Intermidia

Processamento de XML

Renata Pontin M. Fortes

(renata@icmc.usp.br)

PAE: Willian Watanabe (watinha@gmail.com)

Instituto de Ciências Matemáticas e de Computação

ICMC-USP S.Carlos, 2010

Processar XML

Finalidade da XML (Revisão)

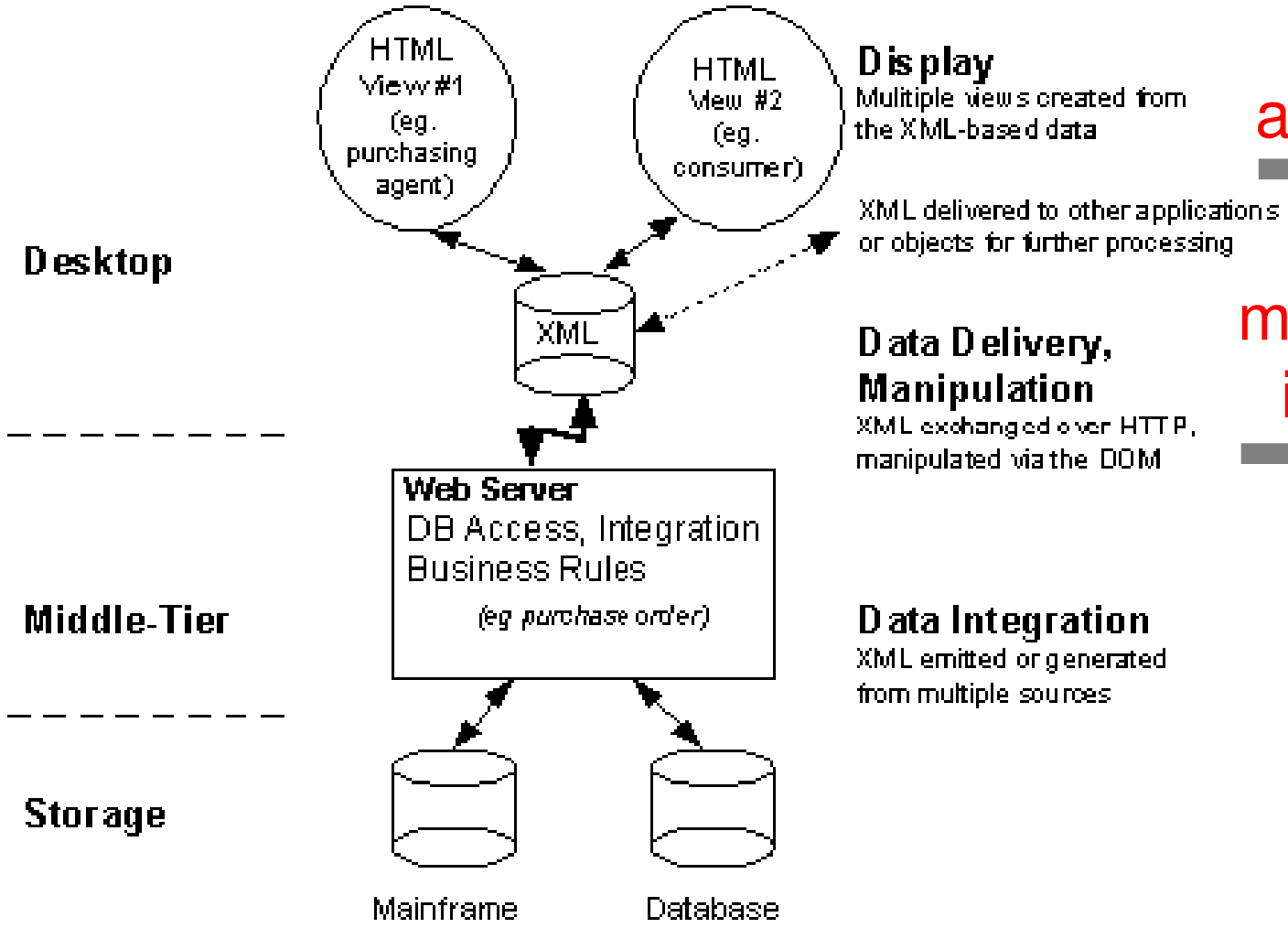
- Codificar (*mark up*) dados somente uma vez
- Gerar diversos produtos a partir dessa marcação
- Permitir busca complexa (semanticamente)
- Reusar dados (no todo ou em parte) diversas vezes
- Realizar intercâmbio de dados
- Permitir comunicação entre computadores

Arquitetura XML

- XML e suas tecnologias relacionadas oferecem uma arquitetura robusta para integração, manipulação, intercâmbio e apresentação de documentos

- Arquitetura de 3 camadas, baseada nas seguintes recomendações do W3C:
  - XML 1.0
  - XML namespaces
  - **API XML - (DOM ou SAX)**
  - XSL - XSLT - XPATH

Arquitetura XML



apresentação

manipulação e intercâmbio

integração

XML – integração de dados

- Utilizando **Namespaces** é possível integrar dados de diferentes origens, permitindo sua validação através de XML Schemas
- XML Namespaces possibilitam:
  - A definição de dicionários locais não ambíguos
  - A utilização de dicionários disponíveis publicamente.
  - Ex.: Utilização do atributo *dt*

```
<data dt:dt="date"
xmlns:dt=urn:schemas-microsoft-com:datatypes">1997-03-17</data>
```

XML – namespaces

- oferecem um método para evitar conflitos entre nomes de elementos.

- possibilitam definir nomes de elementos e atributos XML, associando-os com referências URI.

Conflitos de nomes

Documento XML que contém informações em uma **tabela**:

```
<table>
<tr>
   <td>laranjas</td>
   <td>bananas</td>
</tr>
</table>
```

Documento XML que contém informações sobre uma **mesa**:

```
<table>
<name>Mesa</name>
<width>80</width>
<length>120</length>
</table>
```

Resolvendo problema...

- Prefixos
- Namespaces

Prefixos

```
<h:table>
    <h:tr>
      <h:td>laranjas</h:td>

      <h:td>bananas</h:td>
    </h:tr>
</h:table>
```

```
<f:table>
    <f:name>Mesa</f:name>
    <f:width>80</f:width>
    <f:length>120</f:length>
</f:table>
```

Namespaces

```
<h:table
xmlns:h="http://www.w3.com/TR/html4">
    <h:tr>
        <h:td>laranjas</h:td>
        <h:td>bananas</h:td>
    </h:tr>
</h:table>
```

```
<f:table
xmlns:f="http://www.w3schools.com/fur">
    <f:name>Mesa</f:name>
    <f:width>80</f:width>
    <f:length>120</f:length>
</f:table>
```

Atributo de *Namespace*

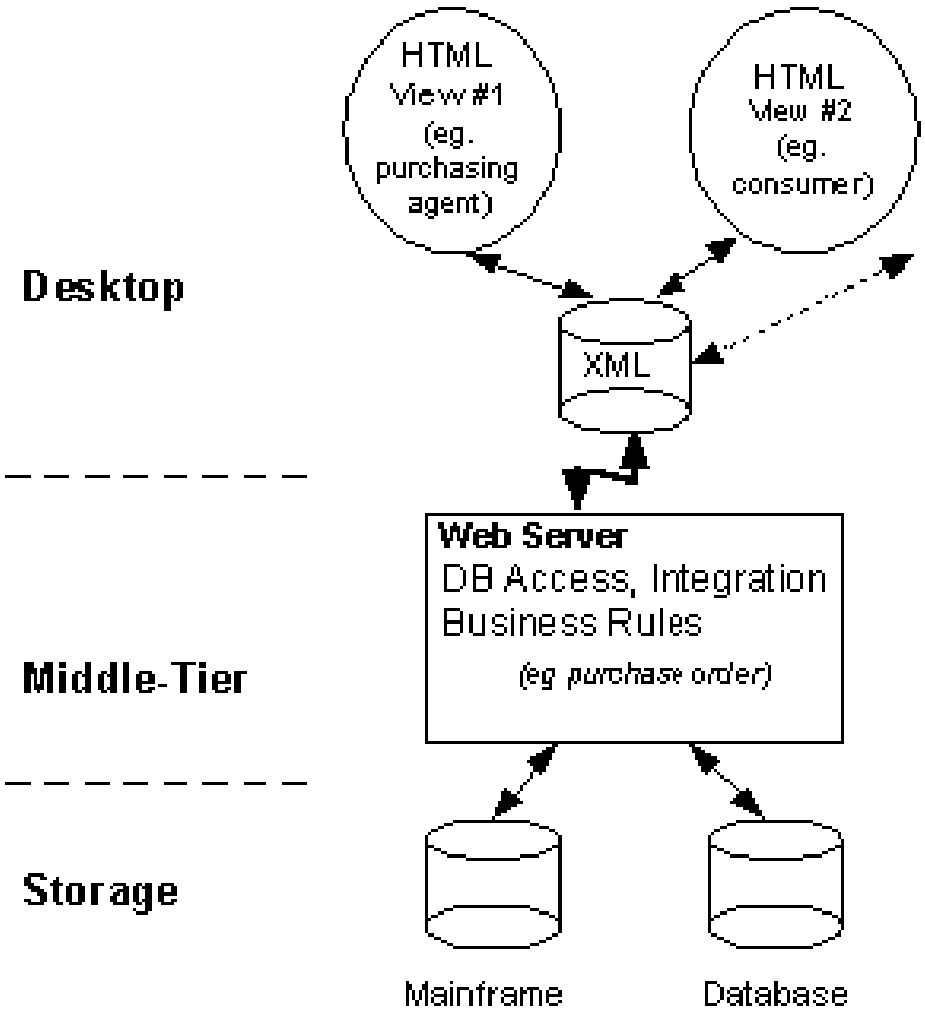
- É colocado na start-tag de um elemento e possui a sintaxe:

  `xmlns:namespace-prefix="namespace"`

- A especificação de namespaces do W3C estabelece que o namespace deveria ser um URI (*Uniform Resource Identifier*).

