

# 3

## XML DTD

Helder da Rocha  
(helder@argonavis.com.br)

# 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 – use um editor XML para descobrir
- 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*



## Por que validar?

- *Para a maior parte das aplicações, um XML bem formado é suficiente*
- *É possível, em documentos XML bem-formados, mas que não são válidos*
  - *Montar a árvore usando DOM*
  - *Extrair nós, acrescentar nós, alterar o conteúdo dos elementos usando SAX ou DOM*
  - *Transformar o documento em outro usando XSLT*
  - *Gerar um PDF ou um SVG com dados contidos no documento*
  - *Exibir o XML em um browser usando CSS*
- *Então porque ter o trabalho de criar um DTD ou um XML Schema?*



# Definir um esquema

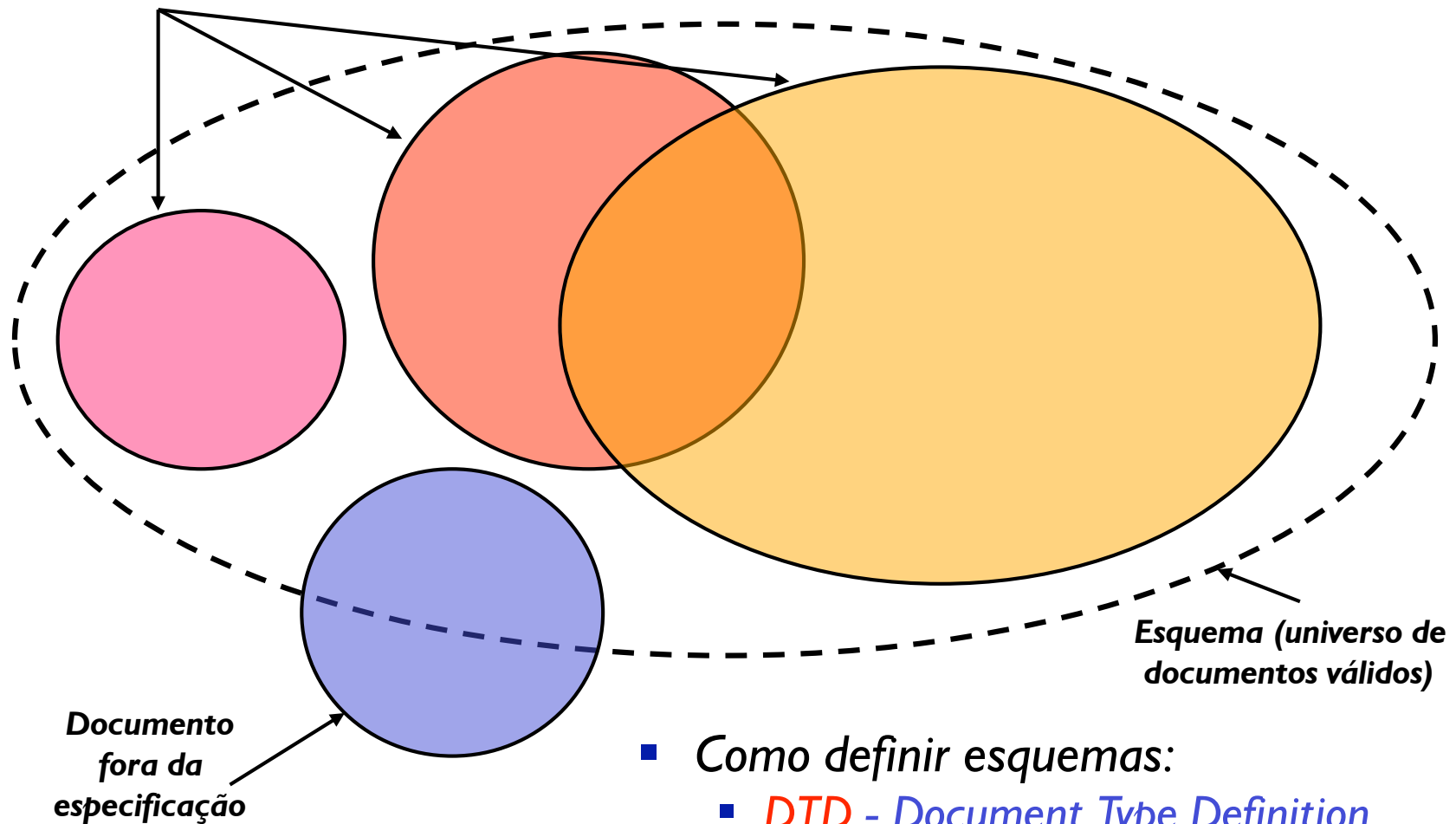
- *Documentos não válidos são "individualistas"*
  - *Um esquema representa um **conjunto** de documentos, que existem e que virão a existir*
  - *É possível fazer muitas coisas com **UM** documento não válido. É difícil automatizar os processos sem considerar uma **CLASSE** de documentos*
- *Um esquema é uma formalidade necessária*
  - *Se você tem uma grande coleção de documentos que foram construídos segundo determinadas regras, você já tem, **informalmente**, um esquema*
  - *Para validar documentos de acordo com suas convenções, é preciso ter um esquema*



