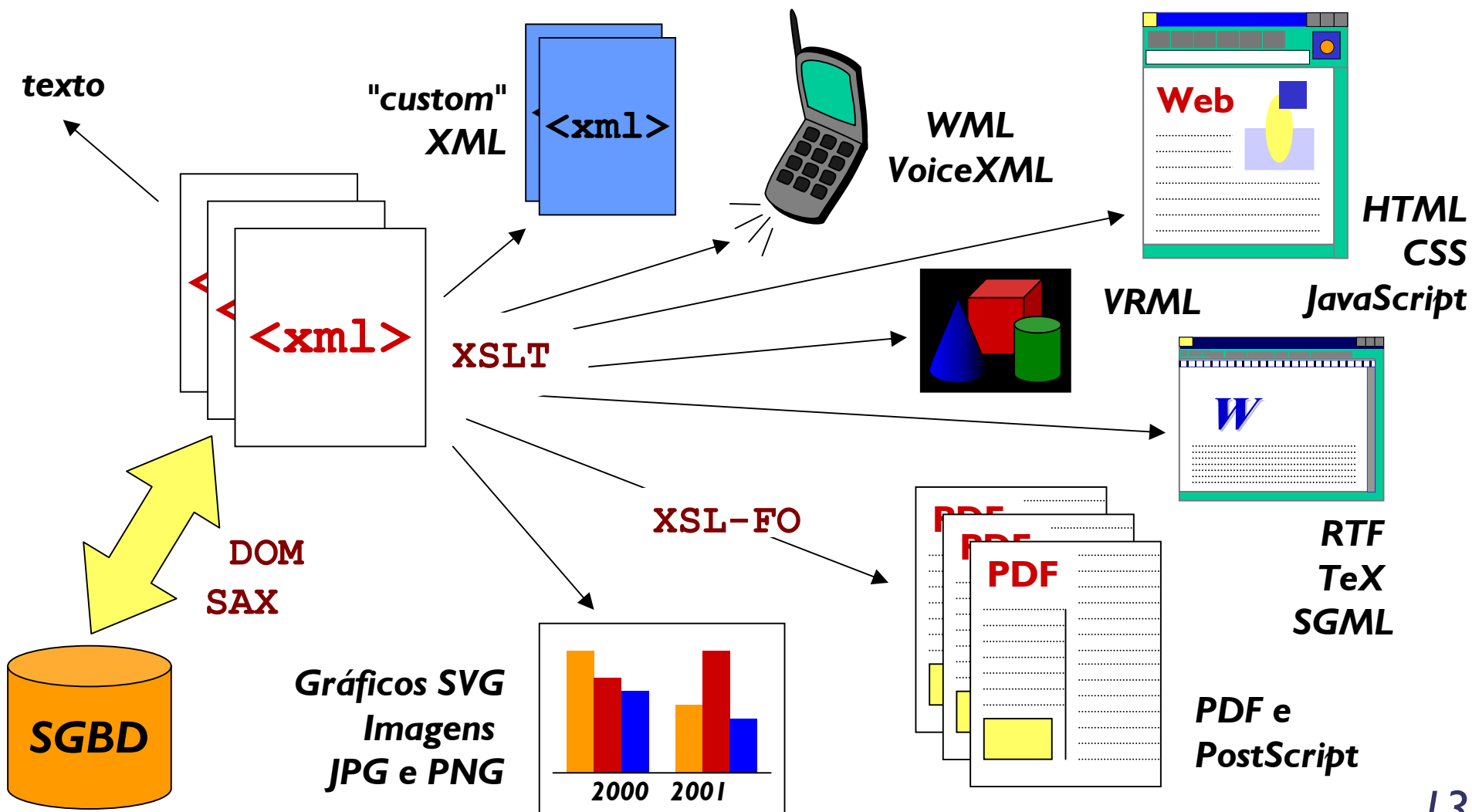
Este elemento <nome> pertence a outro namespace

Por que usar XML para compartilhar dados?

- Porque é um **padrão aberto**
 - Facilidade para converter para formatos proprietários
- Porque é **texto**
 - Fácil de ler, fácil de processar, menos incompatibilidades
- Porque promove a **separação** entre estrutura, conteúdo e apresentação
 - Facilita geração de dados para visualização dinâmica
 - Evita repetição de informação / simplifica manutenção
- Porque permitirá **semântica** na Web
 - Elementos HTML não carregam significado, apenas dicas de formatação: mecanismos de busca ficam prejudicados
 - Solução com XML dependerá de suporte dos clientes

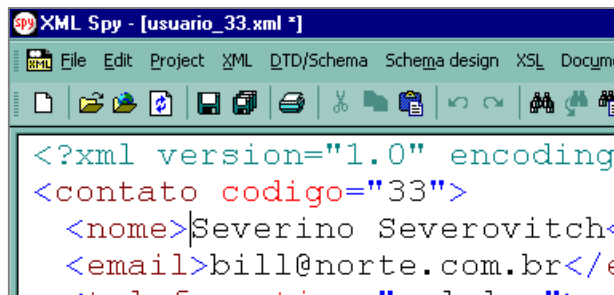
Onde usar XML?

- *Dados armazenados em XML podem ser facilmente transformados em outros formatos*

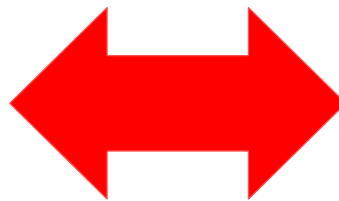


Como produzir XML

- **Criando** um documento de texto Unicode a partir de qualquer editor de textos

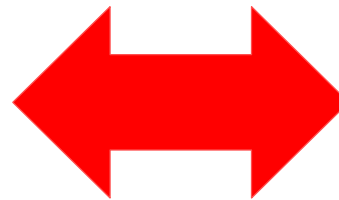
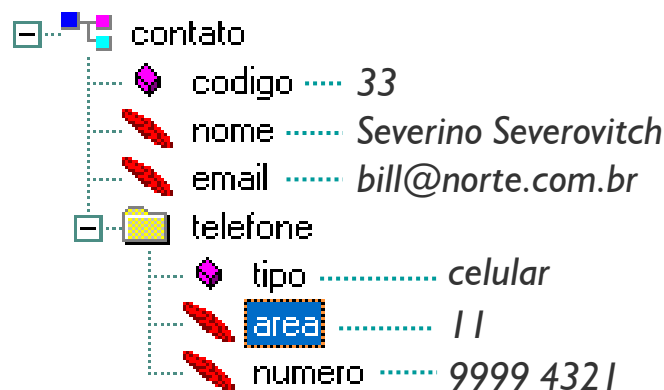


```
<?xml version="1.0" encoding="UTF-8" ?>
<contato codigo="33">
  <nome>Severino Severovitch</nome>
  <email>bill@norte.com.br</email>
  <telefone tipo="celular">
    <area>11</area>
    <numero>9999 4321</numero>
  </telefone>
</contato>
```



```
<contato codigo="33">
  <nome>Severino Severovitch</nome>
  <email>bill@norte.com.br</email>
  <telefone tipo="celular">
    <area>11</area>
    <numero>9999 4321</numero>
  </telefone>
</contato>
```

- **Gerando** um documento a partir de uma árvore montada dinamicamente



```
<contato codigo="33">
  <nome>Severino Severovitch</nome>
  <email>bill@norte.com.br</email>
  <telefone tipo="celular">
    <area>11</area>
    <numero>9999 4321</numero>
  </telefone>
</contato>
```

Documentos XML bem formados

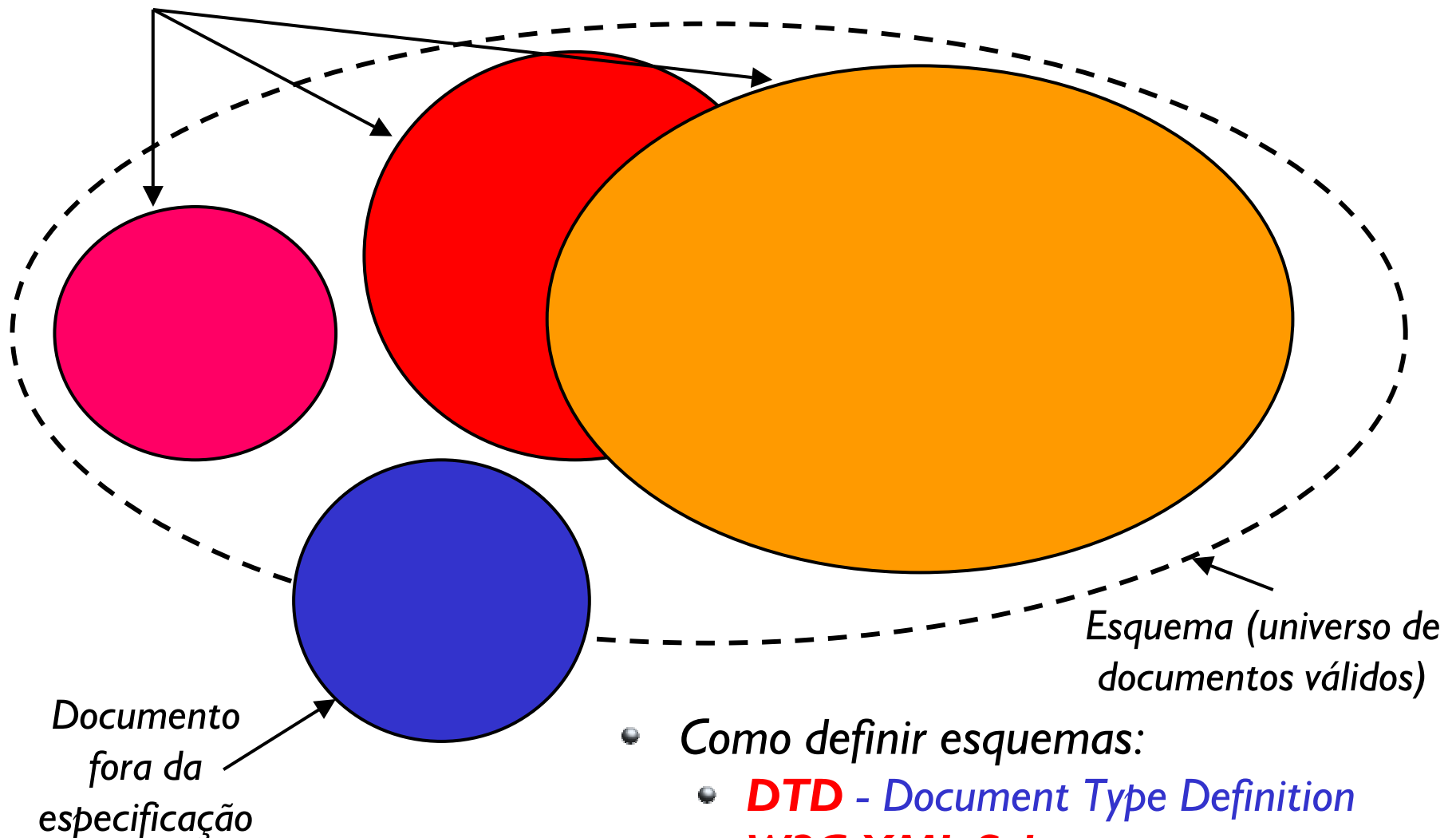
- *Para que possa ser manipulado como uma árvore, um documento XML precisa ser **bem formado***
 - *Documentos que não são bem formados não são documentos XML*
- *Documentos bem-formados obedecem as regras de construção de documentos XML genéricos*
- *Regras incluem*
 - *Ter um, e apenas um, elemento raiz*
 - *Valores dos atributos estarem entre aspas ou apóstrofes*
 - *Atributos não se repetirem*
 - *Todos os elementos terem etiqueta de fechamento*
 - *Elementos estarem corretamente aninhados*

- Um XML bem construído pode não ser **válido** em determinada aplicação
- Aplicação típica pode esperar que
 - elementos façam parte de um **vocabulário** limitado,
 - certos atributos tenham **valores** e **tipos** definidos,
 - elementos sejam organizados de acordo com uma determinada estrutura **hierárquica**, etc.
- É preciso **especificar** a linguagem!
 - **Esquema**: modelo que descreve todos os elementos, atributos, entidades, suas relações e tipos de dados
- Um documento XML é considerado válido **em relação a um esquema** se obedecer todas as suas regras

Esquema

Documentos que aderem à especificação (válidos)

- O esquema representa uma **classe**
- Os documentos são **instâncias**



DTD vs. XML Schema

- Um esquema é essencial para que haja **comunicação usando XML**
 - Pode ser estabelecido "informalmente" (via software)
 - Uso formal permite validação usando ferramentas genéricas de manipulação de XML
- Soluções

DTD

```
<!ELEMENT contato  
    (nome, email, telefone)>  
<!ATTLIST contato  
    codigo NMTOKEN #REQUIRED>
```

- Simples mas não é XML
- Não suporta namespaces
- Limitado quando a tipos de dados

XSchema

```
<xsd:schema  
    xmlns:xsd=".../XMLSchema">  
<xsd:element name="contato">  
    <xsd:complexType>  
        <xsd:attribute name="codigo"  
            use="required">
```

- É XML, porém mais complexo
- Suporta namespaces
- Permite definição de tipos

Visualização em um browser

- **Folha de estilo**: conjunto de regras para formatar ou transformar as informações de um documento XML
- **CSS** - Cascading Style Sheets
 - Transformação visando apresentação visual
 - Aplicação do estilo em tempo de execução no cliente
- **XSLT** - eXtensible Stylesheet Language
 - Transformação em texto, HTML ou outro formato
 - Aplicação em tempo real ou prévia (no servidor)
- Se não estiver associado a uma folha de estilo, o documento XML não tem uma "aparência" definida
 - Internet Explorer e outros mostram a árvore-fonte XML
 - Netscape mostra apenas os nós de texto

Como manipular XML?