Repare que o endereço utilizado para identificar o namespace não é usado pelo parser para consultar a informação. O único propósito é dar ao namespace um nome único. Entretanto, muito frequentemente as empresas utilizam o namespace como um ponteiro para uma página web real, contendo informações sobre o namespace

Arquitetura XML



apresentação

manipulação e intercâmbio

integração

XML *Intercâmbio e Manipulação*

- **Intercâmbio**
  - Através do protocolo HTTP
  - Pode utilizar agentes para comunicação bidirecional cliente/servidor
- **Manipulação**
  - Feita através de um processador XML (**API DOM** ou **SAX**) geralmente embutido em um *parser*
  - *Parser* pode:
    - Apresentar documento, usando CSS ou XSL
    - Disponibilizar dados para serem utilizados por um script
    - Repassar documento para ser processado por outra aplicação

Processar XML

XML não faz nada!

- XML é um formato de dados
- Software pode fazer muito com marcações
- Processadores XML:
  - Simple API for XML (**SAX**)
  - Document Object Model (**DOM**)

XML *Processador XML*

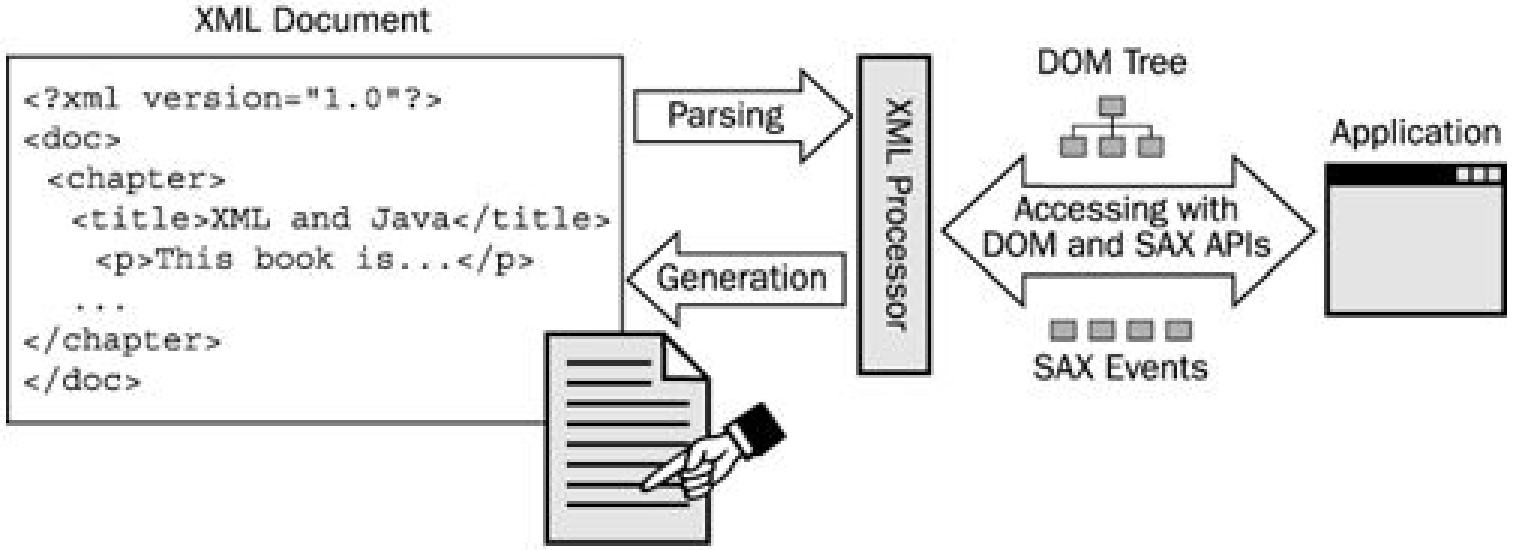
- Módulo de software capaz de ler um documento XML e prover acesso a seu conteúdo e sua estrutura.
- **Implementação** de processadores é desnecessária
  - Existem diversos processadores publicamente disponíveis
- Necessário => Disponibilização de interfaces que permitam a interação com os processadores
  - **APIs XML**

- **Dois tipos:**
  - Baseadas em árvore
    - Processa um documento XML e provê acesso a ele através de um modelo hierárquico de objetos baseado em árvore
    - Ex: **DOM** - Document Object Model
  - Baseadas em eventos
    - Processa um documento XML através da definição de uma interface de eventos usada para servir itens do documento a aplicações à medida que estes vão sendo processados
    - Ex: **SAX -** Simple API for XML

Processar XML

Processador XML

- Módulo de software para ler documentos XML e fornecer acesso ao seu conteúdo e estrutura a uma aplicação



Processar XML

Exemplos

- Apache XML Project
  - http://xml.apache.org/

- Xerces Java Parser
  - Xerces-J-bin.1.4.3.zip

XML Processing: APIs

Document Object Model (DOM)

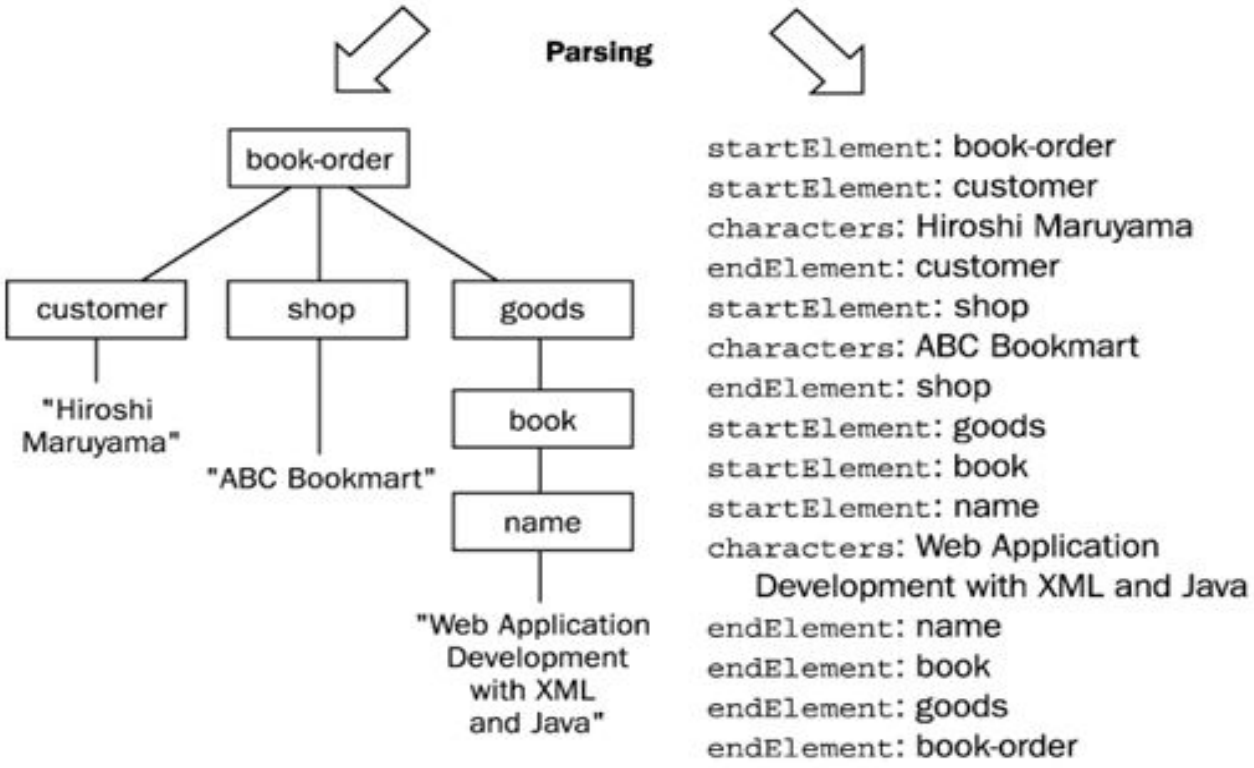
- Tree structure-based API
- W3C
- Level 0: Estrutura equivalente ao HTML
- Level 1 (1998): Inclui recursos para navegação e manipulação dos componentes
- Level 2 (2000): Inclui style sheets e eventos
- Level 3 (2004)

Simple API for XML (SAX)

- Event-driven API
- David Megginson e diversas pessoas na lista xml-dev
- Padrão “de facto”

Processar XML

```
<?xml version="1.0"encoding="utf-8"?>
<book-order>
 <customer>Hiroshi Maruyama</customer>
 <shop>ABC Bookmart</shop>
 <goods>
  <book>
   <name>Web Application Development with
XML and Java</name>
  </book>
 </goods>
</book-order>
```



Gerar uma stream de entrada de dados segundo a API do SAX

Processando XML: SAX

SAX: Event-Driven API

- Um documento XML é processado (*parsing*) em um único passo e a seqüência de eventos é notificada para uma aplicação

- Exemplo de eventos: leitura de uma tag de início; ocorrência de um atributo; existência de texto

- Uma aplicação pode registrar ***handlers*** de eventos

- Um processador XML baseado em SAX notifica os ***handlers*** de eventos

Processando XML: SAX

Interface SAX

- A especificação SAX define uma interface em termos de um conjunto de *handlers* de eventos

- Projetada como **uma API leve** que não gera uma estrutura em árvore de um documento XML

- Uma aplicação deve implementar os métodos da interface associados aos eventos que devem ser tratados

XML *SAX*

Funcionamento

- Disponibiliza a informação contida em um documento XML através de uma seqüência de eventos
- Para utilizar SAX é necessário:
  - Definir um modelo de objetos próprio
  - Criar uma classe que “escuta” eventos SAX e gera o modelo de objetos definido
- **SAX** => realmente simples!
  - O parser apenas gera eventos.
    - Cabe à aplicação interpretar esses eventos

Processando XML: SAX

- Uma interface Java simples
  - Um processador SAX é uma classe que implementa a interface ***org.xml.sax.Parser***
  - Ao percorrer a árvore de nós do documento XML, o parser realiza chamadas para os métodos implementados pela aplicação

XML *SAX*

- **Vantagens**
  - Velocidade
    - O processo de parsing é bem mais rápido graças à simplicidade da API
    - O documento não necessita estar totalmente na memória para que seu processamento seja iniciado
  - Não impõe um modelo de objetos
- **Desvantagem**
  - Oferece poucas facilidades às aplicações