# Esquema

Documentos que aderem à especificação (válidos)

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



- Como definir esquemas:
  - **DTD** - Document Type Definition
  - **W3C XML Schema**



# Documentos válidos

- Um relacionamento **pode** ser estabelecido formalmente entre um esquema e sua instância
  - Declaração de tipo de documento (para DTD)
    - `<!DOCTYPE bilhete SYSTEM "bilhete.dtd">`
  - Declaração de namespace e schema (para XML Schema), no elemento raiz
    - `<bilhete codigo="ZMIKT8" xmlns="http://abc.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://abc.com bilhete.xsd"> ...`
- Para validar
  - Use um parser validador (com suporte à linguagem de esquema desejada)



# O que define um esquema

- **Um vocabulário**
  - *Elementos, atributos*
- **Uma gramática**
  - *Relacionamentos (que elementos são permitidos onde, e de que forma, e com que conteúdo)*
- **Uma coleção de entidades**
  - *Variáveis que são substituídas por valores constantes durante o processamento (nem todo esquema tem suporte a declaração de entidades)*



# O que é um DTD?

- Um **DTD** (**D**ocument **T**ype **D**efinition) declara todos os elementos e atributos de um documento XML
  - Define quais elementos e atributos são válidos e em que contexto
  - A sintaxe é baseada em SGML. Para definir um elemento:  

```
<!ELEMENT nome-do-elemento (modelo de conteudo)>
```

- Exemplo: DTD para um documento simples

```
<!ELEMENT pessoa (nome, web?, telefone+)>
<!ELEMENT nome (prenome, inicial*, sobrenome)>
<!ELEMENT prenome (#PCDATA)>
<!ELEMENT inicial (#PCDATA)>
<!ELEMENT sobrenome (#PCDATA)>
<!ELEMENT web (email|website)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT website (#PCDATA)>
<!ELEMENT telefone (#PCDATA)>
```

*pessoa* tem *nome*, seguido de zero ou um *web* e um ou mais *telefone*

*nome* tem um *prenome*, seguido de zero ou mais *inicial* e um *sobrenome*

*web* pode conter ou um *email* ou um *website*

Elementos que só podem conter texto



# Documentos válidos segundo o DTD

- Os documentos abaixo *são válidos* segundo o DTD mostrado na página anterior

```
<peessoa>  
  <nome><prenome>Giordano</prenome>  
    <sobrenome>Bruno</sobrenome></nome>  
  <telefone>1199343232</telefone>  
</peessoa>
```

```
<peessoa>  
  <nome><prenome>Giordano</prenome>  
    <sobrenome>Bruno</sobrenome></nome>  
  <web><website>www.site.com</website></web>  
  <telefone>1199343232</telefone>  
</peessoa>
```

```
<peessoa>  
  <nome><prenome>Giordano</prenome>  
    <inicial>F</inicial><inicial>R</inicial>  
    <sobrenome>Bruno</sobrenome></nome>  
  <web><email>giordano@web.net</email></web>  
  <telefone>1199343232</telefone>  
  <telefone>1134999992</telefone>  
</peessoa>
```