- Há duas APIs padrão para manipular (interpretar, gerar, extrair dados e tratar eventos) arquivos XML:
 - W3C Document Object Model (W3C DOM)
 - Simple API for XML (SAX)
- Servem a finalidades diferentes
- Implementações disponíveis em várias linguagens
- **SAX** oferece métodos que respondem a **eventos** produzidos durante a leitura do documento
 - notifica quando um elemento abre, quando fecha, etc.
- **DOM** monta uma **árvore**, que permite a navegação na estrutura do documento
 - propriedades dos objetos podem ser manipuladas

Leitura de XML com SAX

- *Se um processador SAX receber o documento ...*

```
<carta>  
  <mensagem id="1">Bom dia!</mensagem>  
</carta>
```

- *... ele irá disparar a seguinte seqüência de eventos:*

```
⇒ startDocument()  
  ⇒ startElement("carta", [])  
    ⇒ startElement("mensagem", [Attribute("id", "1")])  
      ⇒ characters("Bom dia!")  
    ⇒ endElement("mensagem")  
  ⇒ endElement("carta")  
⇒ endDocument()
```

- *Programador deve implementar um objeto "ouvinte" para capturar os eventos e extrair as informações desejadas*

Criação de documentos com DOM (I)

■ Criação dos elementos

/

Document

Obter objeto do tipo **Document** (raiz)
(dependente de processador): **doc**

<carta>

Element

```
carta = doc.createElement("carta")
```

<mensagem>

Element

```
mens = doc.createElement("mensagem")
```

Bom dia!

String

```
texto = doc.createTextNode("Bom dia!")
```

■ Atributos

<mensagem id="1">

```
mens.setAttribute("id", "1")
```

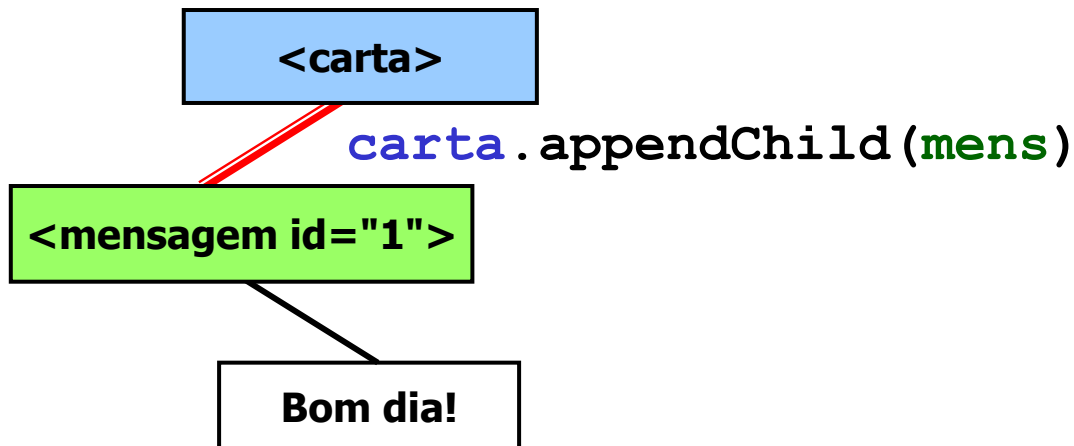
Criação de documentos com DOM (2)

Montagem da árvore passo-a-passo

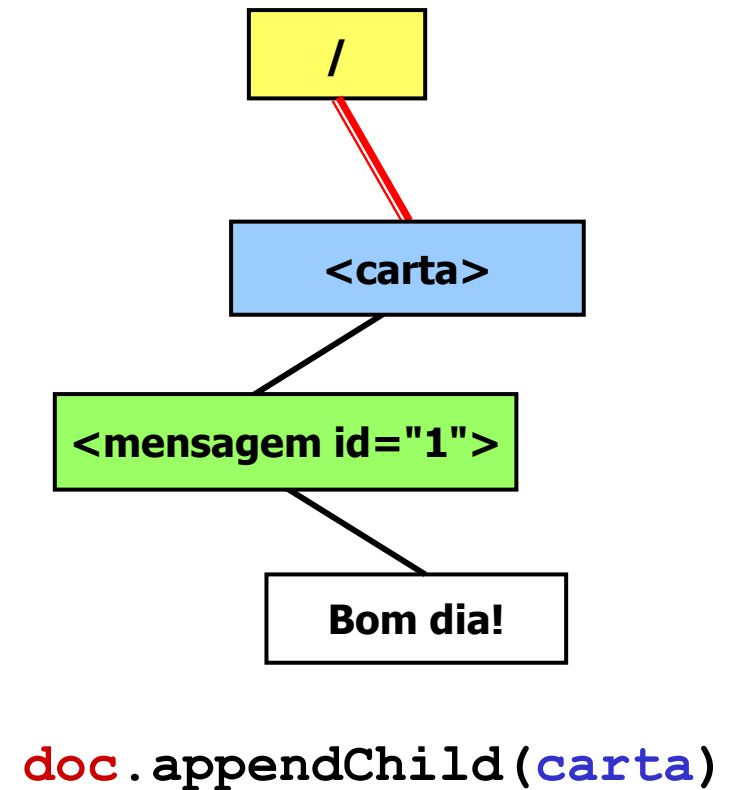
1. Sub-árvore `<mensagem>`



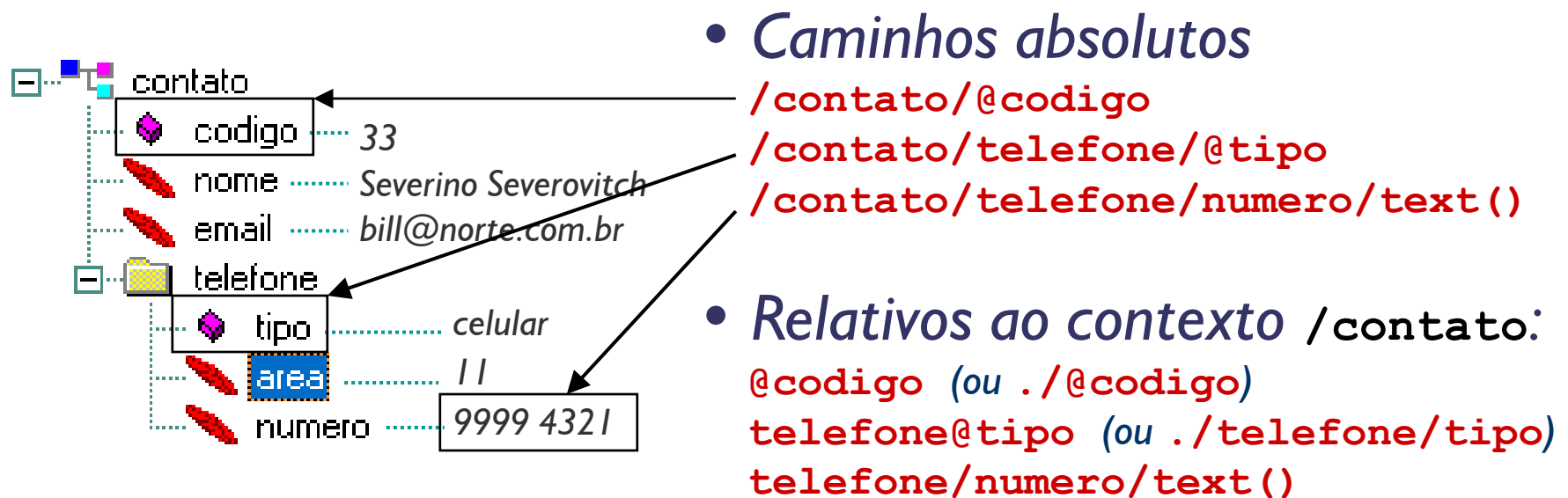
2. Sub-árvore `<carta>`



3. Árvore completa



- Linguagem usada para **navegar** na árvore XML
 - Uma **expressão XPath** é um caminho* na árvore que resulta em um valor (número, texto, booleano), objeto (elemento, atributo, nó de texto) ou conjunto de objetos



- Expressões XPath são usadas dentro de **atributos XML**
 - Usadas em XSLT, XLink, XQuery e XPointer


```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="livro/titulo">
    <td><xsl:value-of select="." /></td>
```

■ XSL Transformations

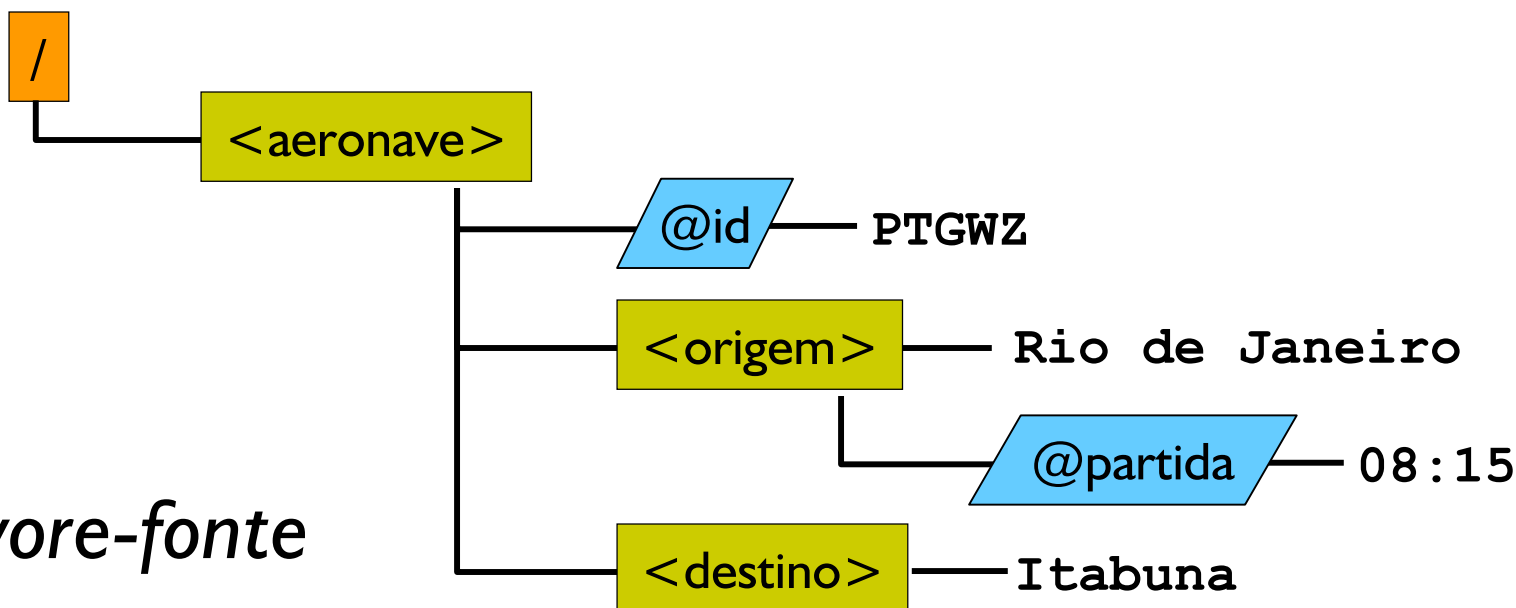
- Linguagem (XML) para criação de documentos que contêm regras de transformação para documentos XML
- Documentos escritos em XSLT são chamados de **folhas de estilo** e contêm
 - Elementos XSLT: <template>, <if>, <foreach>, ...
 - Expressões XPath para localizar nós da árvore-fonte
 - Texto ou XML a ser gerado no documento-resultado
- Processador XSLT



XSLT: documento-fonte (I)

- Considere o seguinte documento-fonte:

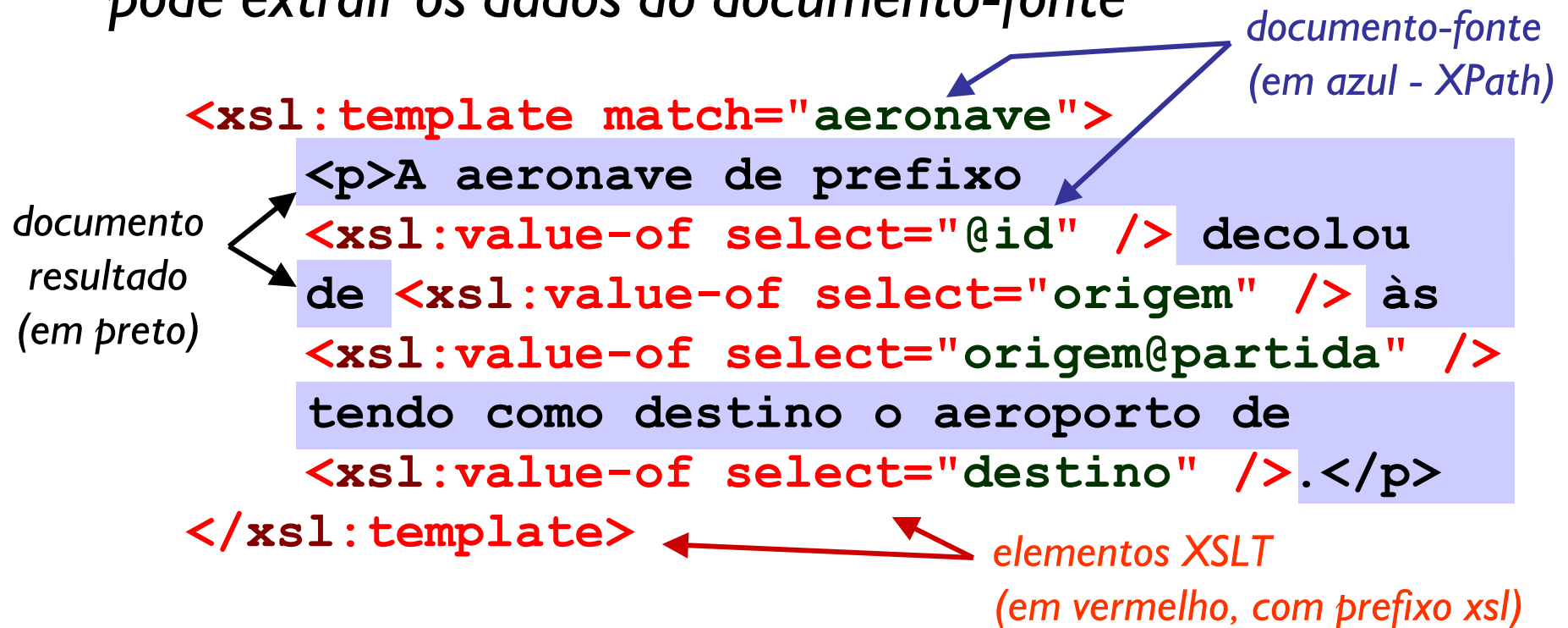
```
<aeronave id="PTGWZ">  
  <origem partida="08:15">Rio de  
                                Janeiro</origem>  
  <destino>Itabuna</destino>  
</aeronave>
```



- *Árvore-fonte*

XSLT: folha de estilos (2)

- O seguinte **template** (parte de uma folha de estilos XSLT) pode extrair os dados do documento-fonte



- Elementos XSLT geralmente são usados com um **prefixo** associado ao seu **namespace**: `<xsl:elemento>` para evitar conflitos com o documento-resultado.

