Módulo II
Introdução a XMLSchema

Prof. Ismael H F Santos

Ementa

- Modulo II – XML Schema
  - Revisão de XML - Extensible Markup Language
  - XML Validação
    - DTD
    - XMLSchema
    - DTD x XMLSchema
  - XML Processing - XSLT

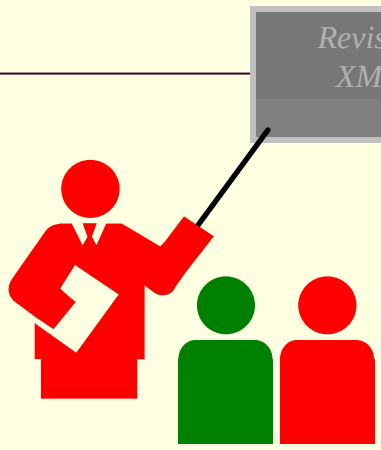
Bibliografia

- *Linguagem de Programação JAVA*
  - *Ismael H. F. Santos, Apostila UniverCidade, 2002*

Livros

- **Core Java 2**, Cay S. Horstmann, Gary Cornell
  - Volume 1 (Fundamentos)
  - Volume 2 (Características Avançadas)
- **Java: Como Programar**, Deitel & Deitel
- **Thinking in Patterns with JAVA**, Bruce Eckel
  - **Gratuito**. http://www.mindview.net/Books/TIJ/

SOA



What is XML

- XML stands for extensible markup language
- It is a hierarchical data description language
- It is a sub set of SGML a general document markup language designed for the American millitary.
- It is defined by w3c.

XML – Extensible Markup Language

- XML é um documento de Texto
- Dois tipos de elementos
  - Marcação – Guarda a estrutura do documento
  - Dados – Informação propriamente dita
- Uma maneira de representar informação
  - não é uma linguagem específica
  - não define vocabulário de comandos
  - não define uma gramática, apenas regras mínimas
- Exemplo:

usuario_33.xml

```
<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>
```

elemento

atributo

"nó" de texto

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):*

```
<nome> Северино Северович </nome>
```

Elemento raiz

  - *Menor documento contendo elemento vazio*

```
<nome></nome>   =   <nome/>
```

  - *Menor documento contendo elemento e conteúdo texto*

```
<nome> Северино Северович </nome>
```

Etiqueta inicial — Conteúdo do Elemento — Etiqueta final

How does XML differ from HTML?

- HTML is a presentation markup language – provides no information about content.
- There is only one standard definition of all of the tags used in HTML.
- XML can define both presentation style and give information about content.
- XML relies on custom documents defining the meaning of tags.

XML x 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>
```

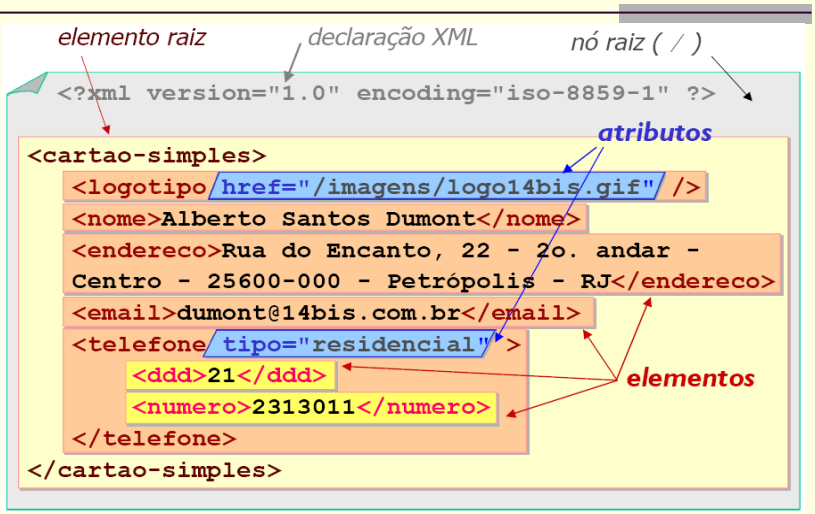
Por que usar XML para compartilhar dados?

- ***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***

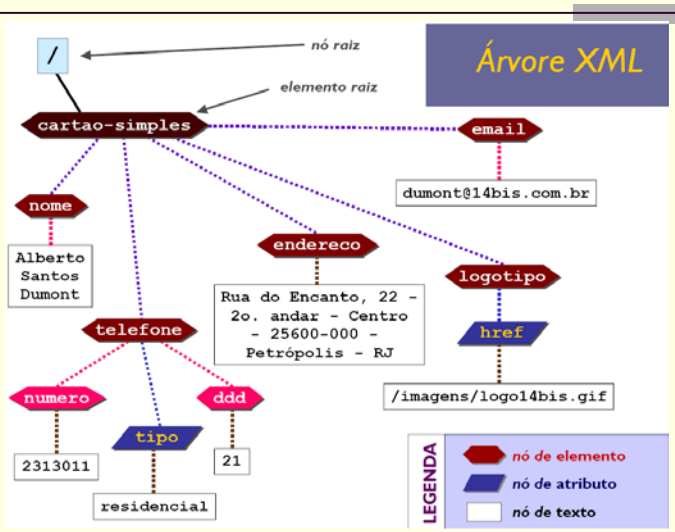
Componentes de um documento XML

- ***Árvore XML***
  - ***nós,***
  - ***raiz,***
  - ***galhos e***
  - ***folhas***
- ***Prólogo***
- ***Comentários***
- ***Instruções de processamento***
- ***Elementos***
- ***Atributos***
- ***Nós de texto***
- ***Entidades***

Partes de um documento XML



Árvore XML



Estrutura XML

- *Um documento XML pode ser representado como uma árvore. A estrutura é formada por vários nós.*

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!-- Isto é um comentário -->
<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>
```