# Documentos inválidos segundo o DTD

- Os documentos abaixo *não são válidos* de acordo com o DTD.
- Por que?

```
<peessoa>  
  <nome><prenome>Giordano</prenome>  
    <sobrenome>Bruno</sobrenome></nome>  
</peessoa>
```

```
<peessoa>  
  <nome><prenome>Giordano</prenome>  
    <sobrenome>Bruno</sobrenome></nome>  
  <web><website>www.site.com</website>  
    <email>giordano@web.net</email></web>  
  <telefone>1199343232</telefone>  
</peessoa>
```

```
<peessoa>  
  <nome><prenome>Giordano</prenome>  
    <sobrenome>Bruno</sobrenome></nome>  
  <telefone>1199343232</telefone>  
  <telefone>1134999992</telefone>  
  <web><email>giordano@web.net</email></web>  
</peessoa>
```

## DTD para validar uma instância

- Considere o seguinte documento XML

```
<bilhete codigo="ZMIKT8">  
  <voo transportador="JH"  
    numero="2349"  
    de="REC" para="CGH" />  
  
  <passageiro>  
    <sobrenome>Newton</sobrenome>  
    <prenome>Isaac</prenome>  
  </passageiro>  
</bilhete>
```



## Possíveis regras de validação

- Os elementos permitidos são
  - *bilhete, voo, passageiro, sobrenome, prenome*
- O elemento **bilhete** contém pelo menos um **voo** e exatamente um **passageiro**
- o elemento **passageiro** deve ter um elemento **sobrenome** e um elemento **nome**
- os atributos **de** e **para** de **voo** contém valores
  - *que podem ser escolhidos de uma lista em um DTD*
  - *que podem ser qualquer coisa, em outro DTD*



- *Este DTD valida o documento de forma pouco rigorosa*

```
<!ELEMENT bilhete ( voo, passageiro ) >
```

```
<!ATTLIST bilhete codigo NMTOKEN #REQUIRED >
```

```
<!ELEMENT passageiro ( sobrenome, prenome ) >
```

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

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

```
<!ELEMENT voo EMPTY >
```

```
<!ATTLIST voo de NMTOKEN #REQUIRED >
```

```
<!ATTLIST voo numero NMTOKEN #REQUIRED >
```

```
<!ATTLIST voo para NMTOKEN #REQUIRED >
```

```
<!ATTLIST voo transportador NMTOKEN #REQUIRED >
```



- *Este DTD restringe as informações que podem ser usadas nos atributos de **voo***

```
<!ELEMENT bilhete ( voo+, passageiro ) >  
<!ATTLIST bilhete codigo NMTOKEN #REQUIRED >
```

```
<!ELEMENT passageiro ( sobrenome, prenome ) >  
<!ELEMENT prenome ( #PCDATA ) >  
<!ELEMENT sobrenome ( #PCDATA ) >
```

```
<!ELEMENT voo EMPTY >  
<!ATTLIST voo de (REC | CGH | GRU | SDU) #REQUIRED >  
<!ATTLIST voo numero NMTOKEN #REQUIRED >  
<!ATTLIST voo para (REC | CGH | GRU | SDU) #REQUIRED >  
<!ATTLIST voo transportador (JH | RG) #REQUIRED >
```



# Elementos de um DTD XML 1.0

- **<!DOCTYPE>**
  - *Vincula o DTD a um documento*
  - *Usado no início do documento XML*
- **<!ELEMENT>**
  - *Define um elemento*
- **<!ATTLIST>**
  - *Define os atributos de um elemento*
- **<!ENTITY>**
  - *Define uma entidade ( ex: &nome; )*
- **<!NOTATION>**
  - *Define notação interna para uma URI*



# Elemento <!DOCTYPE>

- O elemento <!DOCTYPE> é um elemento do DTD que deve ser usado **dentro da página XML**
  - Identifica o elemento raiz
  - Associa o arquivo a um DTD através de URL ou **identificador público**