XSLT: documento-resultado (3)

- Após a transformação, o resultado será

```
<p>A aeronave de prefixo  
PTGWZ decolou  
de Rio de Janeiro às  
8:15  
tendo como destino o aeroporto de  
Itabuna.</p>
```

- Para obter outros resultados e gerar outros formatos com os mesmos dados, deve-se criar folhas de estilo adicionais

XLink, XPointer e XQuery

- **XLink**: é uma especificação W3C que permite definir vínculos entre documentos XML
 - Funcionalidade mínima é igual ao `<a href>` do HTML
 - Funcionalidade estendida permite vínculos bidirecionais, arcos, vários níveis de semântica, etc.
 - É uma coleção de atributos, com namespace próprio, que podem ser usados em elementos de qualquer linguagem XML.
- **XPointer**: aponta para partes de documentos XML
 - Identificador (ID) colocado no destino, acessível através de fragmento de URL: `xlink:href="#identificador"`
 - Caminho resultante de expressão XPath: `xpointer(/livro/id)`
- **XQuery**: linguagem para pesquisar documentos XML
 - Exemplo:

```
FOR $b IN document("usuario_33.xml")/contato
WHERE nome="Severino Severovitch"
RETURN $b
```

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

```
<fo:layout-master-set>
```

```
<fo:simple-page-master master-name="p1">
```

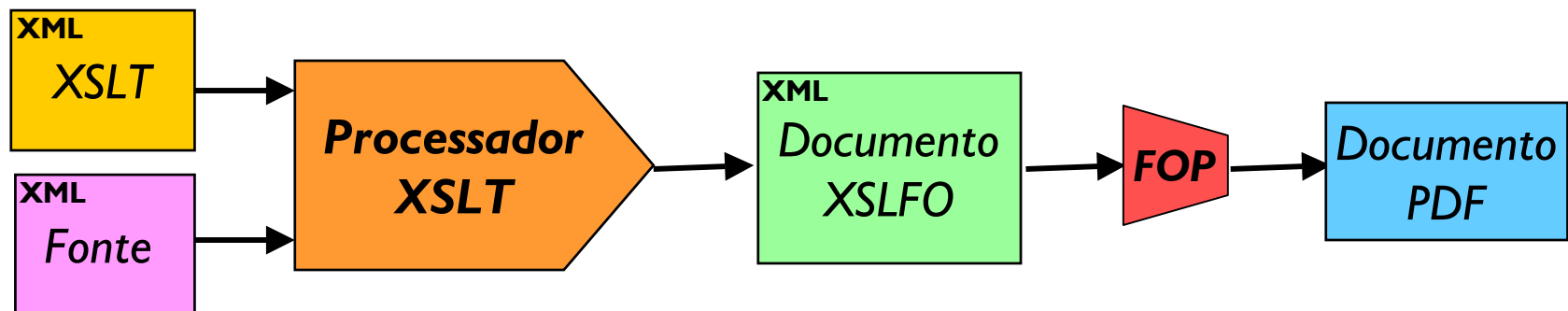
```
<fo:region-body/>
```

XSL-FO

■ XSL Formatting Objects

- Linguagem XML de **descrição de página** com os mesmos recursos que PostScript ou PDF
- Descreve o **layout preciso** de texto e imagens
- Possui centenas de elementos, atributos e propriedades (que são semelhantes às propriedades do CSS)
- Páginas são facilmente convertidas para PDF e PostScript
- Ideal para gerar documentos para impressão (livros, etc.)

■ Normalmente **gerada** via XSLT



XSL-FO: menor documento

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

```
<fo:layout-master-set>
```

```
<fo:simple-page-master master-name="p1">
```

```
<fo:region-body/>
```

```
</fo:simple-page-master>
```

```
</fo:layout-master-set>
```

```
<fo:page-sequence master-name="p1">
```

```
<fo:flow flow-name="xsl-region-body">
```

```
<fo:block color="blue" font-size="20pt">
```

```
    Hello PDF!
```

```
</fo:block>
```

```
</fo:flow>
```

```
</fo:page-sequence>
```

```
</fo:root>
```

← Este é o "<head>"
do XSL-FO

↗ Ligação entre as
regras de layout e
o conteúdo afetado

← Este é o "<body>"
do XSL-FO

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Página XHTML</title></head>
  <body>
    <h1>Página XHTML</h1>
```

XHTML

■ eXtensible *HTML*

- Linguagem XML de **descrição de página Web**
- Mesmos elementos do HTML 4.0 Strict
- Elementos descrevem **somente a estrutura** dos componentes da página.
 - A **forma** precisa ser especificada usando CSS: não há elementos/atributos para mudar cor, alinhamento, etc.
- Pode ser misturada (estendida) com outras linguagens XML (MathML, SVG, linguagens proprietárias)

■ Normalmente **gerada** via XSLT




```
<svg>
```

```
<circle style="fill: red" cx="3cm" cy="3cm" r="2.5cm" />  
<rect style="fill: blue" x="6cm" y="6cm"  
  height="2.5cm" width="1.5cm" />
```

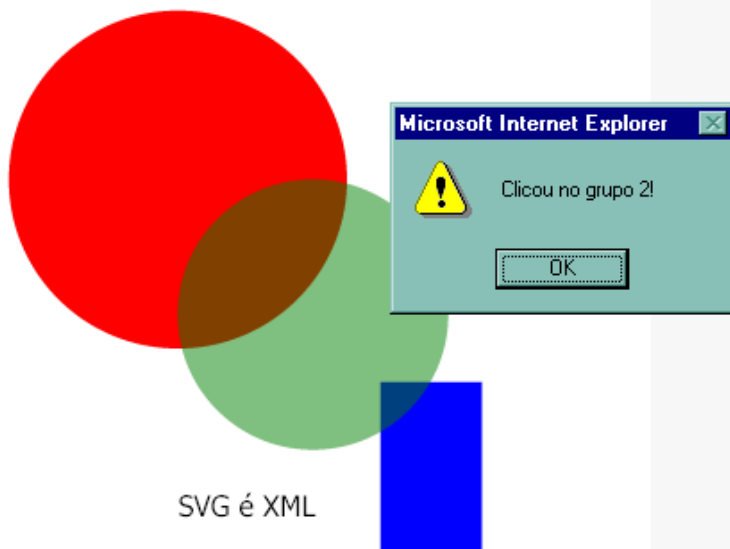
SVG

■ W3C **S**calable **V**ector **G**raphics

- Gráficos vetoriais em XML
- Plug-ins para principais browsers: concorre com Flash
- Suporta animações, links, JavaScript, CSS
- Produzido por ferramentas como Adobe Illustrator
- Pode ser embutido no código XHTML e XSL-FO



Exemplo de SVG



SVG é XML

JavaScript