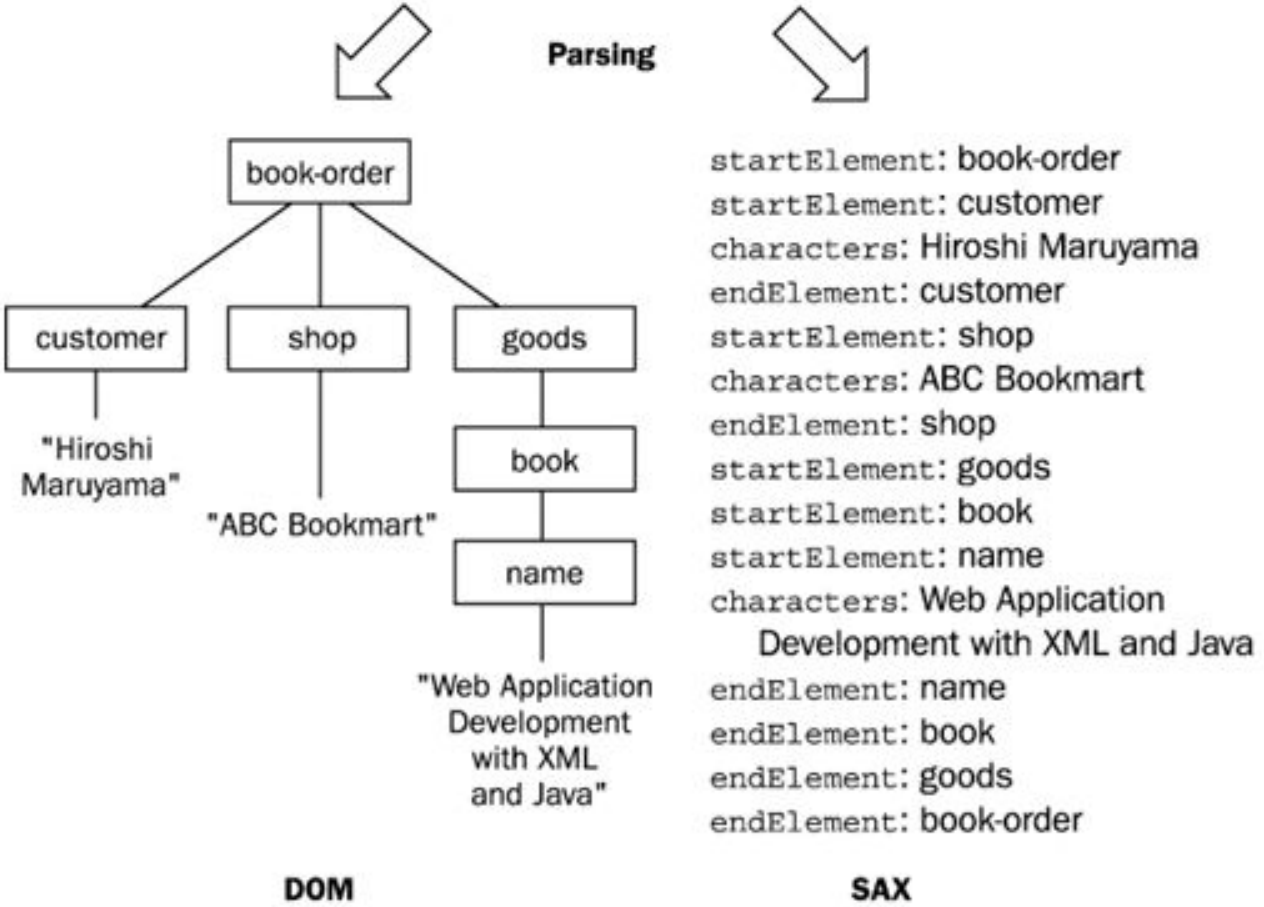
XML *SAX*

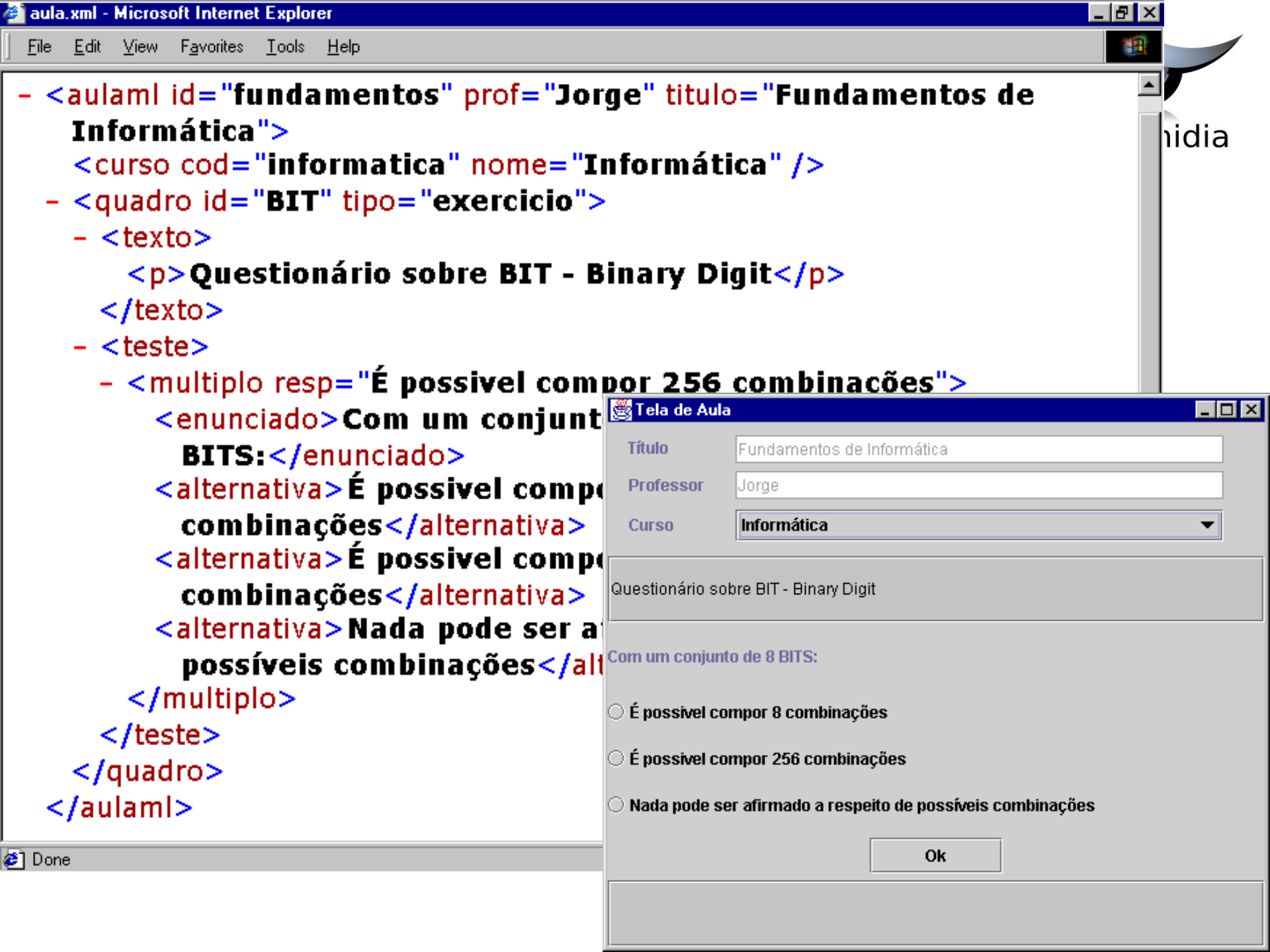
Nome do método	Descrição
startDocument()	Recebe a notificação de início do documento.
endDocument()	Recebe a notificação de final do documento.
startElement(String uri, String localpart, String name, Attributes amap)	Recebe a notificação de início de um elemento.
endElement(String name)	Recebe a notificação de final de um elemento.
characters(char ch[], int start, int length)	Recebe a notificação de dados (texto).
ignorableWhitespace(char ch[],int start, int length)	Recebe a notificação de espaço em branco no conteúdo do elemento.
processingInstruction(String target, String data)	Recebe a notificação de uma instrução de processamento.
setDocumentLocator(Locator locator)	Recebe um objeto para localizar a origem dos eventos do documento. O objeto *Locator* fornece informação sobre a localização do evento, por exemplo, o número da linha e a