- Como vincular um documento XML a um DTD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>  
<!DOCTYPE cartao-simples SYSTEM "cartao.dtd">  
<cartao-simples>  
  <nome> (...)
```

nome do elemento raiz do documento

onde buscar validação: SYSTEM ou PUBLIC

URI ou identificador

- Alguns DTDs possuem um **identificador formal público (FPI)**
  - Neste caso, são declarados com a palavra PUBLIC e duas strings: o **identificador** seguido de uma URL onde pode ser encontrado

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"  
  "http://www.w3.org/TR/REC-html40/strict.dtd">
```



```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE pessoa [
  <!ELEMENT pessoa (nome, profissao*)>
  <!ELEMENT nome (prenome, sobrenome)>
  <!ELEMENT prenome (#PCDATA)>
  <!ELEMENT sobrenome (#PCDATA)>
  <!ELEMENT profissao (#PCDATA)>
]>
<pessoa>
  <nome>
    <prenome>Richard</prenome>
    </sobrenome>Feynman</sobrenome>
  </nome>
</pessoa>
```



## DTD incompleto

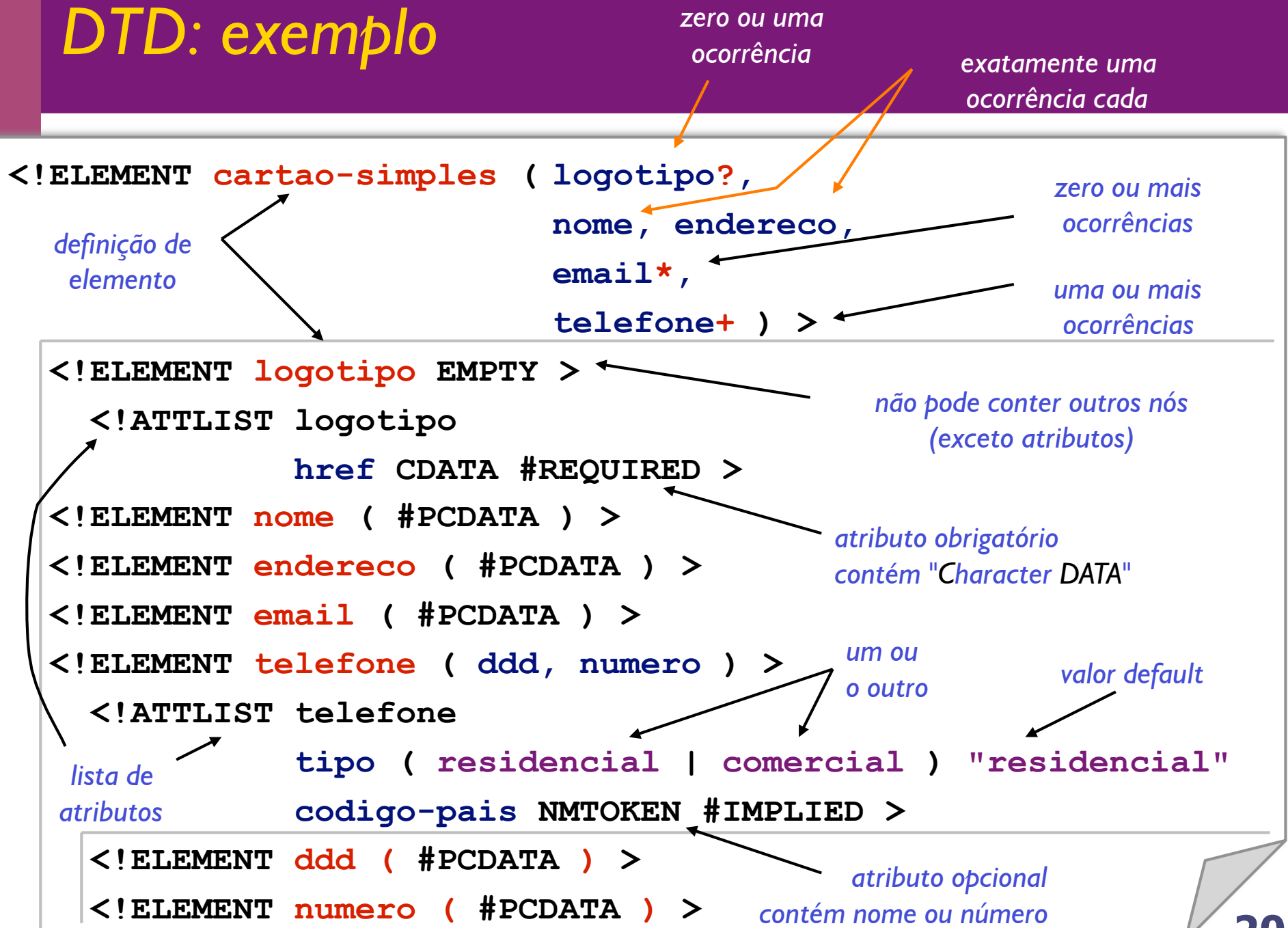
- O DTD foi importado, mas está incompleto. Parte dele é definido localmente

```
<!DOCTYPE pessoa SYSTEM "pessoa.dtd"[  
  <!ELEMENT nome (#PCDATA)>  
  <!ENTITY dtd "Document Type Definition">  
>
```