```
<svg width="10cm" height="10cm">
  <g onclick="alert('Clicou no grupo 1!')">
    <circle style="fill: red"
      cx="3cm" cy="3cm" r="2.5cm" />
    <rect style="fill: blue" x="6cm" y="6cm"
      height="2.5cm" width="1.5cm" /></g>
  <g onclick="alert('Clicou no grupo 2!')">
    <circle style="fill: green; opacity: 0.5"
      cx="5cm" cy="5cm" r="2cm" /></g>
  <a xmlns:xlink="http://www.w3.org/1999/xlink"
    xlink:href="http://www.w3.org/Graphics/SVG">
    <text style="color: black; font-family: tahoma;
      font-size: 12pt" x="3cm" y="8cm">
      SVG é XML</text></a>
</svg>
```

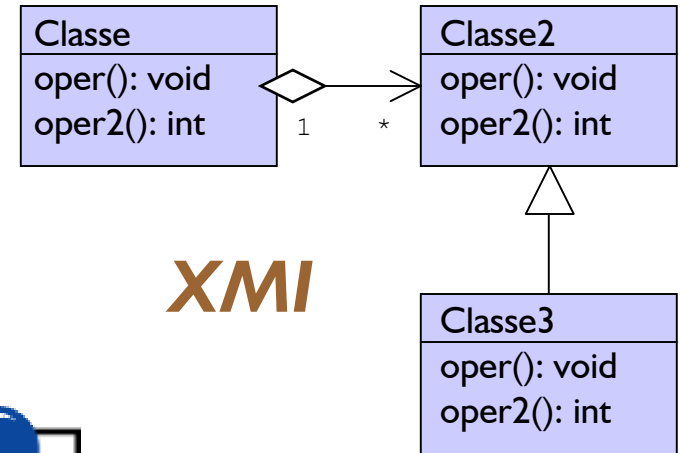
CSS

XLink

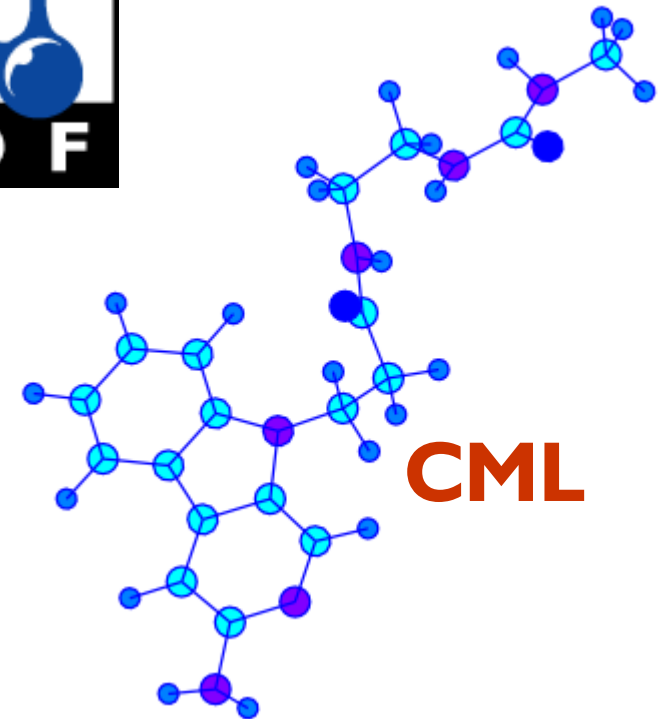
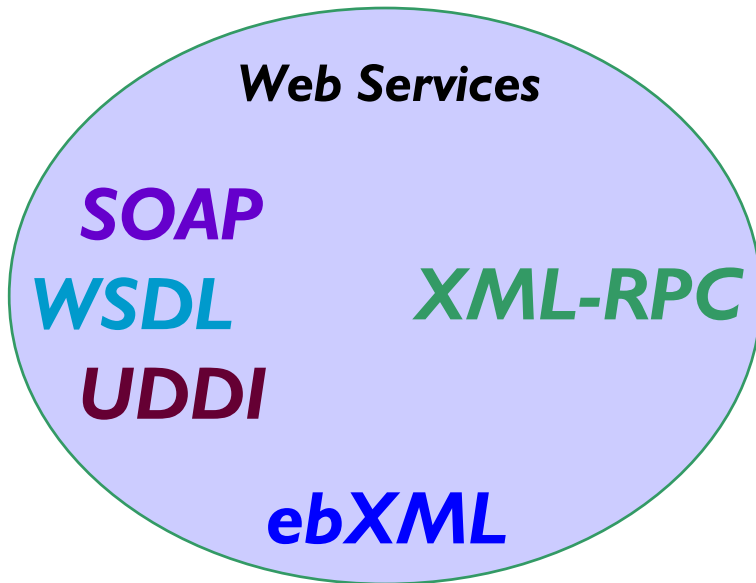
Algumas outras linguagens XML

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

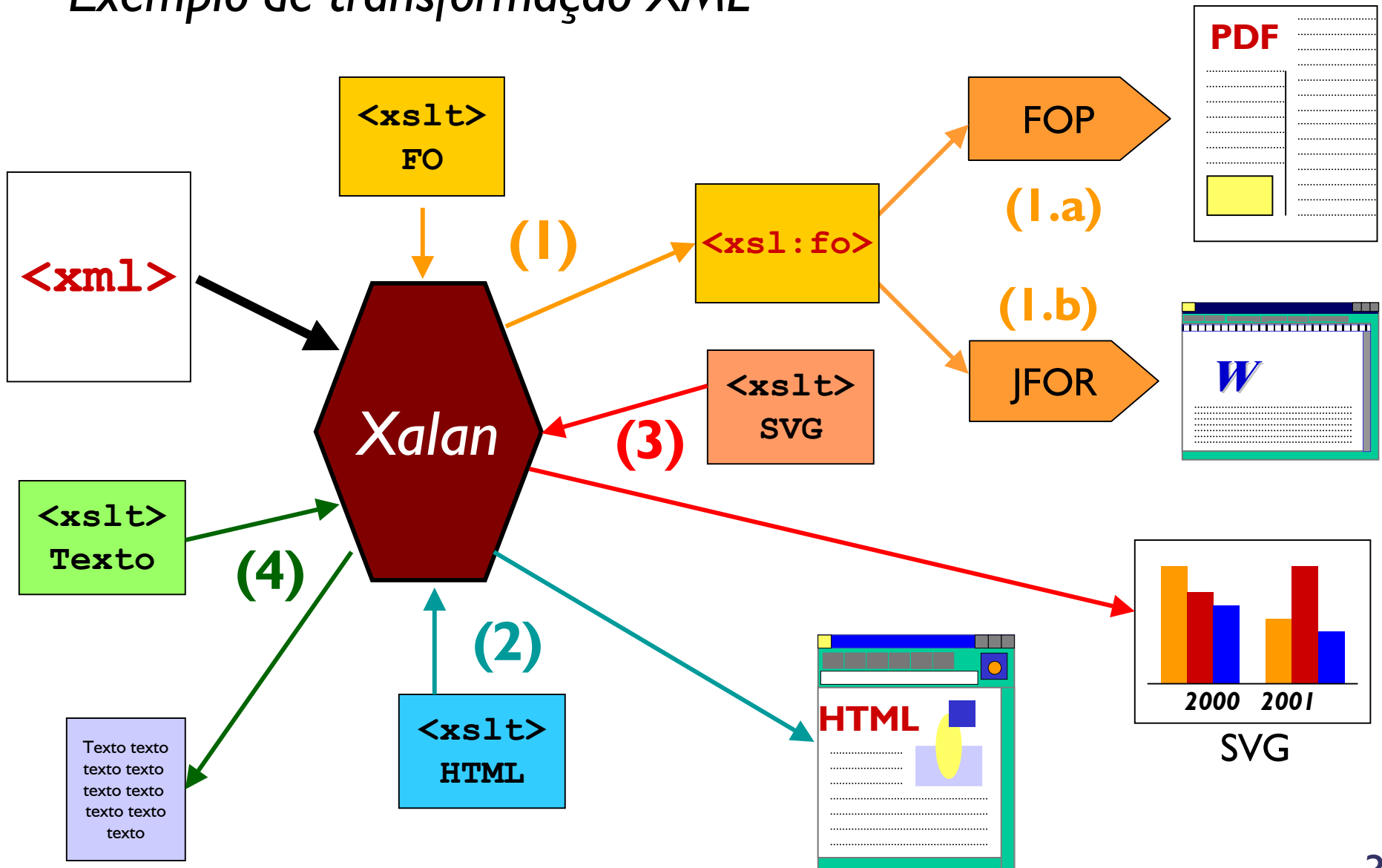
MathML



XMI



Exemplo de transformação XML



Ferramentas e APIs Java

- *Para programação*
 - *Parsers-validadores: Xerces, Crimson, MSXML 4.0*
 - *Validadores: MSV (Sun)*
 - *Transformadores XSL: TrAX (JAXP), Xalan, Xt, Saxon*
 - *APIs: JDOM, JAXP (DOM, SAX)*
 - *Veja mais em xml.apache.org e www.alphaworks.ibm.com*
- *Para edição (de XML genérico)*
 - *XML Spy Suite*
 - *Framemaker / ArborText*
 - *JEdit com plug-ins para XML, XSLT e XPath*
 - *Veja mais em www.w3.org/XML/*

- XML é uma ótima solução para **compartilhar** dados
- Para **implementar** soluções em gestão de informações usando XML, pode-se usar
 - **DTD** ou **XSchema** para especificar o modelo de dados e validar as informações
 - As APIs **DOM** ou **SAX** para extrair dados dos documentos, gerar documentos, ler e gravar em bancos de dados
 - **XSLT** e **XPath** para transformar os dados em outros formatos
 - **XLink**, **XPointer** e **XQuery** para criar vínculos lógicos entre os documentos e localizar seus componentes
 - **XSL-FO** ou **XHTML** para formatar os dados para impressão ou visualização na tela (PDF, Word ou Web)
 - **SVG** para gerar informações em forma de gráfico vetorial

- [1] World Wide Web Consortium (W3C). *eXtensible Markup Language*. <http://www.w3.org/XML/>. *Ponto de partida e principal fonte sobre XML e suas tecnologias "satélite". Contém últimas especificações de XML, XPath, XSchema, XSLT, XSL-FO, XQuery, XLink, XPointer, SVG, XHTML, CSS.*
- [2] Eric Armstrong et al. *Working with XML*. *Aborda DOM, SAX e XML com Java*. <http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/index.html>.
- [3] Adobe. *SVG Tutorial*. <http://www.adobe.com/svg/>. *Contém tutorial sobre SVG e links para o plug-in SVG da Adobe (Win/Mac).*
- [4] IBM Developerworks. <http://www-106.ibm.com/developerworks/>. *Diversos tutoriais e artigos sobre XML, XSLT, DOM e SAX usando geralmente Java.*
- [5] Doug Tidwell. *XSLT*. O'Reilly & Associates, 2001. *Explora XSLT com aplicações práticas em Java.*
- [6] Elliotte Rusty Harold. *XML Bible, Second Edition*, 2001. *Aborda todas as principais tecnologias W3C. 5 capítulos em <http://cafeconleche.org/books/bible2/>*
- [7] Erik T. Ray. *Learning XML*. O'Reilly & Associates, 2001. *Introdução ao XML e DTD, XSLT, XLink e XPointer (os dois últimos baseados em especificações draft).*

Parte II

Java, XML e Web Services

Parte II - Java, XML e Web Services

- **Objetivos:**
 - Definir **Web Services**
 - Descrever as **tecnologias XML** padrão que oferecem suporte a Web Services
 - Descrever as **APIs Java** distribuídas com o Java Web Services Development Pack 1.0
- **Mostrar como criar um Web Service**
 - Utilizar a API **JAX-RPC** para desenvolver e implantar um Web Service simples baseado no protocolo **SOAP**
 - Gerar uma interface **WSDL** e utilizá-la para construir um cliente para o serviço
 - Registrar uma organização e a localização do arquivo WSDL em um servidor **UDDI** local

O que são Web Services

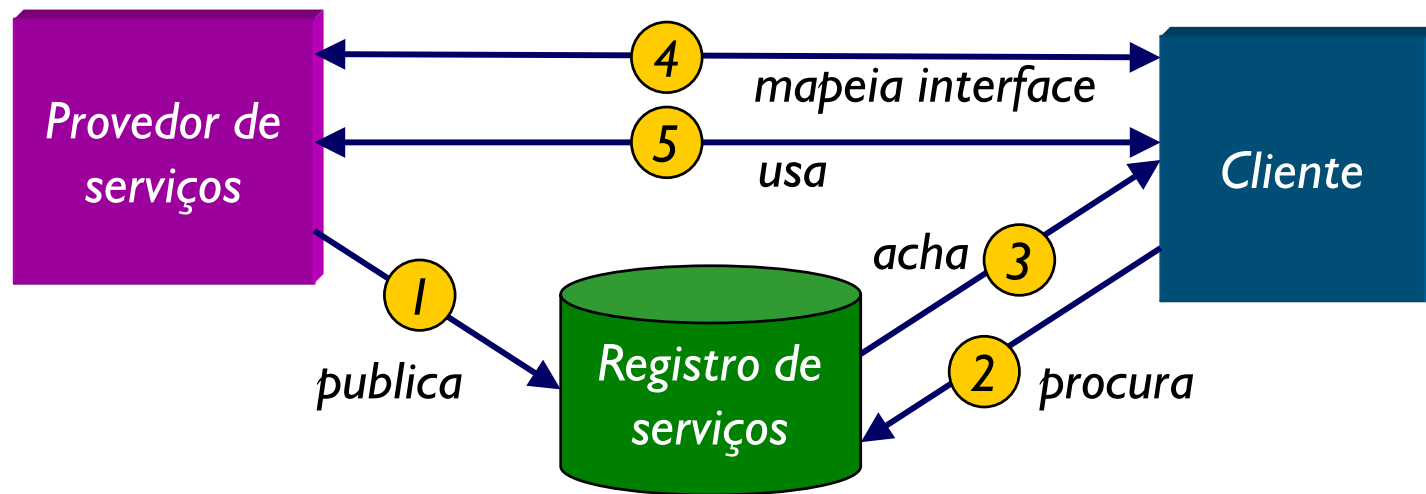
- Ambiente de computação distribuída (DCE) que utiliza **XML** em todas as camadas
 - No formato de dados usado na **comunicação**
 - Na **interface** usada para descrever as operações suportadas
 - Na aplicação usada para **registrar** e **localizar serviços**
- Serviços são transportados principalmente via HTTP
 - Podem também utilizar outros protocolos populares
- Web Services visam comunicação entre **máquinas**
 - Serviços podem ser implementados usando CGI (com C, Perl, etc.), ASP, PHP, servlets, JSP, CFML, etc.
 - Acesso é feito via clientes HTTP (ou de outros protocolos)
- Tudo isto já existia! Qual a novidade?

A novidade é a padronização!

- *Todas as camadas em XML!*
 - *Fácil de ler, transformar, converter*
 - *Existe ainda um esforço para padronizar os esquemas que definem a estrutura e vocabulário do XML usado*
- *Web Services dá nova vida ao RPC*
 - *Agora com formato universal para os dados!*
 - ➔ *Marshalling: converter dados em XML*
 - ➔ *Unmarshalling: extrair dados de XML*
- *Principais características do RPC com Web Services*
 - *Formato padrão de dados usados na comunicação é XML*
 - *Interoperabilidade em todos os níveis*
 - *Transporte é protocolo de larga aceitação: HTTP, SMTP,...*
 - *Transparência de localidade e neutralidade de linguagem*

Arquitetura de Web Services: papéis

- **Provedor de serviços**
 - Oferece serviços, alguns dos quais podem ser Web Services
- **Registro de serviços**
 - *Catálogo de endereços: repositório central que contém informações sobre web services*
- **Cliente de serviços**
 - *Aplicação que descobre um web service, implementa sua interface de comunicação e usa o serviço*



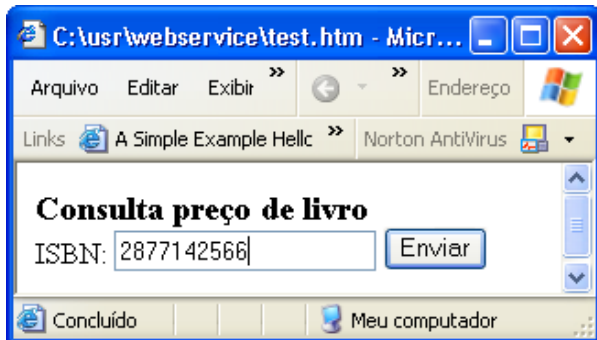
Arquitetura de Web Services: camadas

- *Camada de transporte*
 - Principais: HTTP (POST), FTP, SMTP
 - Emergentes: JRMP (Java RMI), IIOP (CORBA, EJB), JMS, IMAP, POP, BEEP, JXTA, ...
- *Camada de mensagens*
 - SOAP
- *Camada dados ou serviços*
 - XML (formato de mensagens)
 - XML-RPC
- *Camada de descrição de serviços*
 - WSDL
- *Camada de descoberta (registro)*
 - UDDI, ebXML



Requisição e resposta HTTP POST

- Clientes HTTP usam o método POST para **enviar** dados
 - Tipicamente usado por browsers para enviar dados de formulários HTML e fazer upload de arquivos
- Exemplo
 - Formulário HTML



```
<FORM ACTION="/cgi-bin/catalogo.pl"
      METHOD="POST">
  <H3>Consulta preço de livro</H3>
  <P>ISBN: <INPUT TYPE="text" NAME="isbn">
  <INPUT TYPE="Submit" VALUE="Enviar">
</FORM>
```

- Requisição POST gerada pelo browser para o servidor

Cabeçalho HTTP

Linha em branco

Mensagem (corpo da requisição)

```
POST /cgi-bin/catalogo.pl HTTP/1.0
Content-type: text/x-www-form-urlencoded
Content-length: 15

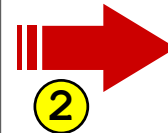
isbn=2877142566
```

Enviando XML sobre POST

- Você pode criar um serviço RPC simples (um Web Service!) trocando mensagens XML via HTTP POST!
 - Defina esquemas para as mensagens de chamada e resposta
 - Escreva cliente que envie requisições POST para servidor Web
 - Escreva uma aplicação Web (JSP, ASP, PHP, servlet, CGI)

```
POST /ISBNService.jsp HTTP/1.0
Content-type: text/xml
Content-length: 90
```

```
<chamada>
  <funcao>
    <nome>getPrice</nome>
    <param>2877142566</param>
  </funcao>
</chamada>
```



ISBNService.jsp

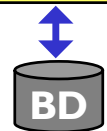
2877142566

ISBNQuery
getPrice()

19.50

3

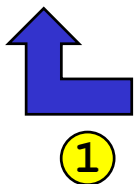
gera
resposta



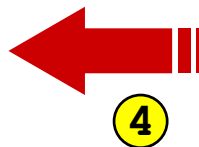
```
HTTP/1.1 200 OK
Content-type: text/xml
Content-length: 77
```

```
<resposta>
  <funcao>
    <param>19.50</param>
  </funcao>
</resposta>
```

gera
requisição



ISBNClient





- *Especificação para RPC em XML via HTTP POST*
 - *Projetada para ser a solução mais simples possível*
 - *Várias implementações: veja www.xml-rpc.com*
- *Exemplo anterior implementado com XML-RPC (cabeçalhos HTTP omitidos)*

```
<methodCall>
  <methodName>getPrice</methodName>
  <params>
    <param>
      <value><string>2877142566</string></value>
    </param>
  </params>
</methodCall>
```

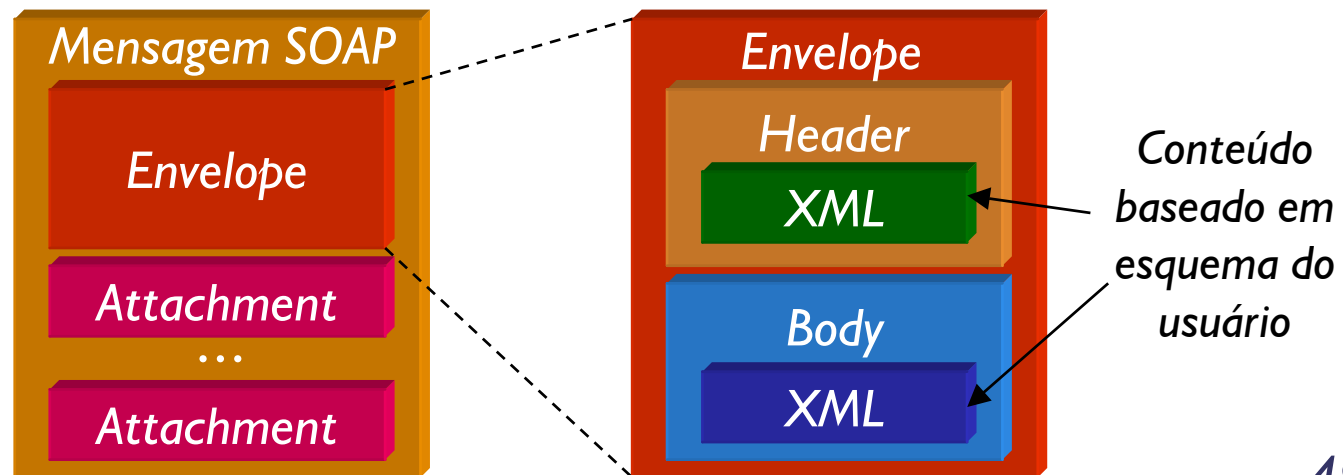
Requisição ←

Resposta →