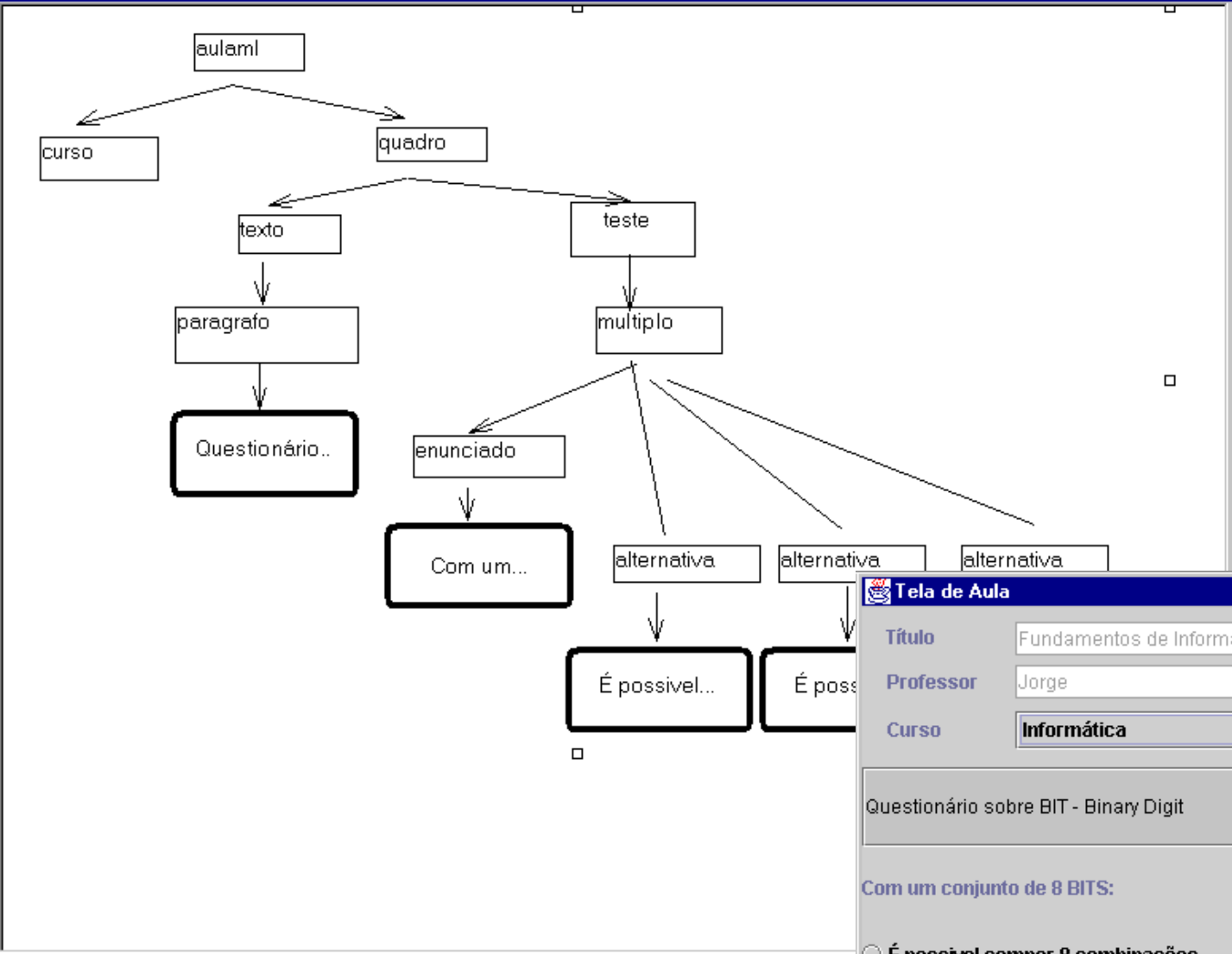
Processando XML

DOM
versus
SAX

```
<?xml version="1.0"encoding="utf-8"?>
<book-order>
 <customer>Hiroshi Maruyama</customer>
 <shop>ABC Bookmart</shop>
 <goods>
  <book>
   <name>Web Application Development with
XML and Java</name>
  </book>
 </goods>
</book-order>
```




aula.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
- <aulaml id="fundamentos" prof="Jorge" titulo="Fundamentos de Informática">
<curso cod="informatica" nome="Informática" />
- <quadro id="BIT" tipo="exercicio">
- <texto>
Questionário sobre BIT - Binary Digit
</texto>
- <teste>
- <multiplo resp="É possivel compor 256 combinacões">
<enunciado>Com um conjunt
BITS:</enunciado>
<alternativa>É possivel compo
combinações</alternativa>
<alternativa>É possivel compo
combinações</alternativa>
<alternativa>Nada pode ser a
possíveis combinações</alt
</multiplo>
</teste>
</quadro>
</aulaml>
Done
Tela de Aula
Título
Fundamentos de Informática
Professor
Jorge
Curso
Informática
Questionário sobre BIT - Binary Digit
Com um conjunto de 8 BITS:
É possivel compor 8 combinações
É possivel compor 256 combinações
Nada pode ser afirmado a respeito de possíveis combinações
Ok



Intermidia

Tela de Aula

Título: Fundamentos de Informática

Professor: Jorge

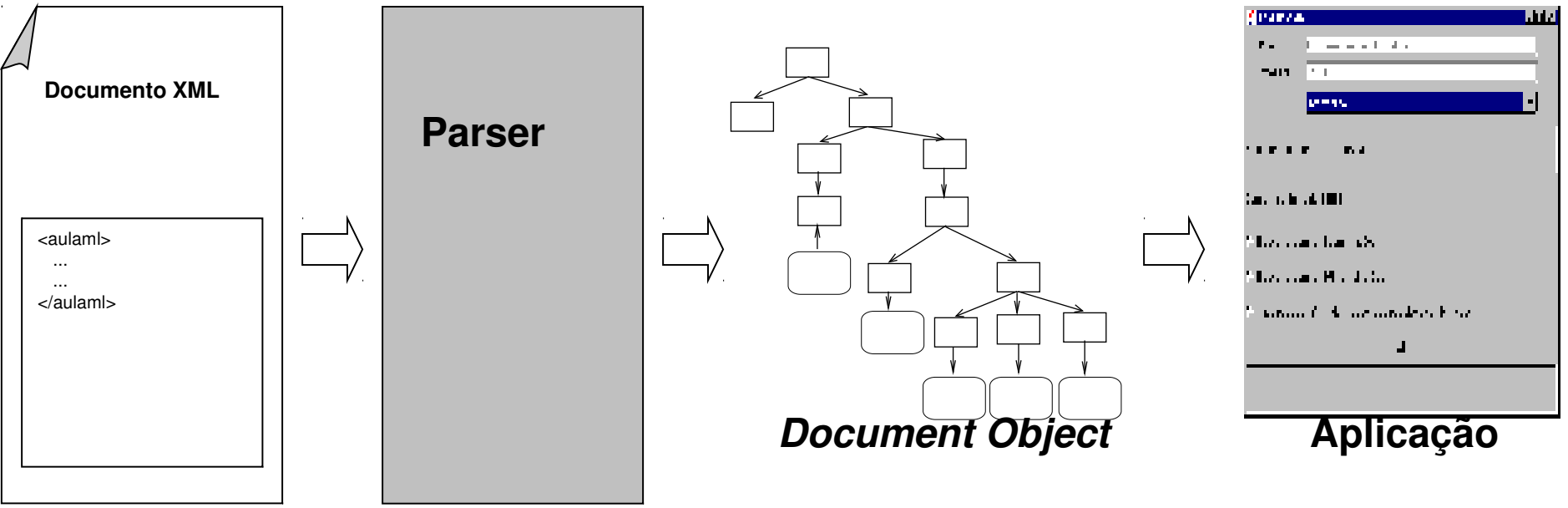
Curso: Informática

Questionário sobre BIT - Binary Digit

Com um conjunto de 8 BITS:

- É possivel compor 8 combinações
- É possivel compor 256 combinações
- Nada pode ser afirmado a respeito de possíveis combinações

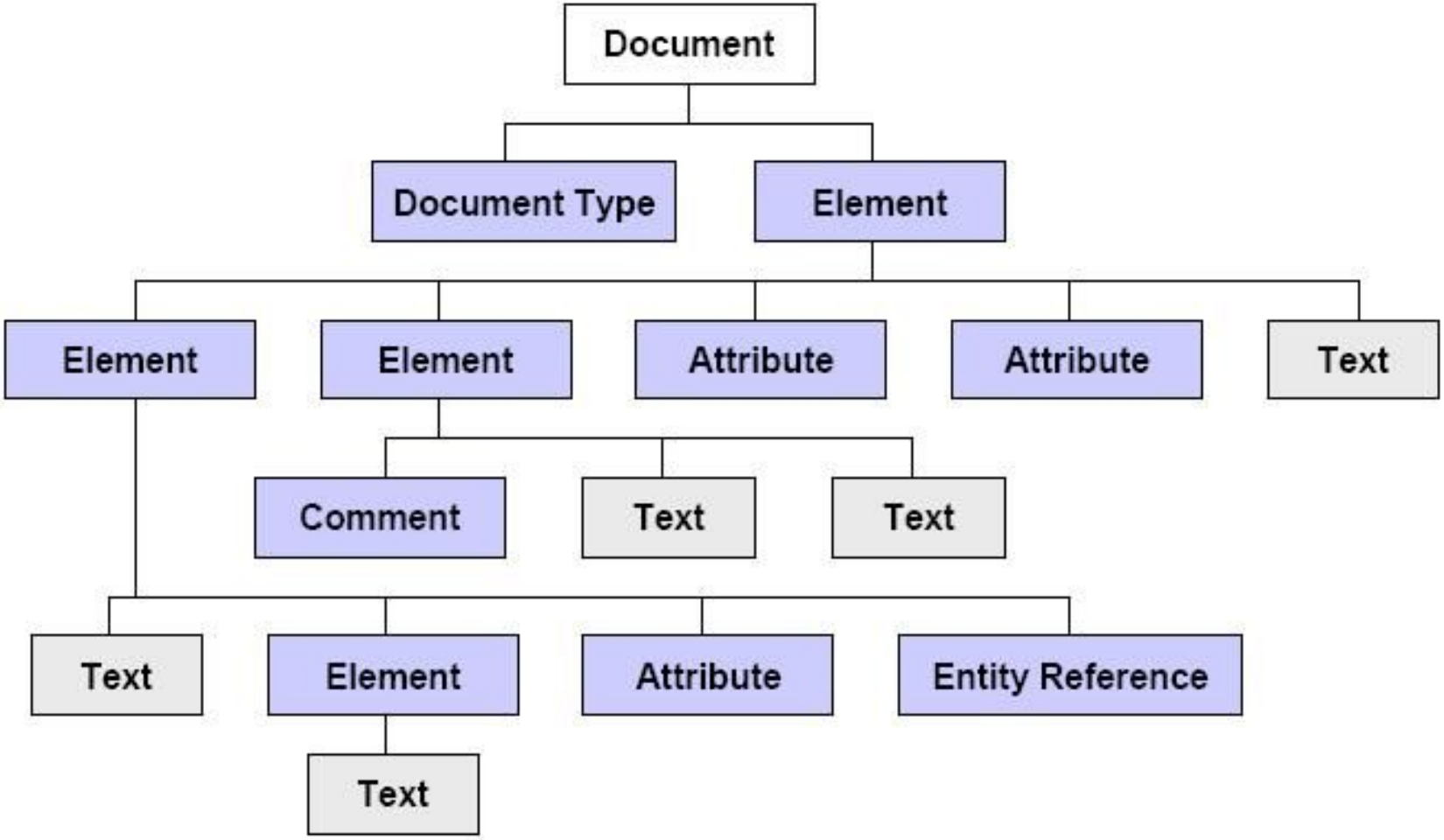
Ok


Documento XML
<aulaml>
...
...
</aulaml>
Parser
Document Object
Aplicação

Árvore XML DOM - Definição

- O primeiro nó é chamado de nó raiz.
- Todo nó, exceto o nó raiz possui um nó pai.
- Um nó pode ter qualquer número de filhos.
- Um nó folha é um nó sem filhos.
- Irmãos são os nós com o mesmo pai.
- Um nó árvore mostra um documento e as suas conexões, ou seja, o grau de parentesco entre os nós.