informações usadas pelo processador XML

um "nó" pode ser ... um elemento, um atributo, um bloco de texto, um comentário, uma instrução, uma declaração, uma entidade, ...

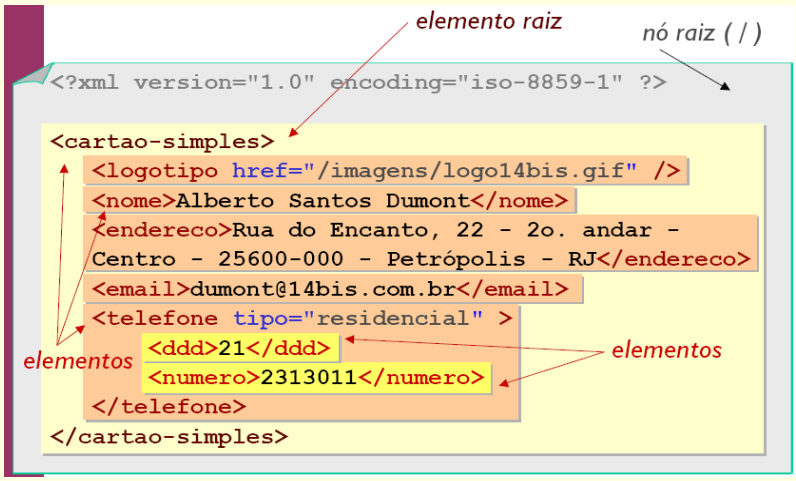
Prólogo XML

Prólogo

Declaração XML
Comentário
Instrução de processamento
Declaração de tipo de documento

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!-- Isto é um comentário -->
<?comando tipo="simples" parametro ?>
<!DOCTYPE cartao-simples SYSTEM "cartoes.dtd">
<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>
```

Nó raiz e elementos



Atributos

- *Só podem conter um descendente (só texto)*

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

<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>
```

atributos

Nós de Texto

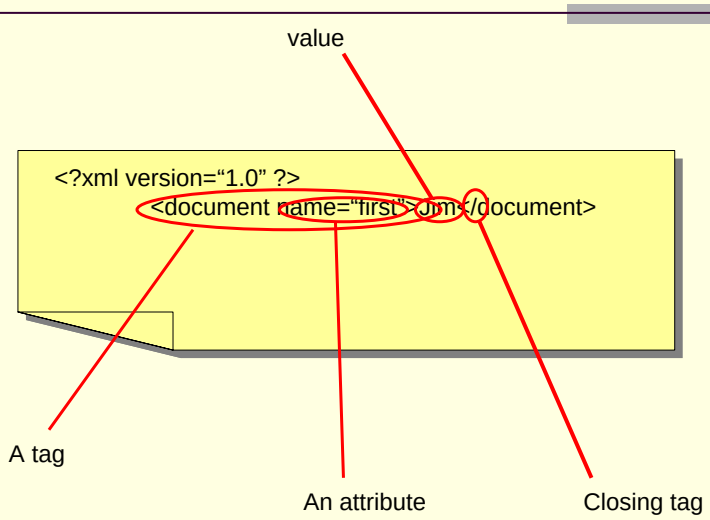
- Não podem ter descendentes (são as folhas da árvore)