```
<methodResponse>
  <params>
    <param>
      <value><double>19.5</double></value>
    </param>
  </params>
</methodResponse>
```


- **S**imple **O**bject **A**ccess **P**rotocol
- Protocolo padrão baseado em XML para trocar **mensagens** entre aplicações
 - SOAP não é um protocolo RPC, mas um par de mensagens SOAP pode ser usado para esse fim
 - Transporte pode ser HTTP, SMTP ou outro
 - Mensagens podem conter qualquer coisa (texto, bytes)
 - É extensível (mecanismo de RPC, por exemplo, é extensão)

Estrutura de
uma mensagem
SOAP



Simple requisição SOAP-RPC

- Principal aplicação do SOAP, hoje, é RPC sobre HTTP
 - Esquema do corpo da mensagem lida com RPC

```
POST /xmlrpc-bookstore/bookpoint/BookstoreIF HTTP/1.0
Content-Type: text/xml; charset="utf-8"
Content-Length: 585
SOAPAction: ""
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<env:Envelope
```

```
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
```

```
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
<env:Body>
```

```
  <ans1:getPrice xmlns:ans1="http://mybooks.org/wsdl">
```

```
    <String_1 xsi:type="xsd:string">2877142566</String_1>
```

```
  </ans1:getPrice>
```

```
</env:Body>
```

```
</env:Envelope>
```




Parâmetro (ISBN)

Resposta SOAP-RPC

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
SOAPAction: ""
Date: Thu, 08 Aug 2002 01:48:22 GMT
Server: Apache Coyote HTTP/1.1 Connector [1.0]
Connection: close
```

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:ns0="http://mybooks.org/types"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <env:Body>
    <ans1:getPriceResponse xmlns:ans1="http://mybooks.org/wsdl">
      <result xsi:type="xsd:decimal">19.50</result>
    </ans1:getPriceResponse>
  </env:Body>
</env:Envelope>
```



Resposta (Preço)

Descrição de um serviço RPC: WSDL

- *Para saber usar um Web Service, é preciso*
 - *Saber o que um serviço faz (quais as operações?)*
 - *Como chamar suas operações (parâmetros? tipos?)*
 - *Como encontrar o serviço (onde ele está?)*
- **Web Services Description Language**
 - *Documento XML de esquema padrão que contém todas as informações necessárias para que um cliente possa utilizar um Web Service*
 - *Define informações básicas (operações, mapeamentos, tipos, mensagens, serviço) e suporta extensões*
 - *Tem basicamente mesmo papel que linguagens IDL usadas em outros sistemas RPC*
 - *Pode ser usada na geração automática de código*

Interoperabilidade com WSDL

- WSDL serve apenas para **descrever interfaces**
 - Não serve para ser executada
 - Nenhuma aplicação **precisa** da WSDL (não faz parte da implementação - é só descrição de interface)
- WSDL pode ser mapeada a linguagens (binding)
 - **Mapeamento**: tipos de dados, estruturas, etc.
 - Pode-se **gerar código** de cliente e servidor a partir de WSDL (stubs & skeletons) em tempo de compilação ou execução
- WSDL facilita a interoperabilidade
 - Viabiliza RPC via SOAP
 - Pode-se gerar a **parte do cliente** em uma plataforma (ex: .NET) e a **parte do servidor** em outra (ex: J2EE), viabilizando a comunicação entre arquiteturas diferentes. 53

Exemplo: WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookstoreService"
  targetNamespace="http://mybooks.org/wsdl"
  xmlns:tns="http://mybooks.org/wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types>...</types>
  <message name="BookstoreIF_getPrice">
    <part name="String_1" type="xsd:string"/>
  </message>
  <message name="BookstoreIF_getPriceResponse">
    <part name="result" type="xsd:decimal"/>
  </message>
  <portType name="BookstoreIF">
    <operation name="getPrice" parameterOrder="String_1">
      <input message="tns:BookstoreIF_getPrice"/>
      <output message="tns:BookstoreIF_getPriceResponse"/>
    </operation>
  </portType>
  <binding ... > ...</binding>
  <service ... > ... </service>
</definitions>
```

Compare com a mensagem SOAP mostrada anteriormente

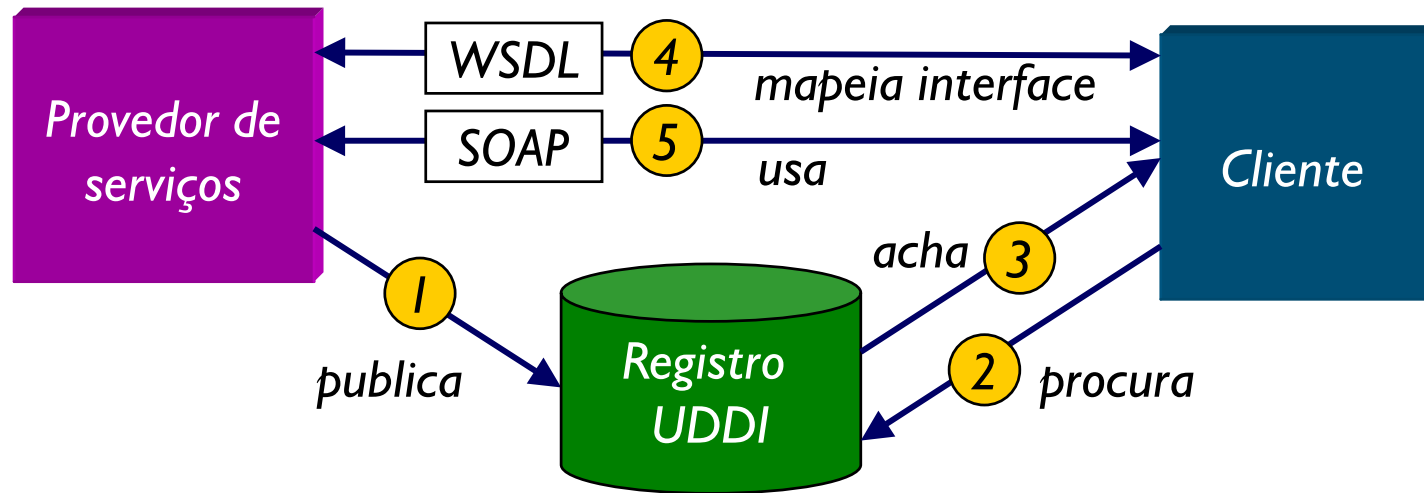
Informa onde está o serviço (endpoint)

Registro e localização do serviço: UDDI

- **U**niversal **D**iscovery and **D**escription **I**ntegration
 - Registro global para Web Services: nuvem UDDI
 - Esquema padrão (XML) para representar firmas, serviços, pontos de acesso dos serviços, relacionamentos, etc.
 - Objetivo é permitir a maior automação no uso dos serviços
 - Registro UDDI acha e devolve URL do WSDL ou serviço
- Registro centralizado permite
 - Independência de localização
 - Facilidade para pesquisar e utilizar serviços existentes
- Tipos de informações armazenadas em UDDI
 - White pages: busca um serviço pelo nome
 - Yellow pages: busca um serviço por assunto
 - Green pages: busca com base em características técnicas

Web Services: Resumo

- Arquitetura de serviços usando SOAP, WSDL e UDDI



- Comparação com outras soluções de RPC

	Java RMI	CORBA	RMI / IIOP	Web Services
Registro	RMI Registry	COS Naming	JNDI	UDDI
Descrição de Serviços	Java	OMG IDL	Java	WSDL
Transporte	Java RMI	IIOP	IIOP	SOAP

Tecnologias Java para Web Services

- *Java 2 Enterprise Edition (J2EE)*
 - *Versão 1.3 (atual): já possui todos os recursos necessários para **infraestrutura** de Web Services (servlets, JSP)*
 - *Versão 1.4 (2003): integração nativa com Web Services - será mais fácil transformar EJBs e componentes Web em clientes e provedores de Web Services*
- *Para criar Web Services em Java hoje*
 - *(1) Java Servlet API 2.3, JSP 1.2, JSTL 1.0*
 - *(2) Implementações Java de XML, SOAP, UDDI (há várias: IBM WSDL4J, UDDI4J, Apache SOAP, AXIS, Xerces, Xalan)*
 - *(3) Java XML Pack ("série JAX")*

➔ *Java Web Services Development Pack = (1) + (3)*

Java Web Services Development Pack 1.0

- APIs
 - Processamento XML: **JAXP 1.1**
 - Web Services: **JAX-RPC 1.0, JAXM 1.1, SAAJ 1.1, JAXR 1.0**
 - Aplicações Web: **Servlet API 2.3, JSP 1.2, JSTL 1.0**
- Implementação de referência
 - **Ferramentas de desenvolvimento:** Web Deploytool, Compilador JAXRPC (xrpcc), Jakarta Ant, Jakarta Tomcat, Registry Browser e Apache Xindice (banco de dados XML)
 - Serviços de **registro UDDI, roteamento SOAP e JAXRPC** (implementados como servlets no Tomcat)

xindice
(Zeen-dee-chay)



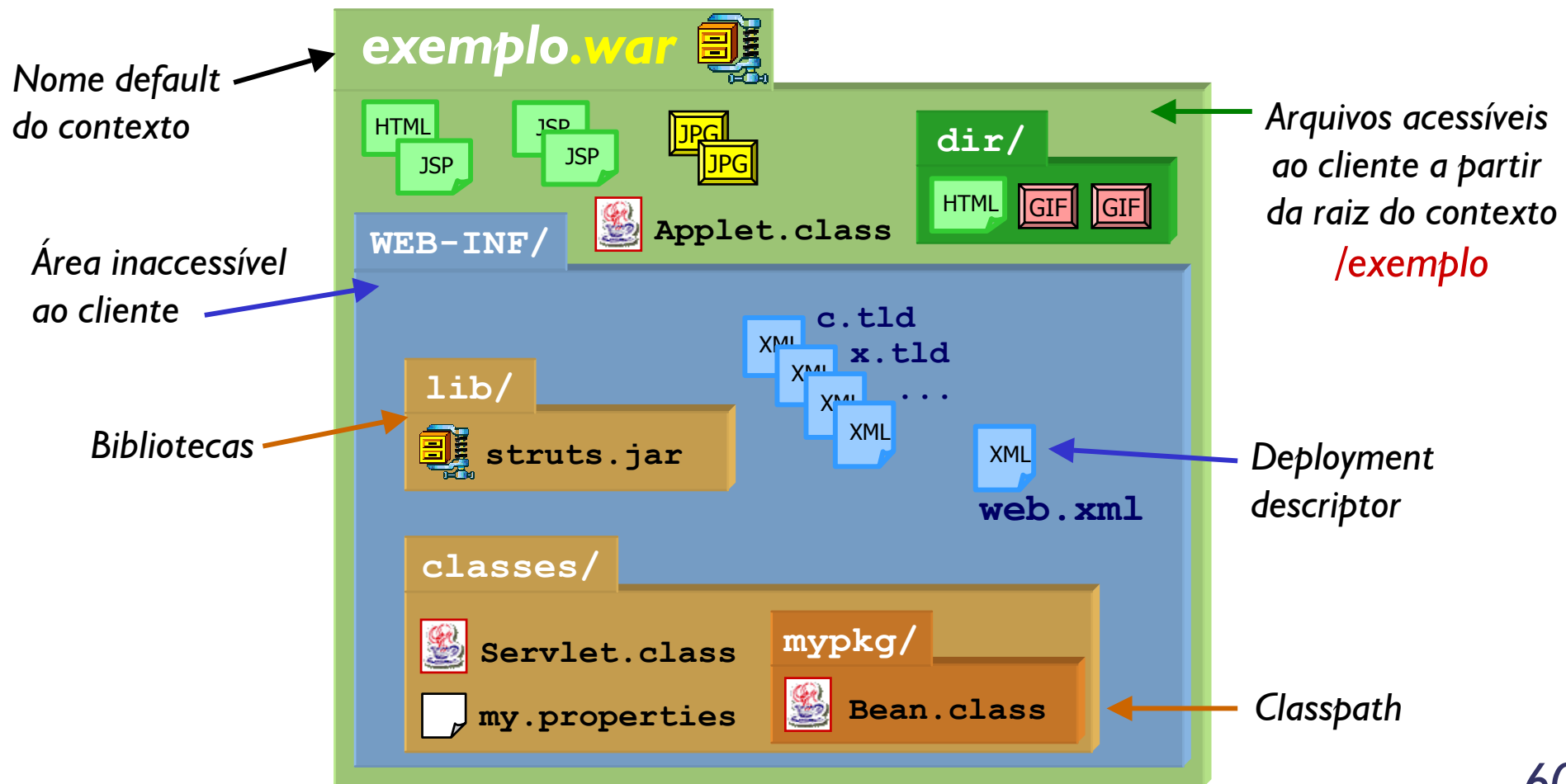
Aplicações Web em Java

- Web Services podem ser desenvolvidos em Java usando os pacotes **javax.servlet.*** que permitem criar
 - **Servlets**: componentes capazes de processar requisições HTTP e gerar respostas HTTP
 - **Páginas JSP**: documentos de texto (HTML, XML) que são transformados em servlets na instalação ou execução
 - **Bibliotecas de tags**: implementações que permitem o uso de XML no lugar do código Java em páginas JSP
- Deployment é muito simples
 - Escreva os servlets / JSPs que implementam Web Services
 - Escreva ou gere um deployment descriptor
 - Coloque tudo em um arquivo WAR
 - Instale o WAR no servidor (ex: copiar p/ pasta webapps/)

Estrutura de um arquivo WAR

- Aplicações Web são empacotadas em arquivos WAR para instalação automática em servidores J2EE