Árvore XML DOM

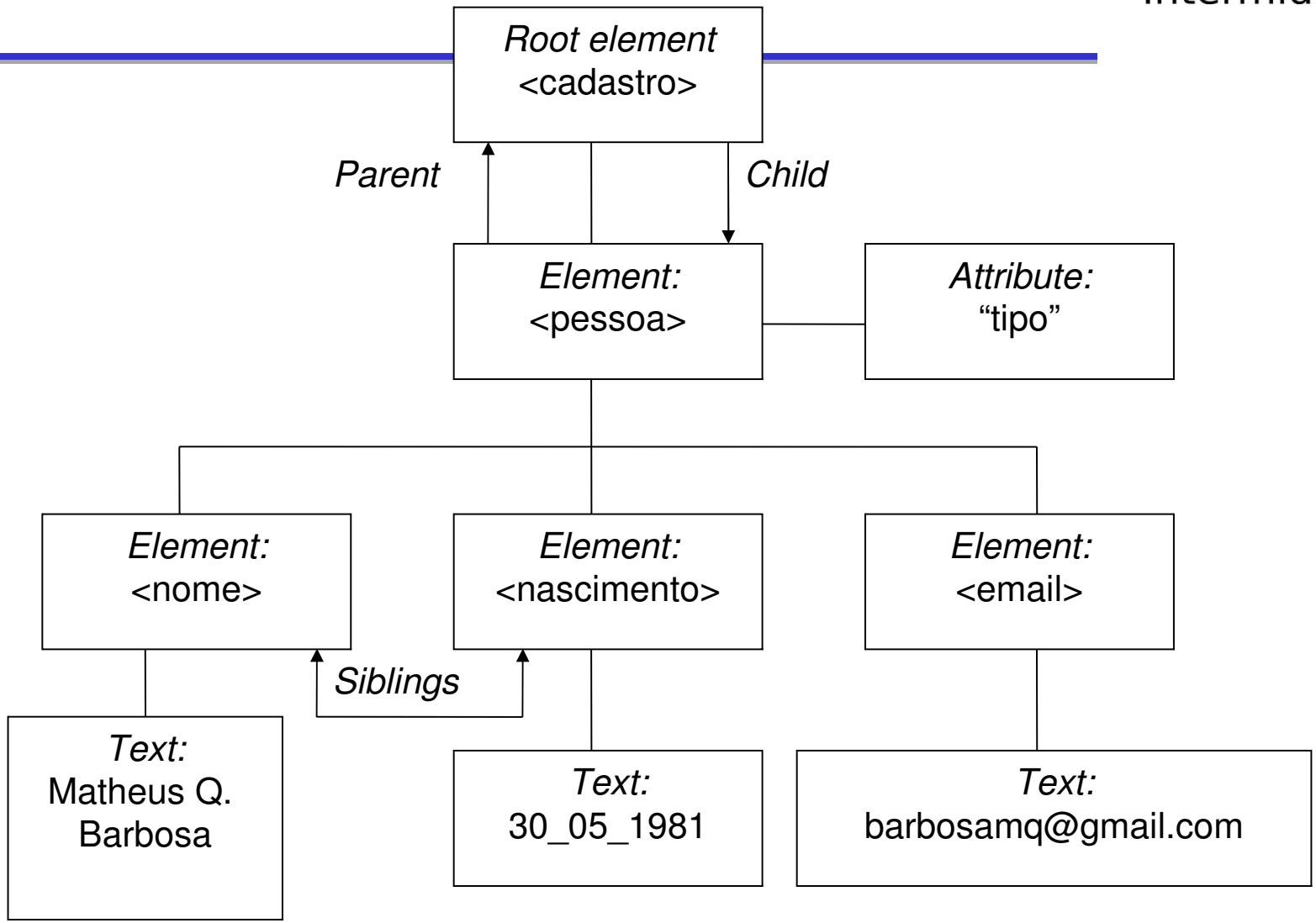


Árvore XML DOM

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cadastro>
<pessoa tipo="FISICA">
   <nome>Matheus Q. Barbosa</nome>
   <nascimento>30_05_1981</nascimento>
   <email>barbosamq@gmail.com</email>
</pessoa>
.
.
.
<pessoa>
...
</pessoa>
</cadastro>
```

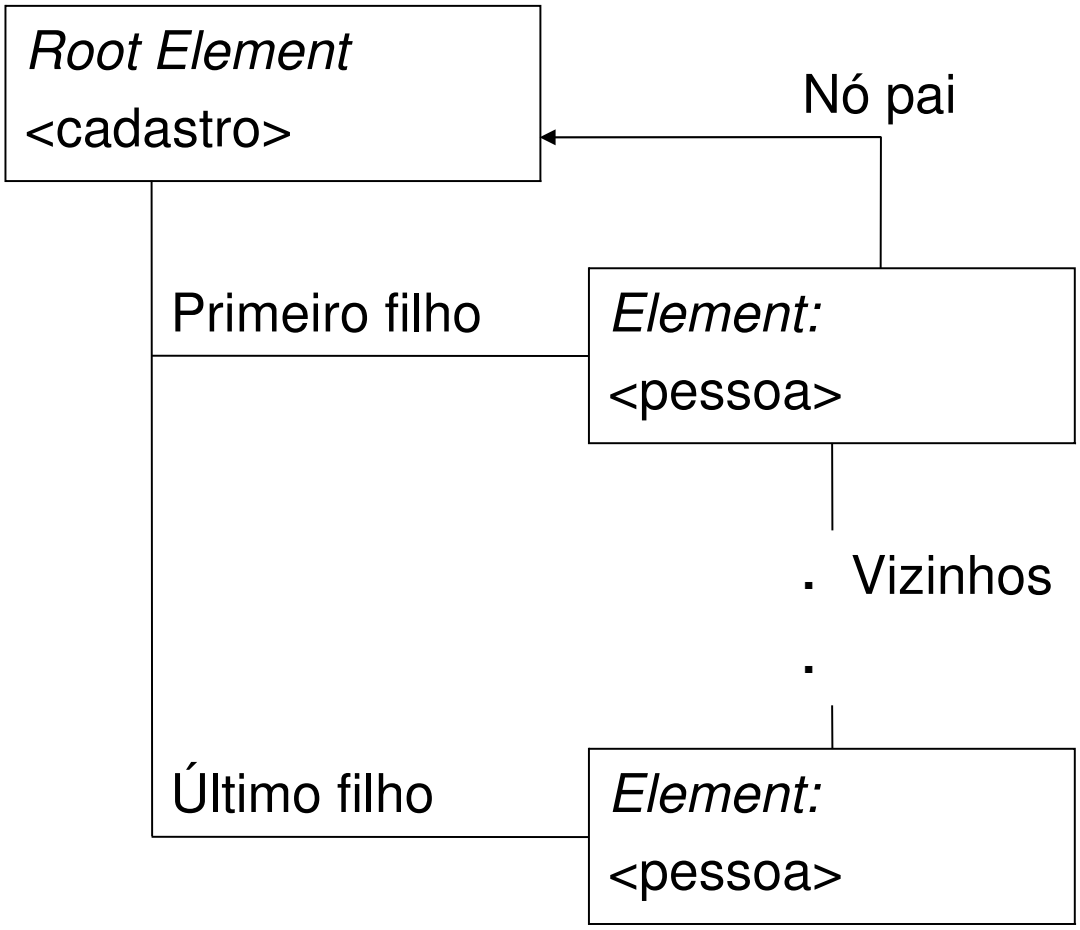
arquivo exemplo XML.

Exemplo de árvore XML DOM



Árvore XML DOM na memória, representando o arquivo XML exemplo

Árvore XML DOM: Grau de Parentesco e navegação



Árvore DOM carregada em memória referente ao arquivo exemplo xml

Tipos de nós das árvores XML DOM

Tipo do nó	Descrição	Filhos
Document	Representa todo o documento. É o nó raiz da árvore DOM.	*Element (max. one), ProcessingInstruction, Comment, DocumentType*
DocumentFragment	Representa um tipo de objeto documento, o qual pode manter um porção de documentos.	*Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference*
DocumentType	Provê uma interface para as entidades definidas por um documento.	Nulo

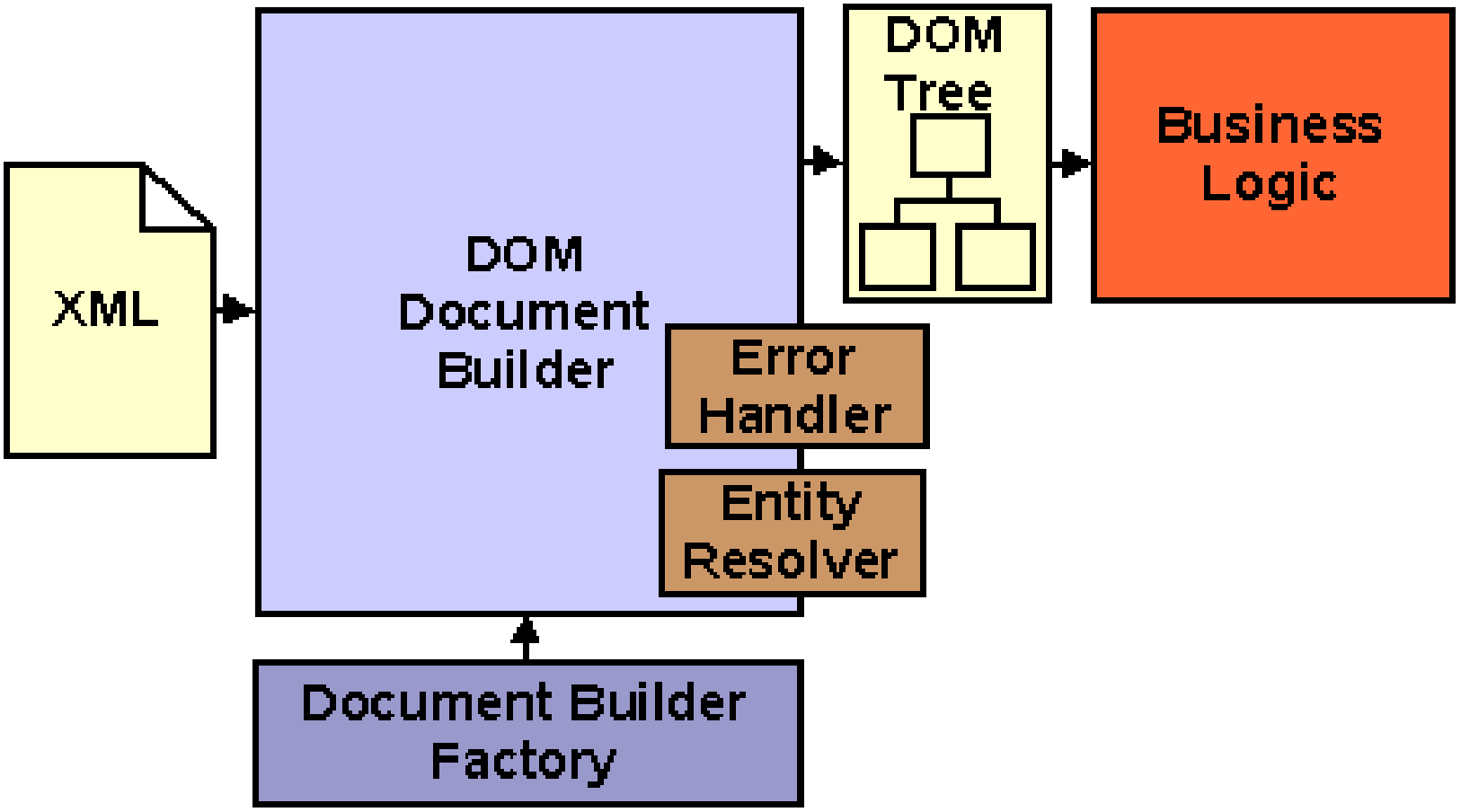
Tipos de nós das árvores XML DOM

ProcessingInstruction	Representa uma instrução de processamento.	Nulo
EntityReference	Representa uma entidade de referência.	*Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference*
Element	Representa um elemento.	*Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference*
Attr	Representa um atributo.	*Text, EntityReference*

Tipos de nós das árvores XML DOM

Text	Representa o conteúdo textual em um elemento ou atributo.	Nulo
CDATASection	Representa um seção CDATA num documento.	Nulo
Comment	Representa um comentário.	Nulo
Entity	Representa uma entidade.	*Element, ProcessingInstruction, Comme Text, CDATASection, EntityReference*
Notation	Representa uma notaçào declarada em DTD.	Nulo

Utilização de XML DOM



Representação da linguagem DOM, adaptada de (Violleau T., 2001).

API JAVA

- **Package `org.w3c.dom`**
- API que provê interfaces para DOM
  - Java API for XML Processing (JAXP)
- Descrição
  - O DOM nível 2 permite programas acessarem e atualizarem conteúdos e estrutura de documentos dinamicamente.

Carregando um arquivo em memória

Para carregar o arquivo em memória é necessário utilizar um documento e um *parser*.

As linhas de código abaixo fazem isso, retornando o nó raiz da árvore DOM. A navegação será iniciada a partir do nó raiz.

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse(new File("cadastro.xml"));
Node root = document.getDocumentElement();
```