- Elementos, atributos e entidades definidos no documento têm prioridade sobre declarações importadas
  - Processador lê primeiro elementos locais, depois os que foram carregados do DTD externo
  - A primeira declaração é usada. Declarações adicionais para o mesmo elemento/atributo/entidade são desconsideradas – logo, declarações locais têm precedência



# DTD: exemplo



# Elemento `<!ELEMENT>`

- *Sintaxe*

`<!ELEMENT nome_do_elemento conteudo >`

- *O conteúdo pode ser*

- *(1) (#PCDATA),*
- *(2) (uma seqüência),*
- *(3) (uma lista de opções),*
- *(4) (conteúdo misto),*
- *(5) a palavra **EMPTY** ou*
- *(6) a palavra **ANY**.*



- ***Parsed Character Data***
  - *Elemento declarado com esse tipo **pode conter apenas texto simples***
  - *Pode conter entidades &valor;*
  - *Não pode conter elementos*



# Conteúdo Sequência

- O conteúdo deve ter uma lista de elementos separados por **vírgula**
  - Indica que os elementos filho que podem aparecer dentro do elemento declarado
  - Define uma ordenação dos elementos filho
- Podem ter sufixos indicando multiplicação
  - \* zero ou mais
  - + um ou mais
  - ? zero ou um
- Exemplos
  - `<!ELEMENT cartao (nome, telefone+, email*, website?) >`
  - `<!ELEMENT trem (locomotiva, vagoes*) >`



# Conteúdo lista de opções

- *Lista de elementos separada por |*
  - *Indica que **um** dentre os elementos filho listados pode ser usado como conteúdo do elemento declarado*
- *Exemplos*

```
<!ELEMENT trecho (ferroviário | aéreo |  
                  rodoviário | fluvial) >
```

```
<!ELEMENT círculo (centro, (raio | diametro)) >
```

```
<!ELEMENT ponto ((x, y) | (r, q)) >
```

```
<!ELEMENT nome (prenome | sobrenome |  
                (prenome, inicial*, sobrenome))>
```



## Conteúdo misto

- **(#PCDATA | elem1 | ... | elemn)\***
  - *Permite conteúdo de elementos, conteúdo de texto ou elementos misturados com texto*
- *Conteúdo misto não admite outra sintaxe*
  - *#PCDATA tem que ser o primeiro elemento*
  - *O \* no final é obrigatório*
  - *Não é possível controlar o número ou ordem dos filhos*

- *Exemplo: a seguinte declaração em DTD*

```
<!ELEMENT frase (#PCDATA | enfase)*>
```

*permite o seguinte conteúdo*

```
<frase>A frase que <enfase>ele</enfase> leu não foi a  
que <enfase>ela</enfase> ouviu.</frase>
```



## Conteúdo Vazio e Any

- *Elementos que não podem conter nada devem ser declarados como **EMPTY***
  - `<!ELEMENT nome EMPTY>`
- *Elementos **EMPTY** podem conter atributos*
  - `<!ATTLIST nome prenome CDATA #REQUIRED>`
- *A declaração acima permite elementos como*
  - `<nome prenome="José" />`
- **ANY**: *use para elementos que podem conter qualquer coisa*
  - `<!ELEMENT todos ANY>`