<http://servidor.com.br/exemplo>



Aplicações XML em Java

- APIs padrão no J2SDK e J2EE
 - **JAXP**: suporte a APIs para processamento XML: DOM, SAX e XSLT
- APIs padrão no Java Web Services Development Pack
 - **JAXM**, **JAX-RPC** e **SAAJ**: suporte a protocolos de comunicação baseados em XML
 - **JAXR**: suporte a sistemas de registro baseados em XML
- Padrões propostos (em desenvolvimento)
 - **JAXB** (JSR-31: XML data binding): suporte à serialização de objetos em XML
 - **JDOM** (JSR-102): outro modelo para processamento XML (que não usa a interface W3C DOM)
 - JSR-181: linguagem de **metadados** para Web Services

- **Java API for XML Processing**

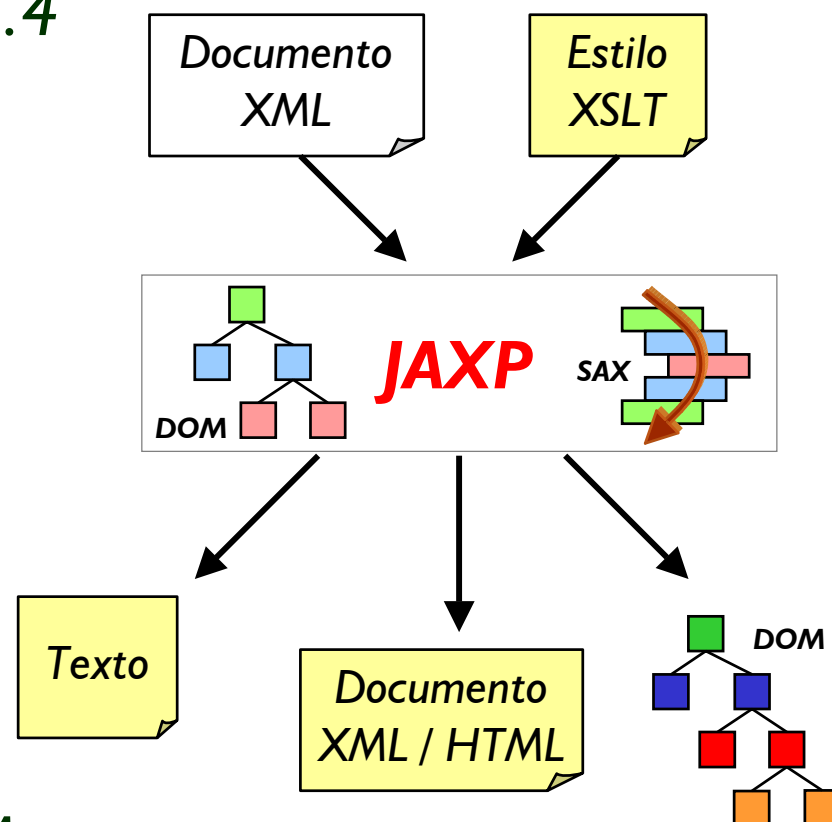
- Para leitura, criação, manipulação, transformação de XML
- Parte integrante do J2SDK 1.4

- **Pacotes**

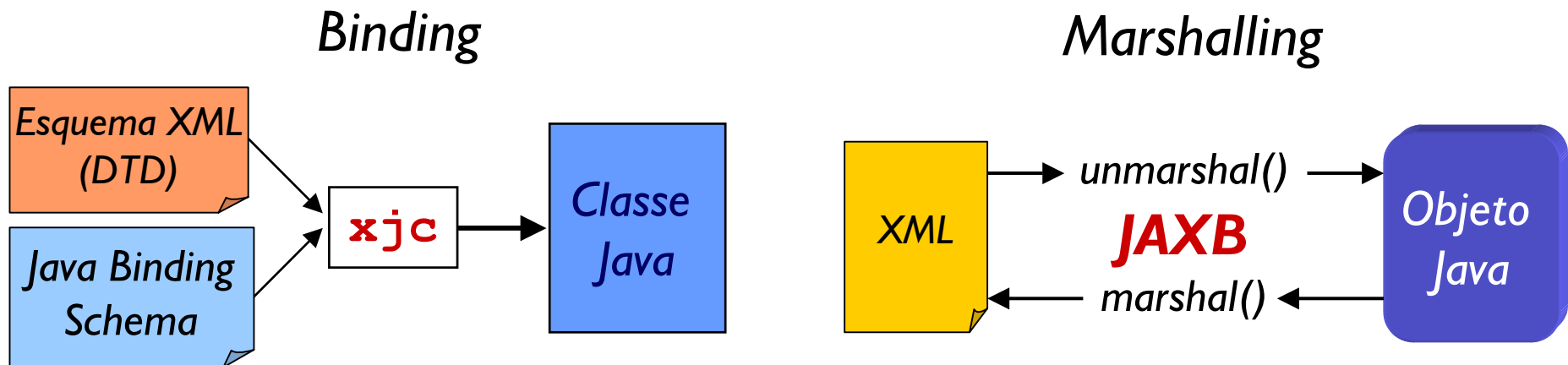
- `javax.xml.parsers`
- `javax.xml.transform.*`
- `org.w3c.dom`
- `org.w3c.sax.*`

- **Componentes**

- *Parsers para SAX e DOM*
- *Implementações em Java das APIs padrão SAX e DOM*
- *Implementações Java de API de transformação XSLT*

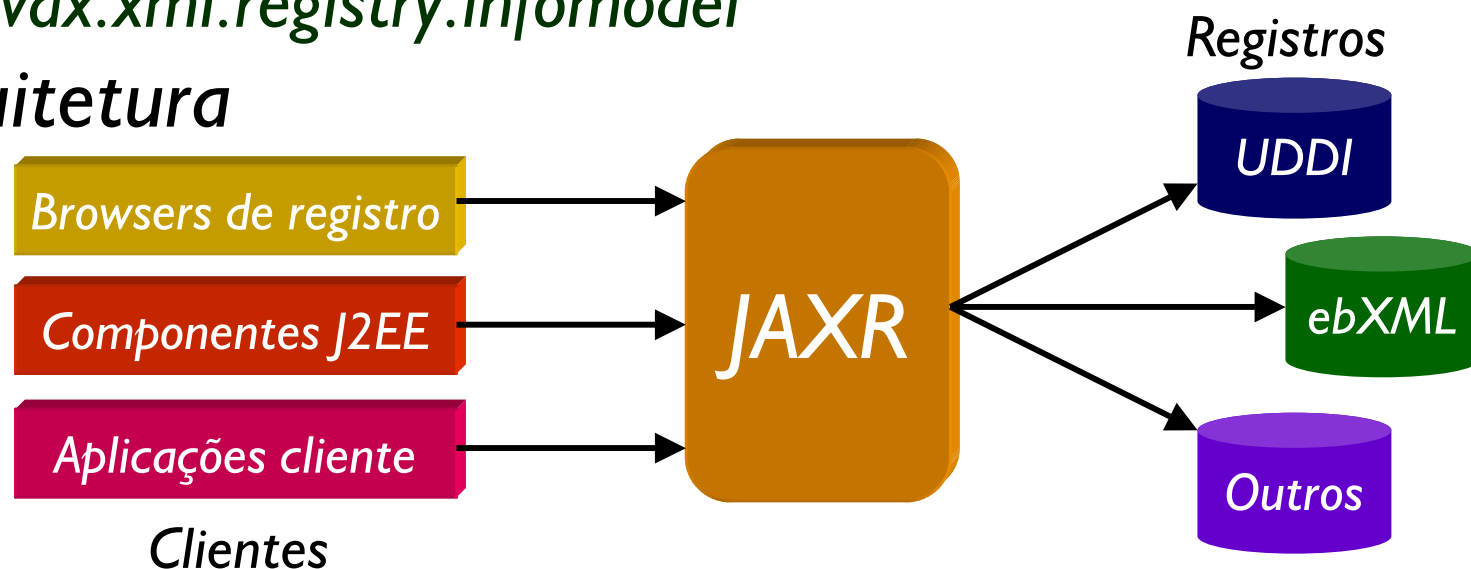


- **Java API for XML Binding (JSR-311)**
 - Mapeia classes Java a documentos XML
 - Permite gerar JavaBeans a partir de esquema XML
 - Permite serializar objetos para XML e vice-versa



- **Pacotes (community review jul-2002)**
 - `javax.xml.bind`
 - `javax.xml.marshall`
- **Em desenvolvimento há 3 anos (29/ago/1999).**

- **Java API for XML Registries**
 - Oferece acesso uniforme a diferentes sistemas de registro de serviços baseados em XML
 - Possui mapeamentos para **UDDI** e **ebXML**
 - Permite a inclusão e pesquisa de organizações, serviços
- **Pacotes**
 - `javax.xml.registry`
 - `javax.xml.registry.infomodel`
- **Arquitetura**

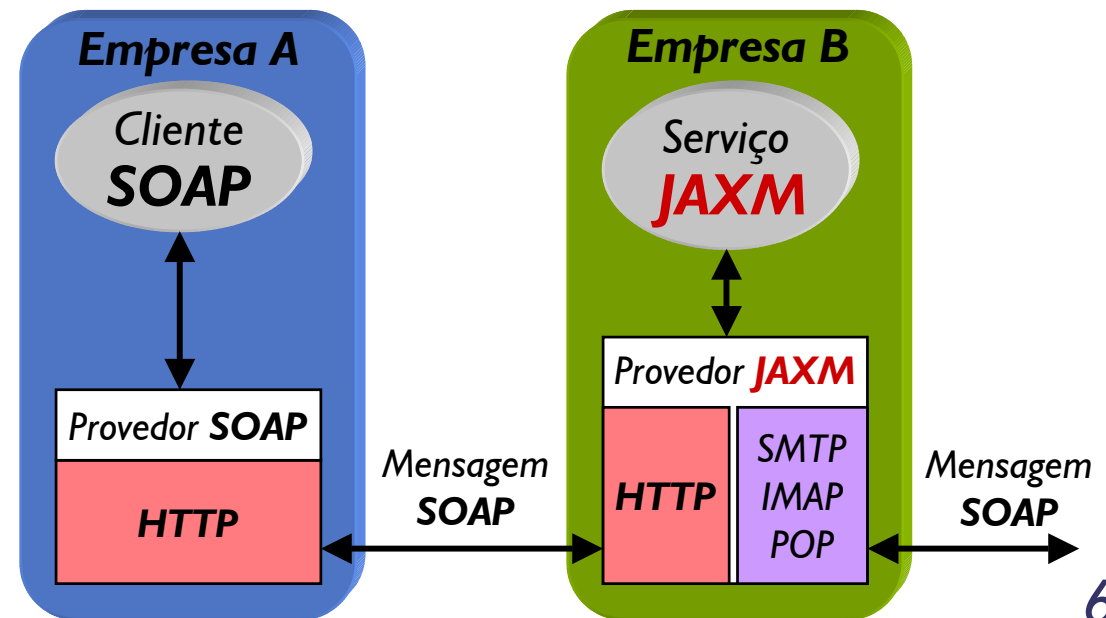


JAXM (e SAAJ)

- **Java API for XML Messaging** (e **SOAP with Attachments API for Java**)
 - Conjunto de APIs para manipular envelopes SOAP e transportá-los sobre HTTP, SMTP ou outros protocolos
 - Suporta comunicação baseada em **eventos** (mensagens) e baseada em **RPC** (par de mensagens requisição/resposta)
 - Suporta especificações SOAP 1.1 e SOAP with Attachments

- **Pacotes:**

- `javax.xml.soap`
- `javax.xml.messaging`
- `javax.xml.rpc.*`



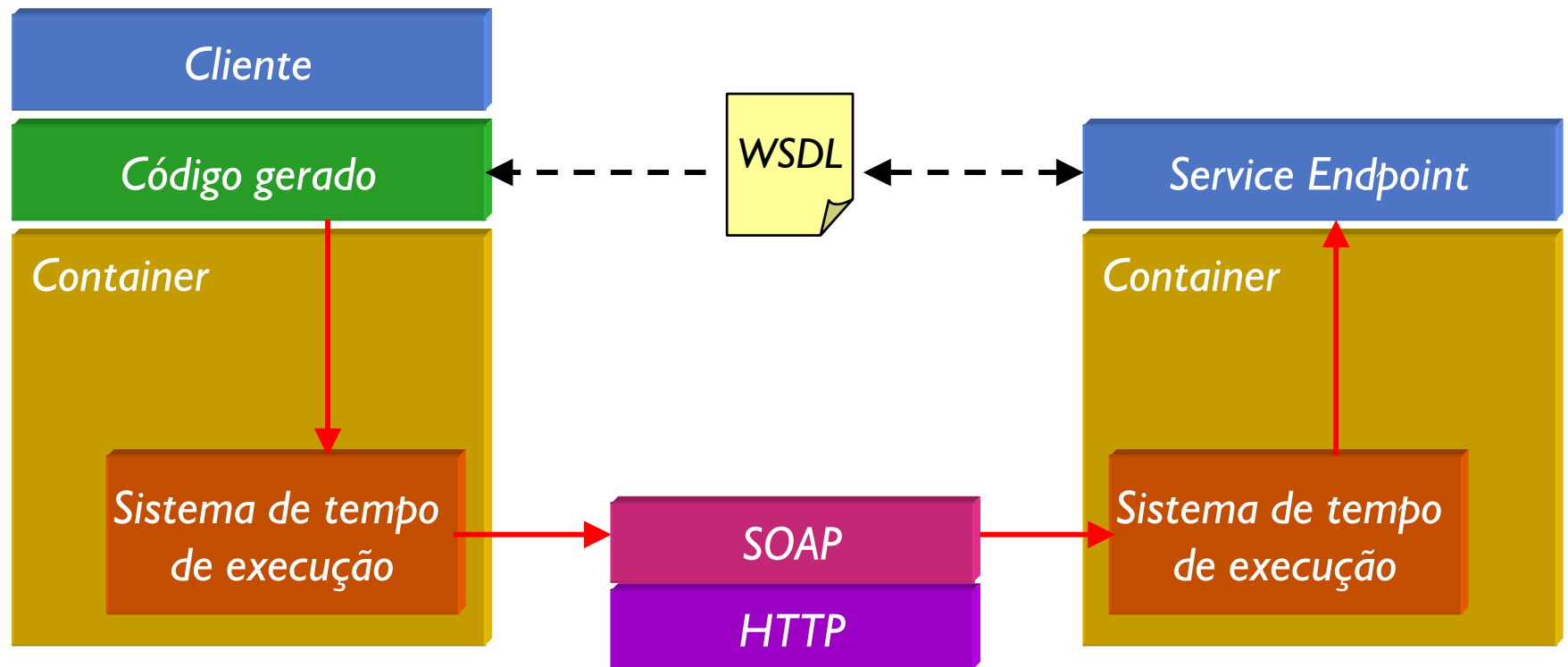
Fonte da ilustração:
JAXM 1.0 specification