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

<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>
```

nós de texto

A minimal XML document



Entidades

- São constantes associadas a um valor de texto
  - Podem aparecer em qualquer lugar do documento
- Substituídas **durante** o processamento do documento
- Sintaxe:
  - &ENTIDADE;
- Exemplo:
  - &data_de_hoje;
- Entidades pré-definidas:
  - **<** que corresponde a <
  - **>** que corresponde a >
  - **&** que corresponde a &
  - **"** que corresponde a "
  - **'** que corresponde a '

Entidades de caracteres

- Substituídas durante o processamento do documento
- Sintaxe:
  - **&#**CÓDIGO_16b_decimal**;**
  - **&#x**CÓDIGO_16b_hexadecimal**;**
- Exemplo:
  -   ou
  - Um espaço em Unicode
  - Veja em www.unicode.org/charts/

Elementos e Atributos

- *Regras básicas*
  - *Etiqueta inicial e final têm que ter o mesmo nome (considerando diferença de maiúscula e minúscula)*
  - *Não pode haver espaço depois do < nas etiquetas iniciais nem depois do </ nas finais*
  - *Atributos têm sempre a forma nome="valor" ou nome = 'valor':*
    - *aspas podem ser usadas entre apóstrofes e apóstrofes podem ser usados entre aspas*
    - *aspas e apóstrofes não podem ser neutralizados mas sempre podem ser representados pelas entidades ' e "*
  - *Não pode haver atributos na etiqueta final*

Elementos e Atributos (2)

- *Elementos mal formados*

```
<Profissão>Arquiteto</profissão>
<TR><TD>item um</td></tr>
<ÄÅÍÃÈÉ>139.00</äåíãèé>
```

- *Há várias maneiras de representar a mesma informação em XML*

```
<data>23/02/1998</data>
<data dia="23" mes="02" ano="1998" />
<data>
     <dia>23</dia>
     <mes>02</mes>
     <ano>1998</ano>
</data>
```

Quando usar elementos/atributos

- ***Questão de design***
  - ***Elementos geralmente referem-se a** coisas **que têm atributos***
  - ***Atributos geralmente são** características **dessas coisas que podem ser descritas com poucas palavras***
- ***Questão de suporte tecnológico***
  - ***Atributos não podem conter subelementos***
  - ***Atributos são mais fáceis de serem validados num DTD***

Identificadores

- ***Nomes de atributos e elementos***
- ***Podem** conter*
  - ***qualquer caractere alfanumérico ou ideograma***
  - ***. (ponto)***
  - ***- (hífen)***
  - ***_ (sublinhado)***
- ***Não podem** começar **com***
  - ***ponto,***
  - ***hífen ou***
  - ***número***

Identificadores (2)

- *Elementos bem formados*

```
<αριστοτελεσ>περι ποιητικησ</αριστοτελεσ>
<êíèãà xml:lang='ru'>
  <íàçâàíèå>Ââåäåíèé Îíåãèí</íàçâàíèå>
  <àâòîð ðîæäåíèå="1799"
         ñìåðòü="1837">Àëåêñàíäð Ñåðãååâè÷ Ïóøêèí</àâòîð>
</êíèãà>
<_1_/>
<cdd:gen.inf cdd:cod="005">Introdução a XML</cdd:gen.inf>
```

- *Elementos mal formados*

```
<3-intro>Fundamentos</3-intro>
<cartão de crédito>1234567887654321</cartão de crédito>
```


Conteúdo misto

```
<trecho>
<secao>2</secao>
<paragrafo>A unidade de informação
dentro de um documento XML é o
<definicao>elemento</definicao>. Um
elemento é formado por duas
<definicao>etiquetas</definicao> que
atribuem algum significado ao conteúdo.
</paragrafo>
</trecho>
```

Seção CDATA (Character DATA)

- *Ignora efeitos especiais dos caracteres*

```
<titulo>Curso de XML</titulo>
<exemplo>Considere o seguinte trecho de
XML:
<![CDATA[
      <empresa>
          <nome>João & Maria S/A</nome>
      </empresa>
]]>
</exemplo>
```


Instruções de processamento

- *Instruções dependentes do processador*
- *Funcionam como comentários para os processadores que não a conhecem*

```
<?nome-do-alvo área de dados ?>

<?query-sql select nome,
                    email
                    from agenda
                    where id=25 ?>
```

- *Iguais aos comentários HTML*