# Elemento <!ATTLIST>

## ■ Sintaxe

```
<!ATTLIST elemento  
  atributo1 tipo valor_default>
```

```
<!ATTLIST elemento  
  atributo2 tipo valor_default>
```

...

ou

```
<!ATTLIST elemento  
  atributo1 tipo valor_default  
  atributo2 tipo valor_default  
  atributo3 tipo valor_default  
  atributo4 tipo valor_default
```

... >



# Exemplos

```
<!ATTLIST voo de NMTOKEN #IMPLIED >
```

```
<!ATTLIST voo numero CDATA #FIXED "12/32-1">
```

```
<!ATTLIST voo para (REC | CGH | GRU | SDU) #REQUIRED >
```

```
<!ATTLIST voo transportador (RG | JH) "RG" >
```

*ou*

```
<!ATTLIST voo de NMTOKEN #IMPLIED
```

```
numero CDATA #FIXED "12/32-1"
```

```
para (REC | CGH | GRU | SDU) #REQUIRED
```

```
transportador (RG | JH) "RG" >
```



**CDATA**

**NMTOKEN**

**NMTOKENS**

*Lista de opções*

**ID**

**IDREF**

**IDREFS**

**ENTITY**

**ENTITIES**

**NOTATION**



## Tipo CDATA e NMTOKEN

- **CDATA** (*character data*) representa qualquer texto arbitrário
  - Pode conter espaços, pontuação, etc.
- **NMTOKEN** (*name token*). É semelhante a um nome de elemento ou atributo
  - Caracteres alfanuméricos apenas
  - Não pode conter espaços nem pontuação
- **NMTOKENS** representa um ou mais NMTOKEN separados por espaços.  
<elemento atributo="um dois três 56 21">



## Tipo lista de opções

- *Uma lista de NMTOKENS dentre os quais pode-se escolher o valor do atributo.*
- *As escolhas são separadas por |:*  

```
<!ATTLIST voo para (REC|CGH|SDU) #REQUIRED>  
<!ATTLIST curso periodo (dia|noite) "noite">
```
- *Elementos não podem conter espaços ou outros caracteres não-alfanuméricos*
  - *O tipo de cada opção é NMTOKEN!*



- Atributos do tipo **ID** tem que conter um nome (e não NMTOKEN) que seja **unívoco no documento**.
  - Nome tem mesmas regras que identificador XML (caracteres alfanuméricos, não começa com número, etc.)
- Não se pode usar um **número** como conteúdo de um atributo declarado como ID.
  - A solução é colocar um prefixo antes do número que seja ou uma letra ou um sublinhado.
- Elementos só podem ter **um tipo ID**
  - Não pode haver outro elemento na mesma página com o mesmo ID
- Exemplos
  - <!ATTLIST voo **codigo ID** #REQUIRED>
  - <!ATTLIST piloto **numero ID** #REQUIRED>



- **IDREF** é referência para um **ID**. Exemplo:

```
<!ATTLIST piloto alocado IDREF #REQUIRED>
```

- *Aplicação*

```
<aeroporto>  
  <voo codigo="RG123"> ... </voo>  
  <voo codigo="RG456"> ... </voo>  
  <piloto numero="S1" alocado="RG123">  
    ... </piloto>  
  <piloto numero="S2" alocado="RG456">  
    ... </piloto>  
</aeroporto>
```



- Lista de elementos **IDREF**. Exemplo:

```
<!ATTLIST piloto alocado IDREFS #REQUIRED>
```

- Aplicação (codigo e numero são IDs)

```
<aeroporto>  
  <voo codigo="RG123"> ... </voo>  
  <voo codigo="RG456"> ... </voo>  
  <piloto numero="S2"  
    alocado="RG456 RG123">  
    ... </piloto>  
</aeroporto>
```