- **Java API for XML-Based Remote Procedure Calls**
 - Um tipo de **Java RMI** sobre SOAP/HTTP
 - Alto nível de abstração **permite ignorar envelope SOAP**
 - Utiliza **WSDL** para **gerar** classes de servidor e cliente
- **Pacotes**
 - `javax.xml.rpc.*`
- **Desenvolvimento semelhante a RMI (simples e baseado em geração de código e container)**
 - **Escreve-se RMI, obtém-se SOAP e WSDL**
 - **Cliente pode obter interface para comunicação com o serviço dinamicamente, em tempo de execução**
 - **Stubs também podem ser gerados em tempo de compilação para maior performance**

JAXM vs. JAX-RPC

- Soluções diferentes para manipular mesmo envelope SOAP
 - JAX-RPC implementa **WSDL**. JAXM **não usa WSDL**.
 - JAXM manipula **mensagens** sem ligar para seu conteúdo
 - JAX-RPC usa WSDL para formato de **requisições** e **respostas**
 - JAXM **expõe** todos os detalhes do envelope; JAX-RPC **oculta**
 - Tudo o que se faz em JAX-RPC, pode-se fazer com JAXM
 - **RPC** é mais fácil com JAX-RPC; JAXM é API de baixo nível e pode ser usada tanto para **messaging** ou RPC
 - Cliente e serviço JAX-RPC rodam em **container**
- Conclusão
 - Use **JAX-RPC** para criar aplicações SOAP-RPC com WSDL
 - Use **JAXM** para messaging ou quando precisar manipular o envelope SOAP diretamente

Arquitetura JAX-RPC



Criação de um Web Service com JAX-RPC (I)

1. Escrever uma interface RMI para o serviço

```
package example.service;  
  
public interface BookstoreIF extends java.rmi.Remote {  
    public BigDecimal getPrice(String isbn)  
        throws java.rmi.RemoteException;  
}
```

2. Implementar a interface

```
package example.service;  
  
public class BookstoreImpl implements BookstoreIF {  
    private BookstoreDB database = DB.getInstance();  
  
    public BigDecimal getPrice(String isbn) {  
        return database.selectPrice(isbn);  
    }  
}
```

Criação de um Web Service com JAX-RPC (2)

3. Escrever arquivo de configuração*

```
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">  
  <service name="BookstoreService" targetNamespace="http://mybooks.org/wsdl"  
          typeNameNamespace="http://mybooks.org/types"  
          packageName="example.service">  
    <interface name="example.service.BookstoreIF"  
              servantName="example.service.BookstoreImpl"/>  
  </service>  
</configuration>
```

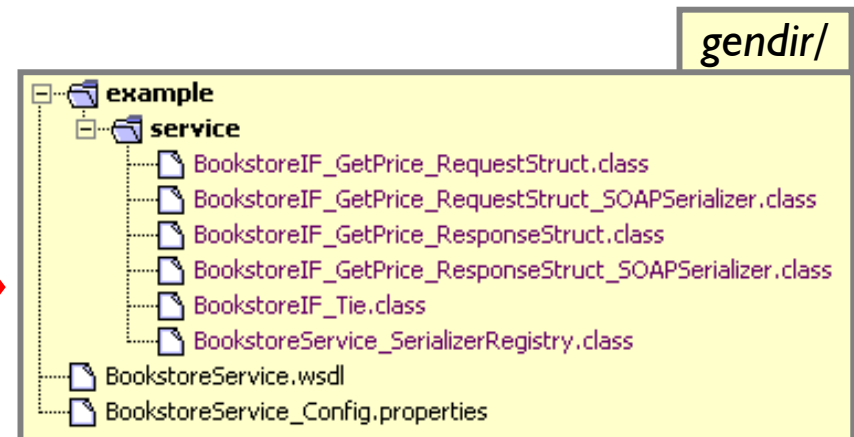
config_rmi.xml

4. Compilar classes e interfaces RMI

```
> javac -d mydir BookstoreIF.java BookstoreImpl.java
```

5. Gerar código do servidor

```
> xrpc -classpath mydir  
      -server -keep  
      -d gendir  
      config_rmi.xml
```



* Não faz parte da especificação - procedimento pode mudar no futuro

Criação de um Web Service com JAX-RPC (3)

■ 6. Criar web deployment descriptor *web.xml*

```
<web-app>
  <servlet>
    <servlet-name>JAXRPCEndpoint</servlet-name>
    <servlet-class>
      com.sun.xml.rpc.server.http.JAXRPCServlet
    </servlet-class>
    <init-param>
      <param-name>configuration.file</param-name>
      <param-value>
        /WEB-INF/BookstoreService_Config.properties
      </param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>JAXRPCEndpoint</servlet-name>
    <url-pattern>/bookpoint/*</url-pattern>
  </servlet-mapping>
</web-app>
```

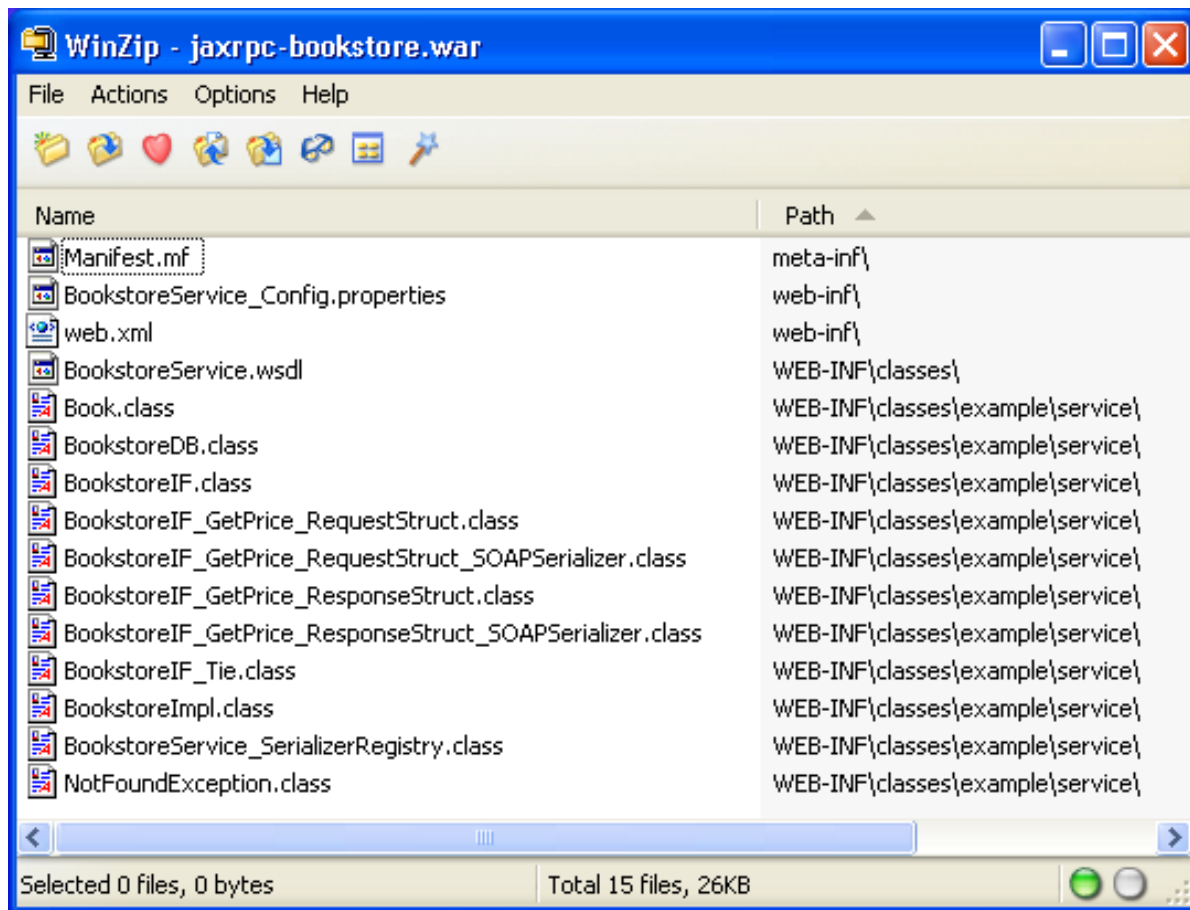
Nosso "container" →

Nome do arquivo gerado pelo xrpcc →

subcontexto que será o endpoint do serviço →

Criação de um Web Service com JAX-RPC (4)

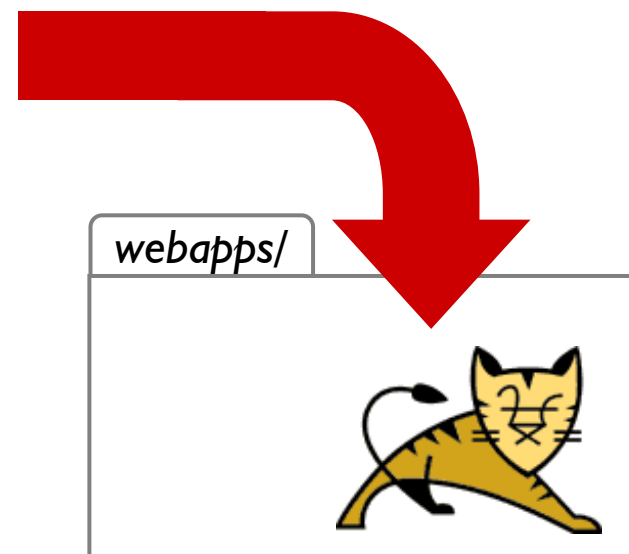
7. Colocar tudo em um WAR



jaxrpc-bookstore.war

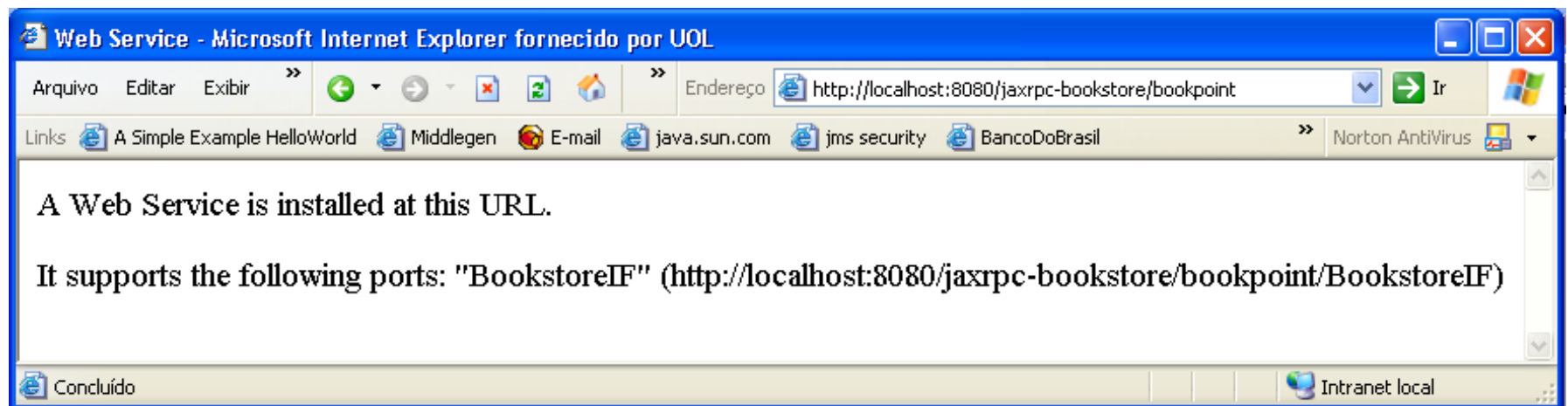
8. Deployment no servidor

- Copiar arquivo para diretório webapps do Tomcat

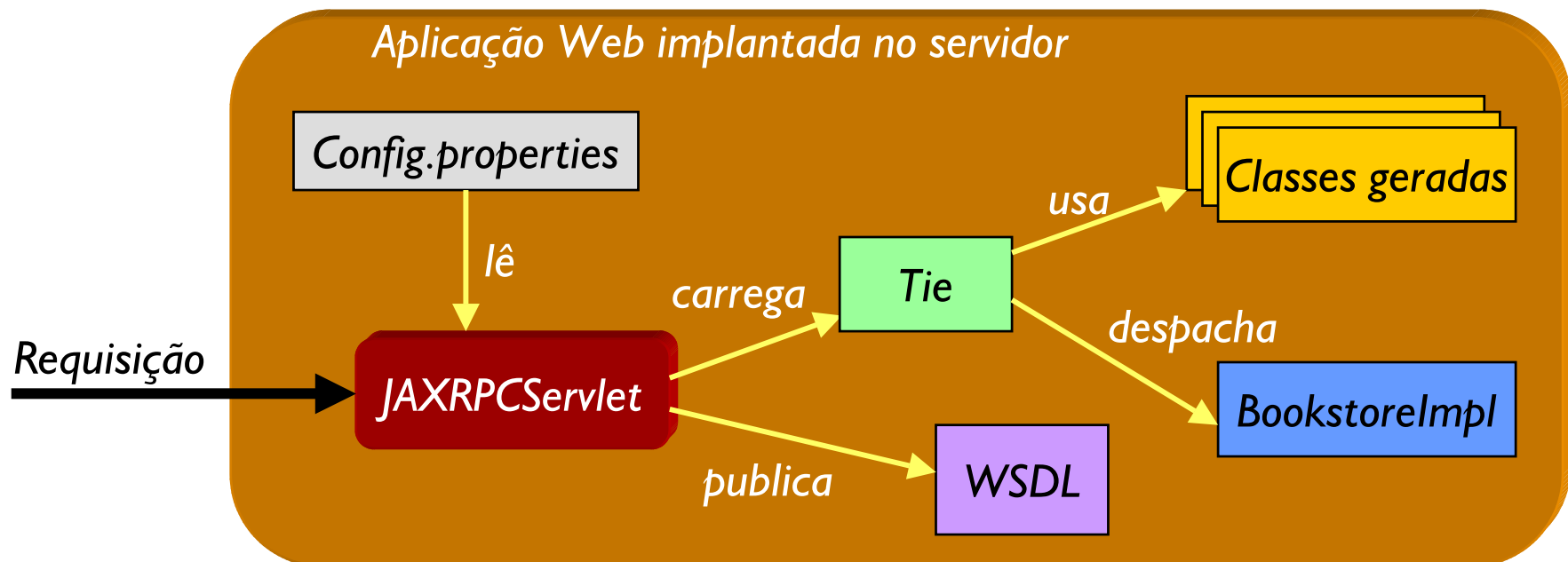


Construção e instalação do serviço com o Ant

- Script do Ant para compilar as classes RMI, compilá-las com **xrjcc**, gerar o WSDL, empacotar no WAR e copiar para o diretório `webapps/` do Tomcat
 - > **ant BUILD.ALL.and.DEPLOY**
- Teste para saber se o serviço está no ar
 - Inicie o Tomcat do JWSDP
 - Acesse: **<http://localhost:8080/jaxrpc-bookstore/bookpoint>**