```
<!-- Isto é um comentário -->

<!--  isto é um erro -- sério! -->
```

Declaração XML

- *Opcional (exceto queno conjunto de caracteres usado for diferente de UTF-8*

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


Using namespaces in XML

- To fully qualify a namespace in XML write the namespace:tag name. eg.

  <my_namespace:tag> </my_namespace:tag>
- In a globally declared single namespace the qualifier may be omitted.
- More than one namespace:

  <my_namespace:tag> </my_namespace:tag>

  <your_namespace:tag> </your_namespace:tag>

  can co-exist if correctly qualified.

Using namespaces in XML

- To fully qualify a namespace in XML write the namespace:tag name. eg.

  <my_namespace:tag> </my_namespace:tag>
- In a globally declared single namespace the qualifier may be omitted.
- More than one namespace:

  <my_namespace:tag> </my_namespace:tag>

  <your_namespace:tag> </your_namespace:tag>

  can co-exist if correctly qualified.

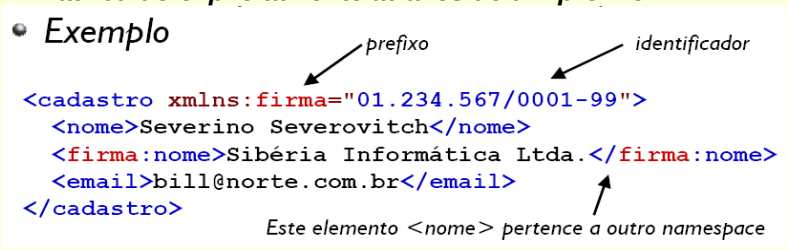
XML Namespaces

- *Permite que elementos de mesmo nome de diferentes aplicações sejam misturados sem que haja conflitos*
- *Um namespace (universo de nomes) é declarado usando atributos reservados*
  - *xmlns="identificador" (namespace default)*
    - *associa o identificador com todos os elementos que não possuem prefixo. Ex: <nome>*
  - *xmlns:prefixo="identificador"*
    - *associa o identificador com os elementos e atributos cujo nome local é precedido do prefixo. Ex <prefixo:nome>*
- *O prefixo é arbitrário e só existe dentro do documento*
- *O identificador (geralmente uma URI) deve ser reconhecido pela aplicação*

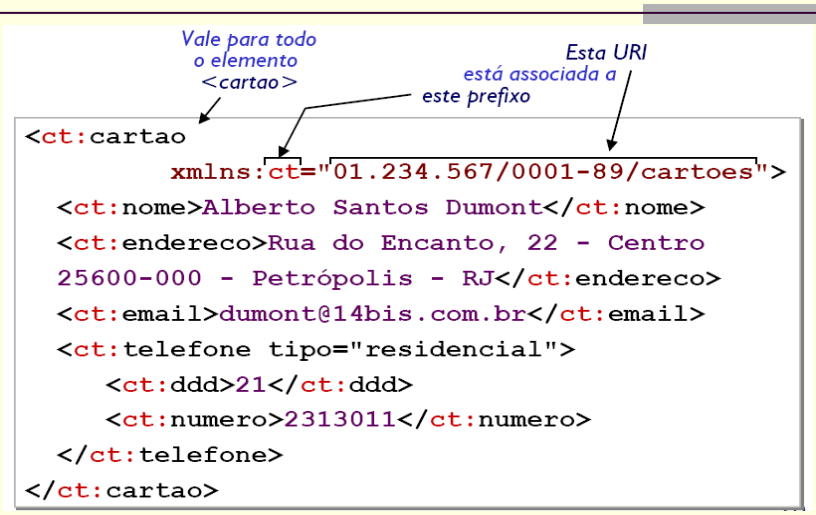
XML Namespaces

- ***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*



Outro Exemplo



Exemplo com 3 Namespaces

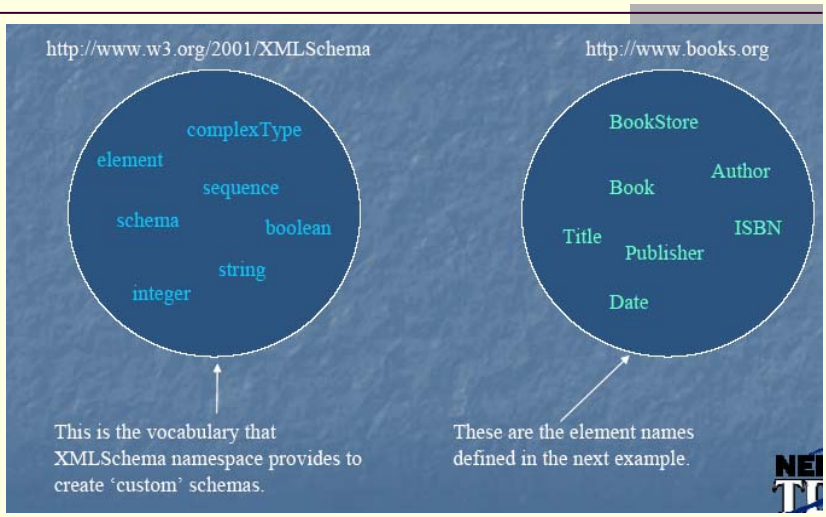
```
<departamento
    xmlns:ct="01.234.567/0001-89/cartoes"
    xmlns="01.234.567/0001-89/empresa"
    xmlns:html="http://www.w3.org/1999/xhtml">