Localizando um nó na árvore

Uma maneira de localizar um nó é através de uma lista de nós ou utilizando métodos de busca na árvore através do nome da *tag*, no caso a *tag message*.

Existem outras maneiras de realizar a busca.

```
Element mypessoaNode = ( Element ) root;
NodeList pessoaNodes =
    mypessoaNode.getElementsByTagName( "pessoa" );
Node pessoa = pessoaNodes.item( 0 );
```

Criando um nó na árvore

- Uma das maneiras de se criar um nó é fazer o clone de um nó já existente. Após fazer o clone de um nó já existente é possível modificar todas as propriedades do nó e o seu conteúdo.

```
Node pessoa2 = pessoa.cloneNode(true);
```

Modificando um nó na árvore

- Uma maneira de trocar a informação do objeto é através do método *replaceChild*, porém as versões mais novas da linguagem possuem novos métodos.
- Pode-se não só trocar as informações da *tag*, mas também trocar atributos.

```
Text nt = document.createTextNode("Pessoa Nova");
Text ot = ( Text ) pessoa2.getChildNodes().item( 0 );
pessoa2.replaceChild( nt, ot );
```

Adicionando um nó na árvore

- Uma maneira de adicionar objetos é através do método *appendChild()*, esse método permite adicionar um nó filho a qualquer outro nó da árvore.

```
pessoa.getParentNode().appendChild(pessoa2);
```

Removendo um nó na árvore

- O método que permite a retira de algum nó da árvore é o *removeChild()*, neste método é necessário informar o nó antigo que será removido.

```
pessoa.getParentNode().removeChild(oldText);
```

Exemplo 1: DOM em JAVA

- Um exemplo simples apenas para demonstrar como DOM funciona.
- Dado o seguinte arquivo introduction.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mymessage>
   <message>teste</message>
</mymessage>
```

Exemplo 1: DOM em JAVA

```
//imports Necessários
import java.io.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import javax.xml.transform.dom.*;
import org.xml.sax.*;
import org.w3c.dom.*;

//Classe ReplaceText
public class ReplaceText {
    private Document document;
   //construtor
    public ReplaceText( String fileName ){...}
   //método main
    public static void main( String args[] ){...}
}
```

Exemplo 1: Construtor

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
//carrega o documento na memória construindo a árvore
Document document = builder.parse(new File("introduction.xml"));
Node root = document.getDocumentElement(); //obtem o nó raiz
 if ( root.getNodeType() == Node.ELEMENT_NODE ) {
     Element mymessageNode = ( Element ) root;
     //localiza o elemento pelo nome da tag
     //no caso busca o elemento message
     NodeList messageNodes=ymessageNode.getElementsByTagName("message");
     //verifica se tem elemento
     if ( messageNodes.getLength() != 0 ) {
         Node message = messageNodes.item( 0 );
         // cria o nó de texto
         Text newText = document.createTextNode( "Novo texto" );
         // obtem o nó de texto antigo
         Text oldText =( Text ) message.getChildNodes().item( 0 );
         // troca o velho pelo novo
         message.replaceChild( newText, oldText );
     } ...
```

Exemplo1: Restante do arquivo

- Constituído do objetos que vão executar a gravação da árvore em memória no arquivo XML.
- Constituído de tratadores de erros: *catch's*
- Após a execução do programa:
  - `Output written to: introduction.xml`

Exemplo1: Resultado

- **Arquivo Introduction.xml antigo**

```
<?xml version="1.0" encoding="UTF-8"?>
<mymessage>
   <message>teste</message>
</mymessage>
```

- **Arquivo Introduction.xml novo**

```
<?xml version="1.0" encoding="UTF-8"?>
<mymessage>
   <message>Novo texto</message>
</mymessage>
```

Exemplo 2: DOM e JAVA

- Um exemplo de como mostrar todas as tags de um arquivo XML.
- Dado o introduction.xml visto anteriormente
- O seguinte programa:

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
public class DOMElements{
 static public void main(String[] arg)...
}
```

Exemplo2: Programa

```
try {
  BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
  System.out.print("Enter XML File name: ");
  String xmlFile = bf.readLine();
  File file = new File(xmlFile);
  //arquivo carregado na memória
  if(file.exists()){
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document doc = builder.parse(xmlFile);
    // Obtém a lista com todos os elementos do documento
    NodeList list = doc.getElementsByTagName("*");
    System.out.println("XML Elements: ");
    for (int i=0; i<list.getLength(); i++) {
   // Obtém o elemento
        Element element = (Element)list.item(i);
       // Imprime o nome do elemento.
   System.out.println(element.getNodeName());
    }
} else{...}}catch(Exception e){...}
```

Exemplo2: Resultado

- Dado o arquivo introduction.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mymessage>
   <message>Novo texto</message>
</mymessage>
```

- Execução do programa:

```
Enter XML File name: introduction.xml
```

- Resultado da execução

```
XML Elements:
mymessage
message
```

Exemplo 3

- Dado o arquivo LaboratóriosML.xml que contém atributos

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<laboratorios>
   ...
   <computadores>
     <quantidade> 51 </quantidade>
     <configuracao>
        <fabricante>HP</fabricante>
    <processador> Pentium 4 </processador>
    <clock medida="GHz“ fabr="atlon">2.8</clock>
    <memoriaRam medida="MBytes">512</memoriaRam>
     </configuracao>
   </computadores>
  </laboratorio>
...
</laboratorios>
```

Exemplo 3: Resultado

- Um programa que retorna os atributos dos elementos

```
... // retorna os atributos do elemento
NamedNodeMap startAttr = start.getAttributes();
// para cada atributo do elemento
for (int i = 0; i < startAttr.getLength(); i++) {
 // retorna atributo
    Node attr = startAttr.item(i);
  // imprime nome do atributo
    System.out.println("Attribute:"+attr.getNodeName()+"="+
 attr.getNodeValue());
} ...
```

- Resposta:

```
Attribute: medida = GHz
Attribute: fabr = atlon
```

Conclusão

- Permite manipulação de arquivos XML em memória
- Utiliza o conceito de árvore para a manipulação dos arquivos
- Possui vários métodos para manipular nós

- Independência de plataforma ou linguagem, várias API’s para diferentes linguagens de programação
- É um padrão W3C

Rastrear o documento XML e montar a partir dele um objeto segundo o DOM