## Valores default

- **#REQUIRED**: força o autor do documento a definir um valor explícito ao atributo.
- **#IMPLIED**: o atributo é opcional.
- **#FIXED**: o atributo tem um valor fixo, constante
  - Valor não pode ser mudado
  - Autor não precisa definir atributo e, se o fizer, não pode mudar o valor.
  - Exemplo:  
`<!ATTLIST valor moeda CDATA #FIXED "US$">`
- **Valor inicial**, entre aspas
  - Autor não precisa definir atributo, mas pode mudá-lo  
`<!ATTLIST voo companhia (RG | JH) "RG" >`  
`<!ATTLIST endereco pais CDATA "Brasil">`



# Elemento `<!NOTATION>` e tipo `NOTATION`

- Associa URI a um nome

- Usado frequentemente para associar valores `CDATA` a `NMTOKEN` (mesmo onde não há URIs)

- Exemplos

```
<!NOTATION amazon SYSTEM "http://www.amazon.com">
```

```
<!NOTATION barnes SYSTEM "http://www.bn.com">
```

- Tipo `NOTATION` de `<!ATTLIST>` é útil em situações onde não se pode usar `CDATA`

```
<!ATTLIST book
```

```
    store NOTATION (amazon | barnes) #REQUIRED>
```

- Assim pode-se limitar melhor valores dos atributos

```
<book store="amazon" />
```



# Elemento `<!ENTITY>`

- Há vários tipos de `<!ENTITY>`
  - Definem **entidades**: variáveis usadas nos DTDs e documentos XML
- Entidades de parâmetro (`%nome;`): usadas **apenas** no DTD
  - internas – contém texto declarado localmente no DTD
  - externas – contém conteúdo de arquivos externos (sub DTDs)
- Entidades gerais (`&nome;`): usadas **apenas** no XML
  - internas (são sempre processadas)
    - **caractere** – código Unicode: `&#333;`; `&#x3AB4;`
    - **globais** – cinco entidades: `&lt;`; `&gt;`; `&amp;`; `&quot;`; e `&apos;`;
    - **definidas pelo usuário** – contém texto declarado no DTD
  - externas (carregam dados de arquivos externos)
    - **processadas** – incluem texto de arquivos externos no XML; o processador resolve as entidades e blocos CDATA se houver
    - **não-processadas** – incluem formatos binários (ex: imagens); o processador ignora o conteúdo e não processa (usado em atributos apenas)



# Entidades gerais internas

`<!ENTITY nome "valor">`

- *Exemplos:*

```
<!ENTITY empresa "ACME Indústria &
Comércio de Coisas S.A.">
```

```
<!ENTITY copyright "<table>
<tr>
  <td>Copyright &#x00A9 2000</td>
</tr>
</table>">
```

- *Uso*

```
<texto> Visite a &empresa; ainda hoje!.
</texto>
<div> &copyright; </div>
```



# Entidades gerais externas

- Carregam texto de arquivos externos ao XML
  - <!ENTITY nome SYSTEM "uri">
  - <!ENTITY nome PUBLIC "fpi" "uri">
- Exemplo
  - <!ENTITY menu\_sup SYSTEM "/sec/menu.xml">
- Uso
  - <elemento>
  - &menu\_sup;
  - <elemento>
- Conteúdo de *menu.xml*:
  - <menu>
  - Texto
  - </menu>
- Resultado
  - <elemento>
  - <menu>
  - Texto
  - </menu>
  - </elemento>



# Entidades gerais externas não processadas

- Usadas para carregar dados que não podem ser processados (que não são texto) através de atributos

`<!ENTITY nome SYSTEM "uri" NDATA notação>`

- Depende de uma declaração NOTATION

- Neste caso típico usada para informar tipo de dados

- Exemplo de uso no DTD

`<!NOTATION gif SYSTEM "image/gif">`

`<!ENTITY logo SYSTEM "logo.gif" NDATA gif>`

- Atributos podem declarar receber tipo ENTITY

`<!ATTLIST imagem fonte ENTITY #REQUIRED>`

- Uso no XML:

`<imagem fonte="logo">`

Esta entidade geral **logo** é usada apenas em atributos

A sintaxe é diferente: seria **&logo;** se fosse uma entidade processada



# Entidades de parâmetro internas

`<!ENTITY % nome "valor">`

- *Variáveis declaradas e usadas apenas dentro do DTD*
- *Em vez de repetir*

```
<!ELEMENT voo de (REC | CGH | GRU | GIG | SDU)>
```

```
<!ELEMENT voo para (REC | CGH | GRU | GIG | SDU)>
```

- *Declare as entidades*

```
<!ENTITY % aerosp "CGH | GRU">
```

```
<!ENTITY % aerorio "GIG | SDU">
```

- *E use no DTD*

```
<!ENTITY % aeroportos "REC | %aerorio; | %aerosp;">
```

```
<!ATTLIST voo de (%aeroportos;) #REQUIRED >
```

```
<!ATTLIST voo para (%aeroportos;) #REQUIRED >
```



# Entidades de parâmetro externas

- Carregam fragmentos de DTDs externos

```
<!ENTITY % nome SYSTEM "valor">
```

- Exemplo

```
<!ENTITY % tabela SYSTEM "voos.dtd">
```

- É preciso chamar a entidade dentro do DTD.

- Uso

- Resultado

```
<!ENTITY ...>
```

```
<!ENTITY ...>
```

```
%tabela;
```



```
<!ENTITY % sp "CGH|GRU">
```

```
<!ELEMENT ...>
```

```
<!ENTITY % rio "GIG|SDU">
```

```
...
```

```
<!ELEMENT ...>
```

- Conteúdo de *voos.dtd*:

```
<!ENTITY % sp "CGH|GRU">
```

```
<!ENTITY % rio "GIG|SDU">
```



- *À primeira vista parecem inúteis*
- *Servem para construir DTDs configuráveis e modulares*
  - *Módulos permitem segmentação e reuso*
  - *São usados por aplicações como XHTML 1.1, SVG 1.1*
- **Bloco IGNORE:** *ignora o conteúdo*

```
<![IGNORE [  
    <!ELEMENT detalhes (#PCDATA)>  
]]>
```

- **Bloco INCLUDE:** *declara que o texto deve ser interpretado*

```
<![INCLUDE [  
    <!ELEMENT detalhes (#PCDATA)>  
]]>
```



# Utilidade de blocos condicionais

- Se texto "IGNORE" OU "INCLUDE" for atribuído a entidades de parâmetro, pode-se "ligar" ou "desligar" declarações

- Exemplo: no DTD importado

```
<!ENTITY % define.mod.um "IGNORE">
<!ENTITY % define.mod.dois "IGNORE">
<![%define.mod.um [
    <!ELEMENT coisas (#PCDATA)>
]]>
<![%define.mod.dois [
    <!ELEMENT maiscoisas (#PCDATA)>
]]>
```

- Blocos acima são "desligados" por default
- Em um DTD local, pode-se "ligar" o que se deseja usar

```
<!ENTITY % define.mod.um "INCLUDE">
```

- Agora elemento <coisas> faz parte do DTD!



# Validação: além do DTD

- **DTD**
  - **vantagem:** é simples
  - **desvantagens:** (1) não usa sintaxe XML; (2) é limitado.
- **XML Schema**
  - **vantagem:** (1) é XML; (2) permite especificação muito mais precisa e detalhada.
  - **desvantagem:** é muito mais complicado que DTD
- **Schema ainda não resolve todos os problemas**
  - **Outras alternativas:** Trax, RELAX, Schematron
  - XML Schema é extensível e permite usar essas linguagens para completar a validação
  - Em certos casos ainda pode ser preciso realizar validação adicional usando alguma linguagem (Python, Java)



# XML Schema

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="bilhete">
    <xs:complexType>
      <xs:sequence>
        (...)
      </xs:sequence>
      <xs:element name="voo">
        <xs:complexType>
          <xs:attribute name="de" use="required">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="REC" />
                <xs:enumeration value="CGH" />
                <xs:enumeration value="GRU" />
                <xs:enumeration value="SDU" />
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute> (...)
        </xs:complexType>
      </xs:element>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

- XML Schema é uma alternativa ao DTD
  - Oferece mais recursos
  - Usa sintaxe XML
- Exemplo (trecho do XML Schema para o bilhete de vôo)