  <ct:nome>Fulano de Tal</ct:nome>
  <nome>Contabilidade</nome>
  <endereco>Rua Projetada, 33</endereco>
  <html:a href="web.html">
     <html:strong>link negrito HTML</html:strong>
  </html:a>
  <urgencia><ct:numero>2313011</ct:numero></urgencia>
</departamento>
```

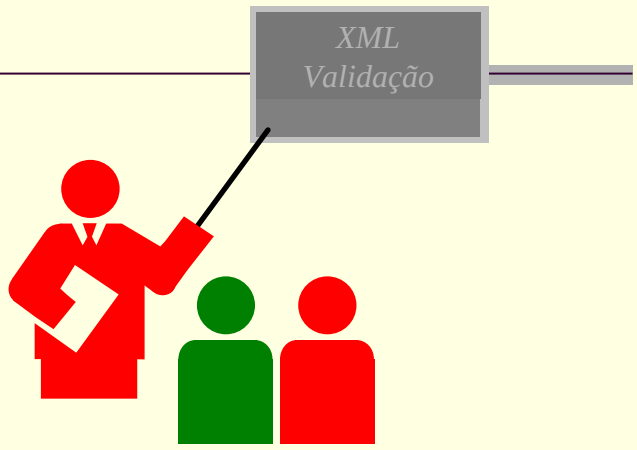
Namespace default

URI padrão XHTML

Namespace Ilustration



SOA



Problemas do XML

- An XML element name may not mean the same thing in different XML files
- XML on its own doesn XML on its own doesn't specify allowable elements
  - New elements may be added without breaking applications applications – but applications won't recognize them
- XML on its own doesn't mandate a structure
- XML on its own doesn't enforce data types

Documento XML bem-formado

- ***Documento bem-formado***
  - ***ter um único elemento raiz***
  - ***etiquetas iniciais e finais combinam (levando em conta que caracteres maiúsculos e minúsculos são diferentes)***
  - ***elementos bem aninhados***
  - ***valores de atributos entre aspas ou apóstrofes***
  - ***atributos não repetidos***
  - ***identificadores válidos para elementos e atributos***
  - ***comentários não devem aparecer dentro das etiquetas***
  - ***sinais < ou & nunca devem ocorrer dentro dos valores dos atributos ou nos nós de texto do documento.***

XML Solution – Schemas

- An XML Schema defines the legal building blocks of an XML document. It mandates: of an XML document. It mandates:
  - elements that can appear in a document
  - attributes that can be used for a given element
  - which elements are parent elements and which elements are parent elements and which elements are child elements
  - the order of child elements
  - data types for elements and attributes
  - allowable values for elements and attributes

Por que validar ?

- *Para a maior parte das aplicações, um XML bem formado é suficiente*
- *É possível, em documentos XML não 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*
- *Para que serve, então, o trabalho de desenvolver um DTD?*

Valid and well formed

- A correct XML document must be both valid and well formed.
- Well formed means that the syntax must be correct and all tags must close correctly (eg <…> </…>).
- Valid means that the document must conform to some XML definition (a DTD or Schema).

(Otherwise there can be no definition of what the tags mean)

Definindo um Esquema XML

- *Para se ter uma linguagem precisa-se de um **esquema***
  - *Linguagem implica comunicação.*
  - *Não há comunicação eficiente sem uma convenção entre as partes sobre vocabulários, regras de formação, etc.*
- *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*

Classes x Instâncias

- *Um esquema define uma classe de documentos*
  - *Os documentos que quiserem fazer parte dessa classe devem aderir ao esquema*
- *Um documento pode pertencer a várias classes*
  - *Um documento pode ser válido em vários esquemas*

Documentos Válidos

- *Documentos válidos contém*
  - *Declaração de tipo de documento (para DTD), ou*

```
<!DOCTYPE bilhete SYSTEM "bilhete.dtd">
```

  - *Declaração de namespace e schema (para XML Schema)*

```
<bilhete codigo="ZMIKT8"
xmlns="urn:123456789"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:123456789 bilhete.xsd">
```

- *Para validar*
  - *Use um validador (com suporte a Schema)*

O que define um Esquema XML

- *Um vocabulário*
  - *Elementos, atributos*
- *Uma gramática*
  - *Relacionamentos*
- *Uma coleção de entidades*
  - *No caso dos DTDs*

```
<pessoa>
  <nome>
    <prenome>Richard</prenome>
    <sobrenome>Feynman</sobrenome>
  </nome>
  <profissão>Físico</profissão>
  <profissão>Matemático</profissão>
  <profissão>Arrombador de cofres</profissão>
</pessoa>
```

DTD