```
Document documento =
  XmlDocument.createXmlDocument(origem,
  false);
```

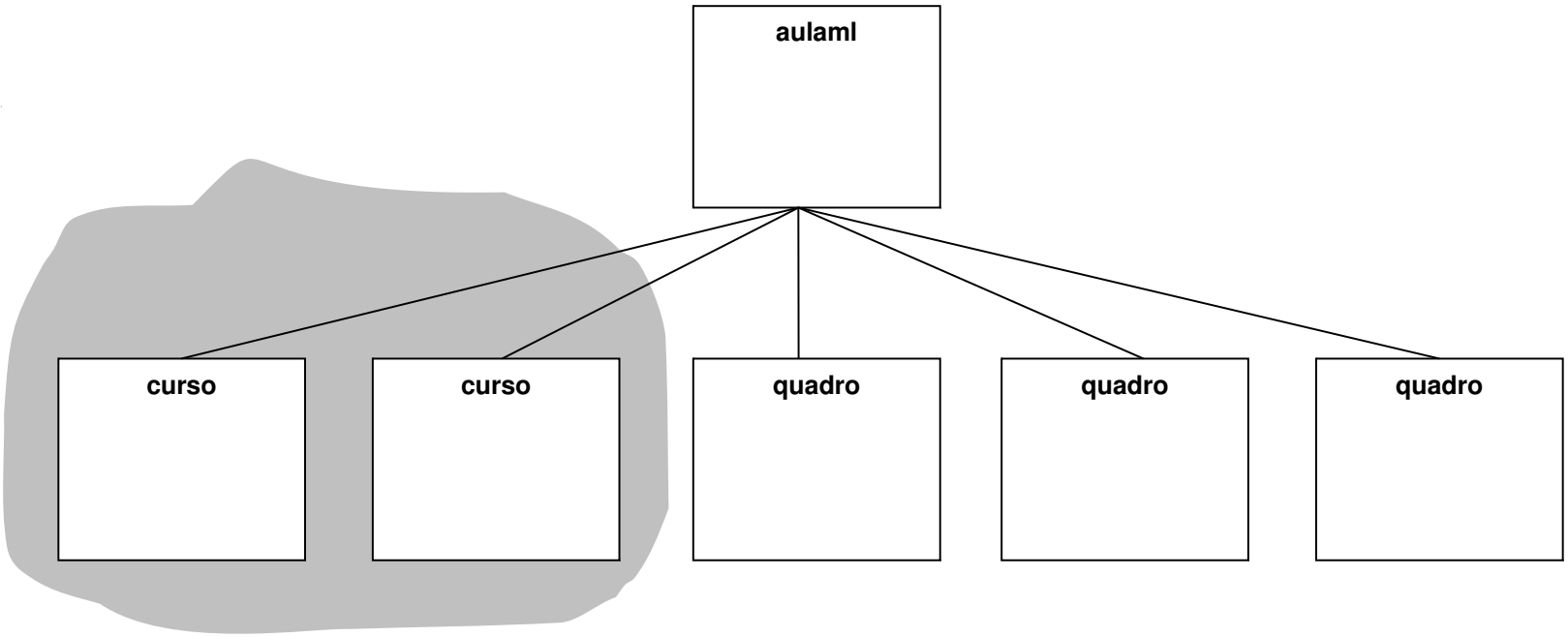
Recuperar do elemento de mais alto nível do documento

```
Element aula =
  documento.getDocumentElement();
```

Verificar o marcador da raiz – getNodeName()

```
if
  (aula.getNodeName().equals("aulaml"))
```

Navegar pelo Documento: obter cursos



getElementsByTagName(“curso”)

```
NodeList lista =
  aula.getElementsByTagName("curso");
Element curso = null;
if (lista != null) {
  int tamanho = lista.getLength();
  for (int i = 0; i < tamanho; i++) {
    curso = (Element)lista.item(i);

  listaCursos.addItem(curso.getAttribute("nome"
  ));
  }
}
```

Processando XML: DOM

DOM: Tree-Based API

- Um documento XML é representado como uma árvore
  - Os nós da árvore são os elementos, texto, etc.

Um processador XML gera a árvore e a envia para uma aplicação

- Um processador XML baseado em DOM cria a estrutura inteira de um documento XML na memória
- A aplicação utiliza a API DOM para manipular a árvore

Processando XML: DOM

Interface DOM

- A especificação DOM define uma interface em termos de um conjunto padrão de objetos
  - **Os nós da árvore são definidos como objetos**

- APIs: visa facilitar a interoperabilidade

- Do ponto de vista de programação orientada a objetos, a API DOM é um conjunto de interfaces que devem ser implementadas por uma determinada implementação DOM

Apresentar o Titulo da Aula e o Nome do Professor – getAttribute()