- O endpoint do serviço na implementação de referência JWSDP 1.0 é um **servlet**
`com.sun.xml.rpc.server.http.JAXRPCServlet`
 - Próximas versões (e J2EE 1.4) devem oferecer implementação em **stateless session bean**
- Servlet é ponto de entrada para todas as requisições

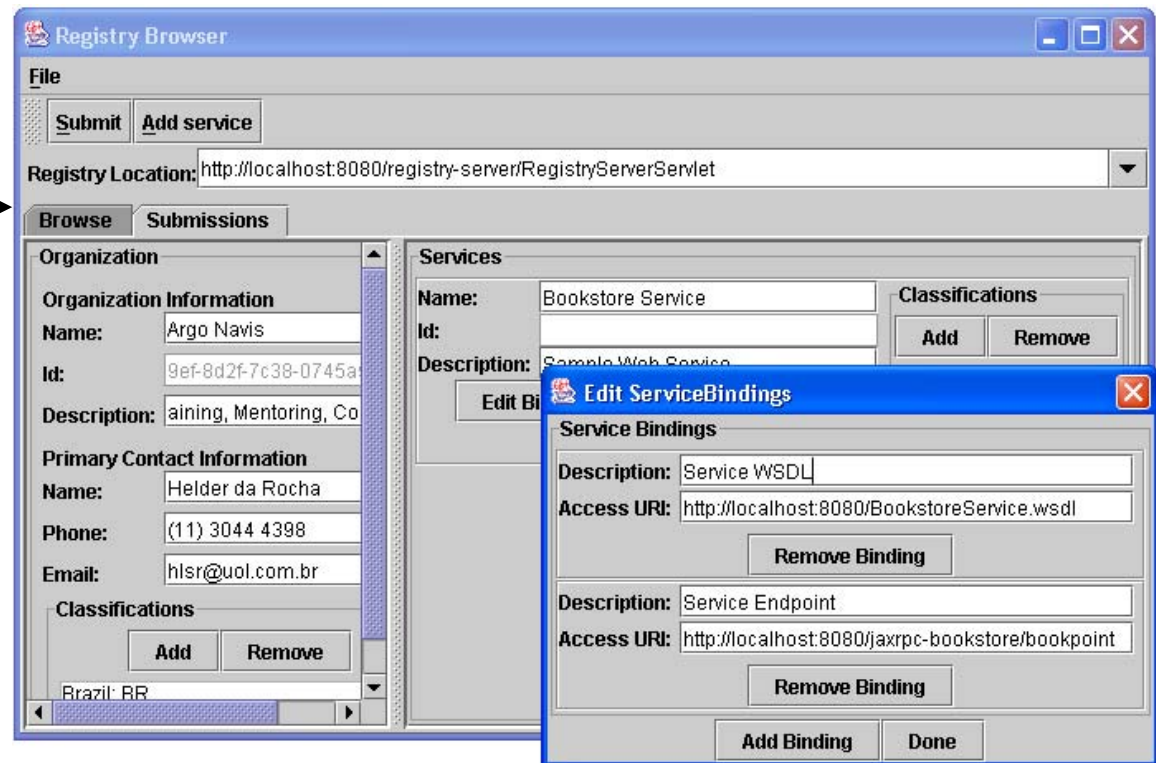


Registro do serviço

- Podemos registrar o nosso Web Service
 - Automaticamente executando um cliente (ant REGISTER)
 - Interativamente usando o Registry Browser
- Para usar o servidor UDDI do JWSDP
 - 1. inicie o Xindice
 - 2. inicie o Tomcat

Registry Browser

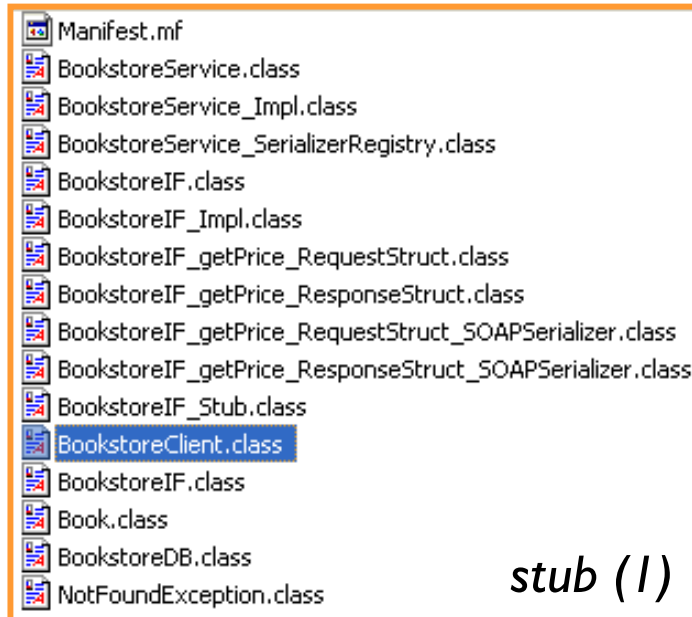
1. Selecione a localização do servidor (*http://localhost/...*)
2. Crie uma nova organização
3. Crie novo serviço
4. Em "edit bindings" coloque URLs dos serviços
5. Aperte *submit*. Use "testuser" como nome e senha



- Há três tipos de cliente JAX-RPC:
 - 1. *Cliente estático tipo-RMI*: usa stubs gerados em **tempo de compilação** para se comunicar com o servidor e chama métodos do serviço remoto como se fossem locais
 - 2. *Cliente WSDL de interface dinâmica (DII)*: descobre a interface de comunicação em **tempo de execução** e chama métodos via mecanismo similar a Java reflection
 - 3. *Cliente WSDL de interface estática*: usa interface Java implementada por stubs gerados em **tempo de execução** e chama métodos remotos como se fossem locais
- Clientes precisam aderir ao contrato com o Web Service (WSDL) mas podem ser implementados e usados com ou sem WSDL

Cientes JAX-RPC

Ciente de implementação estática



+ performance, + acoplamento

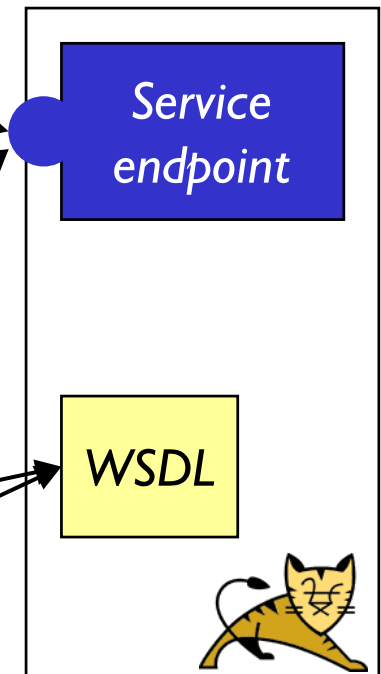
2. Chama o serviço

Cientes de implementação dinâmica



1. Obtém informações sobre o serviço

- performance, - acoplamento



Cientes JAX-RPC (detalhes)

■ 1) Cliente com stub estático

```
Stub stub = (Stub)(new BookstoreService_Impl().getBookstoreIFPort());
stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, epointURL);
BookstoreIF proxy = (BookstoreIF)stub;
System.out.println("Price R$ " + proxy.getPrice("2877142566"));
```

■ Cliente com interface dinâmica (DII)

```
Service srv = factory.createService(new URL(wsdlURL),
                                     new QName(NS, "BookService"));
Call call = srv.createCall( new QName(NS, "BookstoreIFPort") );
call.setOperationName(new QName(NS, "getPrice"));
BigDecimal pr = (BigDecimal)call.invoke(new String[] {"2877142566"});
System.out.println("Price R$ " + pr);
```

■ Cliente com stub dinâmico (proxy)

```
Service srv = factory.createService(new URL(wsdlURL),
                                     new QName(NS, "BookService"));
BookstoreIF proxy = (BookstoreIF)
    srv.getPort(new QName(NS, "BookstoreIFPort"), BookstoreIF.class);
System.out.println("Price R$ " + proxy.getPrice("2877142566"));
```

- *Para gerar os clientes*
 - *Cliente (1): gere stubs com `xrpsc -client` e arquivo WSDL (use `config_wsdl.xml`) e depois compile classe do cliente*
 - *Cientes (2) e (3): apenas compile a classe do cliente*
- *Script do Ant para compilar os três clientes e colocar as classes em um JAR*

```
> ant client.BUILD
```

- *Para rodar o cliente e executar o Web Service*

```
> ant dynamic-client.RUN
```

```
Buildfile: build.xml
```

```
dynamic-client.RUN:
```

```
    [java] ISBN 2877142566. Price R$ 19.50
```

```
BUILD SUCCESSFUL
```

- Nesta palestra apresentamos a arquitetura de **Web Services**, suas tecnologias fundamentais SOAP, WSDL e UDDI e as APIs Java que as implementam.
- Java oferece **APIs** que permitem desde a **manipulação direta** de XML (DOM e SAX) até a criação de Web Services **sem contato** com XML (JAX-RPC)
- JAX-RPC é a forma mais **fácil** e **rápida** de criar Web Services em Java
- Serviços desenvolvidos em JAX-RPC poderão ser acessados de aplicações .NET e vice-versa.
 - Web Services viabilizam a integração de serviços entre plataformas diferentes: **interoperabilidade!**

- [1] JSR-101 Expert Group. *Java™ API for XML-based RPC: JAX-RPC 1.0 Specification*. Java Community Process: www.jcp.org.
- [2] Sun Microsystems. *Java™ Web Services Tutorial*. java.sun.com/webservices/. *Coleção de tutoriais sobre XML, JSP, servlets, Tomcat, SOAP, JAX-RPC, JAXM, etc.*
- [3] JSR-109 Expert Group. *Web Services for J2EE 1.0 (Public Draft 15/04/2002)*. Java Community Process: www.jcp.org. *Descreve o suporte a Web Services em J2EE 1.3*
- [4] Nicholas Kasseem et al. (JSR-67). *Java™ API for XML Messaging (JAXM) e Soap with Attachments API for Java 1.1*. java.sun.com. *Modelo de programação de baixo nível (lida diretamente com SOAP enquanto JAX-RPC esconde) e mais abrangente.*
- [5] Roberto Chinnici. *Implementing Web Services with the Java™ Web Services Development Pack*. JavaONE Session 1777. java.sun.com/javaone. *Apresentação que oferece uma visão geral de JAX-RPC e o Web Services Development Pack da Sun.*
- [6] Brett McLaughlin. *Java & XML 2nd. Edition*. O'Reilly and Associates, 2001. *Explora as APIs Java para XML e oferece uma introdução à programação de WebServices em Java*
- [7] Ethan Cerami. *Web Services Essentials*. O'Reilly, Fev 2002. *XML-RPC, SOAP, UDDI e WSDL são explorados de forma didática e exemplos são implementados em Java usando ferramentas open-source.*
- [8] W3C Web Services Activity. <http://www.w3.org/2002/ws/>. *Página que dá acesso aos grupos de trabalho que desenvolvem especificações de SOAP (XMLP), WSDL e Arquitetura*

- [9] *Apache XML Project*. xml.apache.org. *Duas implementações de SOAP e uma implementação de XML-RPC em Java.*
- [10] *IBM Developerworks Open Source Projects*. <http://www-124.ibm.com/>. *Implementações UDDI4J e WSDL4J.*
- [11] Al Saganich. *Java and Web Services Primer*. O'Reilly Network 2001. <http://www.onjava.com/pub/a/onjava/2001/08/07/webservices.html>. *Ótimo tutorial sobre Web Services.*
- [12] Al Saganich. *Hangin' with the JAX Pack. Part 1: JAXP and JAXB, Part 2: JAXM, Part 3: Registries (JAXR), Part 4: JAX-RPC*. O'Reilly Network 2001-2002. <http://www.onjava.com/pub/a/onjava/2001/11/07/jax.html> *Esta série de quatro artigos publicados entre nov/2001 e abr/2002 é talvez o melhor ponto de partida para quem deseja aprender a usar as APIs Java para Web Services.*
- [13] David Chappell, Tyler Jewel. *Java Web Services*. O'Reilly and Associates, Mar 2002. *Explora implementações Java de Apache SOAP, WSDL e UDDI em Java. Tem um capítulo dedicado às APIs do JWSDP.*
- [14] Al Saganich. *JSR-109 Web Services inside of J2EE Apps*. O'Reilly Network, Aug 2002. <http://www.onjava.com/pub/a/onjava/2002/08/07/j2eewebsvs.html> *Mostra um resumo da proposta do JSR-109, que prevê a integração J2EE-Web Services.*

helder@argonavis.com.br

www.argonavis.com.br

Palestra: Como Implementar Web Services em Java

COMDEX 2002, São Paulo

Curso: XML 100 - Introdução a XML

© 2001, 2002, Helder da Rocha