```
<!ELEMENT pessoa (nome, profissao*)>
<!ELEMENT nome (prenome, sobrenome)>
<!ELEMENT prenome (#PCDATA)>
<!ELEMENT sobrenome (#PCDATA)>
<!ELEMENT profissao (#PCDATA)>
```

```
<!DOCTYPE pessoa SYSTEM
                "http://pessoas.com/pessoa.dtd">
<pessoa>
  <nome>
    <prenome>Richard</prenome>
    <sobrenome>Feynman</sobrenome>
  </nome>
</pessoa>
```

Documento Válido

Documentos não-válidos

- Diga porque os documentos são não-válidos ?

```
<!DOCTYPE pessoa SYSTEM
                  "http://pessoas.com/pessoa.dtd">
<pessoa>
  <nome>
    <prenome>Richard</prenome>
  </nome>
  <profissão>Arrombador de cofres</profissão>
</pessoa>
```

Falta elemento sobrenome

```
<!DOCTYPE pessoa SYSTEM
                  "http://pessoas.com/pessoa.dtd">
<pessoa>
  <profissão>Arrombador de cofres</profissão>
  <nome>
    <prenome>Richard</prenome>
    <sobrenome>Feynman</sobrenome>
  </nome>
  <profissão>Físico</profissão>
</pessoa>
```

Elemento profissão não pode vir antes do elemento nome

What is a Schema?

- A schema is the definition of the meaning of each of the tags within a XML document.
- *Analogy: A HTML style sheet can be seen as a limited schema which only specifies the presentational style of HTML which refers to it.*
- *Example: in HTML the tag pre-defined. In XML you would need to define this in the context of your document.*

Namespaces e XML Schema

- *DTDs validam "documentos", não "linguagens"*
  - *DTDs supõem que documentos contém apenas uma linguagem*
- *DTDs não suportam namespaces. Para declarar um elemento é preciso usar o nome qualificado:*

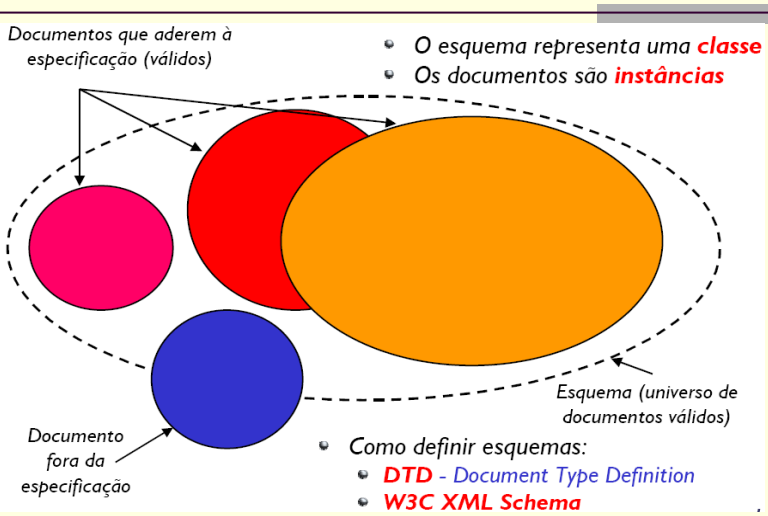
```
<!ELEMENT prefixo:elemento (#PCDATA) >
```

- *Para validar namespaces em vez de documentos, use XML Schema*
  - *Cada namespace pode ser associado a um esquema diferente (um documento pode ter vários esquemas)*
  - *DTDs ainda podem ser usados, para declarar, por exemplo, entidades.*

XML válido

- ***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***

Esquemas XML



- O esquema representa uma **classe**
- Os documentos são **instâncias**
- Como definir esquemas:
  - **DTD** - Document Type Definition
  - **W3C XML Schema**

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

Schema

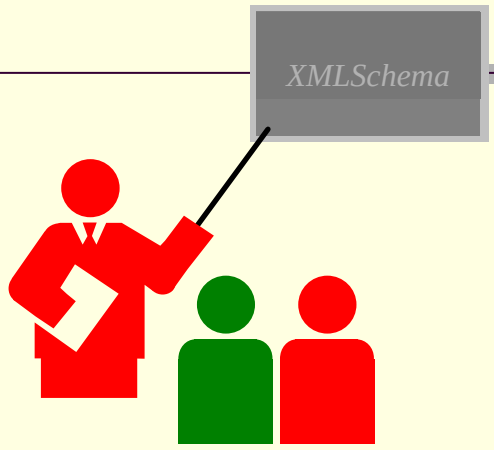
```
<?xml version="1.0"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
xmlns="document" >
<xs:element name = "DOCUMENT">
        <xs:element name="CUSTOMER"> </xs:element>
</xs:element>
</xs:schema>
```

Simple schema saved as order.xsd