```
String titulo =
  aula.getAttribute("titulo");
String professor =
  aula.getAttribute("prof");
```

XML *DOM*

- **Funcionamento:**
  - Cria um objeto árvore cujos nós são obtidos a partir da estrutura e das informações contidas em um documento XML
    - Através desse objeto e das interfaces definidas pela API aplicações podem manipular o documento XML
  - Documentos XML são naturalmente hierárquicos
    - Estrutura de árvore funciona bem

XML *DOM*

Vantagens

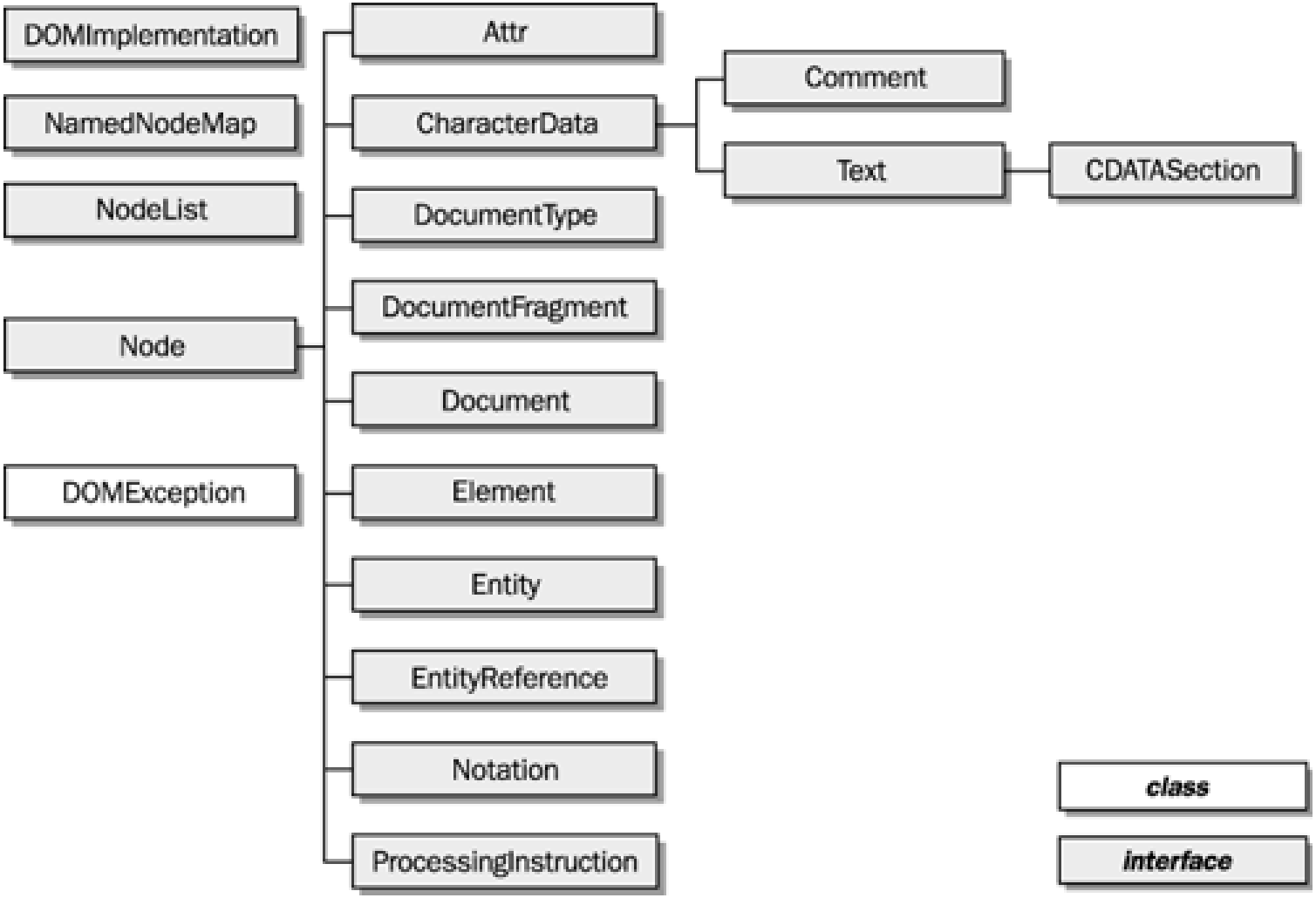
- Tem um modelo de objetos “pronto”
- É uma recomendação W3C

Desvantagens

- Documento deve estar totalmente disponível para poder ser utilizado
  - Objeto árvore só estará disponível quando o documento for totalmente processado
- Impõe um modelo de objetos baseado em árvore

XML DOM

Hierarquia de classe/interface



Nome da interface	Descrição
Node	Principal tipo de dados; um único nó na árvore do documento.
Document	O documento XML todo.
Element	Um elemento e todos os nós contidos nesse elemento.
Attr	Um atributo em um objeto *Element*.
ProcessingInstruction	Uma instrução de processamento.
CDATASection	Uma seção CDATA.
DocumentFragment	Um objeto documento usado para representar sub-árvores.
Entity	Um objeto documento usado para representar sub-árvores.
EntityReference	Uma referência de entidade .
DocumentType	Um DTD.
Notation	Uma notação declarada no DTD.
CharacterData	Uma interface pai de *Text*.
Comment	Um comentário.
Text	Texto.
DOMException	Uma exceção quando nenhum processamento adicional é possí
DOMImplementation	Métodos que não são dependentes de implementações DOM.
NodeList	Uma coleção ordenada de nós com acesso pelo índice.
NamedNodeMap	Uma coleção de nós com acesso pelo nome.

DOM versus SAX

DOM é indicado quando:

- A estrutura de um documento XML deve ser modificada
- O documento XML deve ser compartilhado na memória com outras aplicações
- O tamanho dos documentos XML não é grande
- As aplicações devem iniciar o processamento depois de concluir a validação

DOM versus SAX

SAX é indicado quando:

- A aplicação precisa de memória e desempenho
- A aplicação não precisa reconhecer a estrutura (complexa) de um documento XML

DOM versus SAX

- **Melhor usar DOM quando**
  - O modelo de objetos baseado em árvore é adequado às aplicações
  - Documentos que devem ser processados estão disponíveis localmente ou são pequenos
  - Tempo de processamento não é relevante
- **Melhor usar SAX quando**
  - O modelo de objetos baseado em árvore **NÃO** é adequado às aplicações
  - É necessário processar documentos grandes não disponíveis localmente

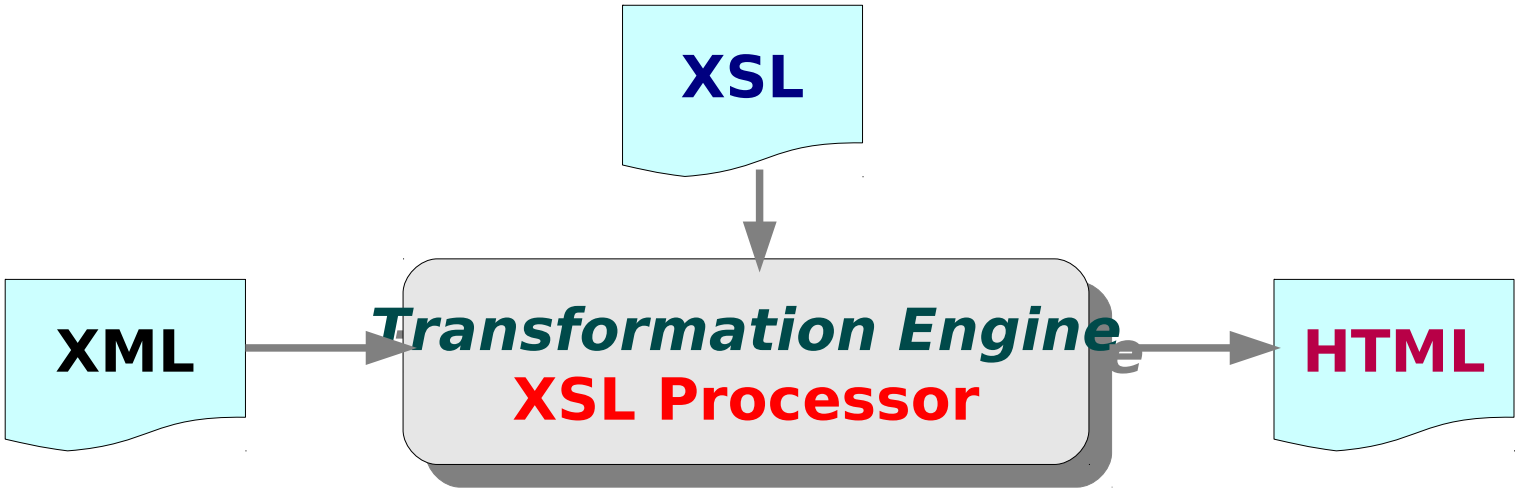
XML *Apresentação*

- **Documento XML não informa como apresentar seus dados**
  - Para tanto usa-se uma *stylesheet*, definida através da linguagem **XSL**
  - Stylesheet define quais elementos de um documento serão exibidos, e como será o aspecto de cada elemento apresentado
    - É possível definir várias “visões” de um mesmo documento

XML *Apresentação*

XSL

- Definida para transformar um documento XML em outro (também XML).
- Inclui facilidades para gerar documentos HTML a partir de documentos XML
  - Documentos HTML gerados são documentos XML



Intermidia

XSL

- Explora a característica básica dos documentos XML
  - Formato de árvore
- Cada nó de uma árvore (com exceção) das folhas forma uma sub-árvore
- Processador XSL navega pelos nós da árvore
  - Se um dado nó é uma folha, sua informação é processada e fim.
  - Se o nó não é uma folha, sua informação é processada e seus filhos são visitados e processados

XML *XSL*

- *Onde ocorre a transformação?*
  - 1) No cliente (web browser):
    - Documento XML e stylesheet associada são enviadas ao cliente, o qual possui um XSL Processor que processa o documento
      - Ex: Browser IE 5
  - 2) No servidor
    - Servidor aplica uma stylesheet em um documento XML e depois o envia ao cliente
      - IBM alphaWork's XML enabler servlet
  - 1 e 2 => Processamento síncrono

- *Onde ocorre a transformação?*
  - 3) Fora do contexto cliente-servidor (assincrona-mente)
    - Processador XSL é usado para converter documentos XML (provavelmente em docs HTML)
      - Documentos convertidos são colocados no servidor
      - Clientes e servidores manipulam apenas documentos convertidos
    - Ex: Processador XT (desenvolvido por James Clark)
      - Melhor implementação da especificação XSL

Referências

- Simple API for XML (SAX). http://sax.sourceforge.net/
- SAX - http://www.saxproject.org/
- Document Object Model (DOM). http://www.w3.org/DOM/
- Maruyama, H., Tamura, K., Uramoto, N., Murata, M., Clark, A., Nakamura, Y., Neyama, R., Kosaka, K., Hada, S. (2002) XML and Java: Developing Web Applications. Second Edition. Addison Wesley. ftp://ftp.kraft-s.ru/pub/doc/linux/How-to/tech/BOOKS/ftp.runnet.ru/BOOKS/ (arquivo Addison Wesley - XML and Java (2002).chm)

Referências

- (Armstrong, 2001) Eric Armstrong 2001, Working with XML The Java API for Xml Processing (JAXP) Tutorial, disponível em http://java.sun.com/webservices/jaxp/dist/1.1/docs/tutorial/index.html, acessado em 26 de Junho de 2007 as 12hs.
- **(**Deitel & Deitel, 2002) Harvey M. Deitel & Paul J. Deitel 2002, Java Como Programar, 4.º Edição, Editora: BOOKMAN, Ano de Edição: 2002, Capítulo 4, Document Object Model (DOM), disponível em: http://www.inf.ed.ac.uk/teaching/courses/ec/miniatures/dom-up.pdf, acessado em 26 de Junho de 2007 as 12hs.
- (Hall M., 2007) Marty Hall 2007, Creating and Parsing XML Files with DOM, disponível em http://courses.coreservlets.com/Course-Materials/pdf/java5/21-XML-and-DOM.pdf, acessado em 26 de Junho de 2007 as 12hs.
- (Harold, 2002) Elliotte Rusty Harold, 2002, Processing XML with Java, disponível em http://www.cafeconleche.org/books/xmljava/chapters/index.html, acessado em 26 de Junho de 2007 as 12hs.
- (Java, 2007) Sun Microsystems, Inc.2007, Java API for XML Code Samples, disponível em http://java.sun.com/developer/codesamples/xml.html#dom, acessado em 26 de Junho de 2007 as 12hs.
- (Violleau T., 2001) Thierry Violleau November 2001, Technology and XML Part 1 -- An Introduction to APIs for XML Processing, disponível em http://java.sun.com/developer/technicalArticles/xml/JavaTechandXML/#code14, acessado em 26 de Junho de 2007 as 12hs.
- (W3Schools, 2007) Refsnes Data 2007, XML DOM Tutorial, disponível em http://www.w3schools.com/dom/default.as, acessado em 26 de Junho de 2007 as 12hs.

Referências

Outros *links* disponíveis estão localizados em: http://notes.corewebprogramming.com/student/DOM.pdf, http://paginas.fe.up.pt/~jvv/Assuntos/DOM/dom.pdf, http://courses.coreservlets.com/Course-Materials/pdf/java5/21-XML-and-DOM.pdf,http://home.zhwin.ch/~rea/xml_ws05/DOM.pdf, http://www.dcc.ufrj.br/~braganholo/disciplinas/mab617/material/13-DOM.pdf, http://www.w3schools.com/dom/default.asp, http://www.guj.com.br/posts/list/489.java, http://www.guj.com.br/java.tutorial.artigo.22.1.guj, http://www.xmlbr.com.br/?conteudo=ver_artigo&idCat=6&artigo_id=29, http://docs.python.org/lib/module-xml.dom.html, http://www.devguru.com/Technologies/xmldom/QuickRef/xmldom_intro.html, http://www.rafaeldohms.com.br/2006/07/12/um-estudo-em-rss-parte-1-xml-dom/pt/, http://www.xmlfiles.com/dom/, http://courses.coreservlets.com, http://www.cafeconleche.org/books/xmljava/chapters/ch09.html#d0e13336, http://java.sun.com/developer/technicalArticles/xml/JavaTechandXML/#code14, http://www.inf.ed.ac.uk/teaching/courses/ec/miniatures/dom-up.pdf, http://www.idevelopment.info/data/Programming/java/PROGRAMMING_Java_Programming.shtml, http://java.sun.com/developer/codesamples/xml.html#dom, http://www.w3schools.com/dom/dom_nodes_get.asp, http://sax.sourceforge.net/, http://www.megginson.com/downloads/SAX/, http://www.inf.ed.ac.uk/teaching/courses/ec/ e http://www.cafeconleche.org/books/xmljava/chapters/ch11.html, todos acessados

Referências mais antigas

- Understanding DOM - https://www6.software.ibm.com/developerworks/education/x-udom/index.html
- Understanding SAX - https://www6.software.ibm.com/developerworks/education/x-usax/index.html
- Xerces2 Java Parser - http://xml.apache.org/xerces2-j/
- Crimson - http://xml.apache.org/crimson/
- XML Parsers: DOM and SAX Put to the Test - http://www.devx.com/xml/Article/16922/0/page/1
- Oracle XML Parser for Java - http://www.oracle.com/technology/tech/xml/xdk/doc/production/java/doc/java/parser/readme.html#Overview
- An Overview of MSXML 4.0 - http://www.xml.com/pub/a/2002/06/05/msxml4.html
- XML – DOM and SAX – http://www.progress.com/progress/exchange/post_2004/technical_sessions/c2720.ppt

Tarefa

- Elabore um programa para manipular o XML da sua grade-horária para:
  - listar todas as tags da sua grade-horária
  - Incluir uma nova disciplina em um dos horarios da sua grade-horaria
  - Substituir a disciplina 'scc265 – Sistemas interativos web' por “fim dos trabalhos“

  Entregue em Atividades do agora.tidia-ae até dia 21 maio!