```
<?xml version="1.0"?>
<DOCUMENT xmlns="document"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
Xsi:schemaLocation="order.xsd">
<DOCUMENT>
        <CUSTOMER>sam smith</CUSTOMER>
        <CUSTOMER>sam smith</CUSTOMER>
</DOCUMENT>
```

XML document derived from schema.

SOA



Namespaces e XML Schema

- *DTDs validam "documentos", não "linguagens"*
  - *DTDs supõem que documentos contém apenas uma linguagem*
- *DTDs não suportam namespaces. Para declarar um elemento é preciso usar o nome qualificado:*

```
<!ELEMENT prefixo:elemento (#PCDATA) >
```

- *Para validar namespaces em vez de documentos, use XML Schema*
  - *Cada namespace pode ser associado a um esquema diferente (um documento pode ter vários esquemas)*
  - *DTDs ainda podem ser usados, para declarar, por exemplo, entidades.*

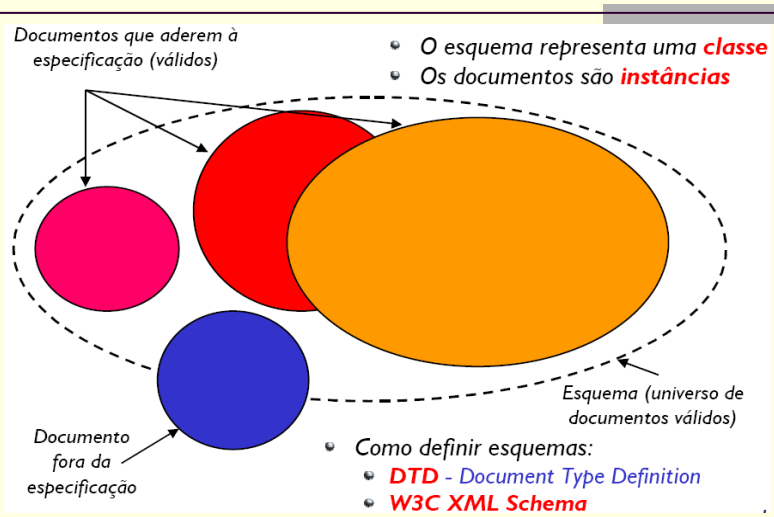
Documentos XML bem formados

- ***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***

XML válido

- ***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***

Esquemas XML



Basic XML Schema concepts

- Syntax is not the Schema
- Namespaces are fundamental
- But a schema is not a namespace
- Separation of tag from type
- Simple and Complex types
- Modular Schema construction
- Powerful type construction
- Local tag-type association
- Powerful wildcards
- Element equivalence classes
- Extension mechanism
- Documentation mechanism

XML Schemas

- **What is an XML Schema?**
- The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

- An XML Schema:
  - defines elements that can appear in a document
  - defines attributes that can appear in a document
  - defines which elements are child elements
  - defines the order of child elements
  - defines the number of child elements
  - defines whether an element is empty or can include text
  - defines data types for elements and attributes
  - defines default and fixed values for elements and attributes

XML Schemas

- **XML Schemas are the Successors of DTDs**
- XML Schemas will be used in most Web applications as a replacement for DTDs. Here are some reasons:
  - XML Schemas are extensible to future additions
  - XML Schemas are richer and more powerful than DTDs
  - XML Schemas are written in XML
  - XML Schemas support data types
  - XML Schemas support namespaces

XML Schemas

- **XML Schemas Support Data Types**
- One of their greatest strengths
- With support for data types:
  - It is easier to describe allowable document content
  - It is easier to validate the correctness of data
  - It is easier to work with data from a database
  - It is easier to define data facets (restrictions on data)
  - It is easier to define data patterns (data formats)
  - It is easier to convert data between different data types

XML Schemas

- **XML Schemas use XML Syntax**
  - Schemas are XML documents
- Benefits of Schemas as XML docs
  - You don't have to learn a new language
  - You can use your XML editor to edit your Schema files
  - You can use your XML parser to parse your Schema files
  - You can manipulate your Schema with the XML DOM
  - You can transform your Schema with XSLT

XML Schemas

- **XML Schemas are Extensible**
  - XML Schemas are extensible, because XML is extensible
  - With an extensible Schema definition you can:
    - Reuse your Schema in other Schemas
    - Create your own data types derived from the standard types
    - Reference multiple schemas in the same document

XML Schemas

- **Well-Formed is not Enough**
- A well-formed XML document is a document that conforms to the XML syntax rules, like:
  - it must begin with the XML declaration
  - it must have one unique root element
  - start-tags must have matching end-tags
  - elements are case sensitive
  - all elements must be closed
  - all elements must be properly nested
  - all attribute values must be quoted
  - entities must be used for special characters

XML Schemas

- Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.
- Think of the following situation: you order 5 gross of laser printers, instead of 5 laser printers. With XML Schemas, most of these errors can be caught by your validating software.

XML Schemas

- **Simple XML Document "note.xml":**

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

<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```


XML Schemas

- **An XML Schema**
  - The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):
  - The note element is a **complex type** because it contains other elements. The other elements (to, from, heading, body) are **simple types** because they do not contain other elements.

XML Schemas

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

<xs:schema xmlns:xs= “http://www.w3.org/2001/XMLSchema”
targetNamespace=     “http://www.w3schools.com”
xmlns=               “http://www.w3schools.com”
elementFormDefault=  "qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to"      type="xs:string"/>
      <xs:element name="from"    type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body"    type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```


XML Schemas

- This XML document has a reference to a Schema:

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

<note
  xmlns= “http://www.w3schools.com”
  xmlns:xsi="http://www.w3.org/2001/XMLSchema instance“
  xsi:schemaLocation="http://www.w3schools.com note.xsd">

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don'tforget me this weekend!</body>
</note>
```

XML Schemas

- **The <schema> Element**
  - The <schema> element is the root element of every XML Schema:

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

<xs:schema>
  ...
  ...
</xs:schema>
```


XML Schemas

- The <schema> element may contain some attributes. A schema declaration often looks something like this:

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs=           "http://www.w3.org/2001/XMLSchema"
  targetNamespace=    "http://www.w3schools.com"
  xmlns=              "http://www.w3schools.com"
  elementFormDefault= "qualified">
 ...
 ...
 </xs:schema>
```

XML Schemas

- The following fragment:

`xmlns:xs= “http://www.w3.org/2001/XMLSchema”`

  - Indicates that the elements and data types used in the schema come from the `“http://www.w3.org/2001/XMLSchema"` namespace.
  - It also specifies that the elements and data types that come from the `“http://www.w3.org/2001/XMLSchema"` namespace should be prefixed with **xs:**

`targetNamespace="http://www.w3schools.com"`

  - Indicates that the elements defined by this schema (note, to, from, heading, body.) come from the target namespace.

XML Schemas

`xmlns=“http://www.w3schools.com”`

- Indicates the default namespace

`elementFormDefault="qualified"`

- Indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

XML Schemas

- This XML document has a reference to a Schema:

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

<note
  xmlns= “http://www.w3schools.com”
  xmlns:xsi="http://www.w3.org/2001/XMLSchema instance“
  xsi:schemaLocation="http://www.w3schools.com
   note.xsd">

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don'tforget me this weekend!</body>
</note>
```


XML Schemas

`xmlns="http://www.w3schools.com"`

- Specifies the default namespace declaration.
- Tells the schema-validator that all the elements used in this XML document are declared in this namespace.

- Once the XML Schema Instance namespace is available:

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

- Can use the `schemaLocation` attribute. The first value is the namespace to use. The second value is the location of the XML schema to use for that namespace:

  xsi:schemaLocation="http://www.w3schools.com note.xsd"

XML Schemas

- Schemas define the elements of your XML files.
- Simple element is an XML element that contains only text. It cannot contain any other elements or attributes.
- The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.
- You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

XML Schemas

- The syntax for defining a simple element is:

```
<xs:element name="xxx" type="yyy"/>
```

- Where xxx is the name of the element and yyy is the data type of the element.
- XML Schema has a lot of built-in data types. The most common types are:
  - `xs:string`
  - `xs:decimal`
  - `xs:integer`
  - `xs:boolean`
  - `xs:date`
  - `xs:time`

XML Schemas

- Here are some simple XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

- Here are the corresponding simple element definitions:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age"
  type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```


XML Schemas

- Simple elements may have a default value OR a fixed value specified.
- Default value is automatically assigned to the element when no other value is specified. In the following example the default value is "red":

```
<xs:element name="color" type="xs:string" default="red"/>
```

- Fixed value is also automatically assigned to the element, and you cannot specify another value. In the following example the fixed value is "red":

```
<xs:element name="color" type="xs:string" fixed="red"/>
```

XML Schemas

- The syntax for defining an attribute is:
  `<xs:attribute name="xxx" type="yyy"/>`
  - Where xxx is the name of the attribute and yyy specifies the data type of the attribute.
  - Simple elements can't have attributes!
- Here is an XML element with an attribute:
  `<lastname lang="EN">Smith</lastname>`
- Here is the corresponding attribute definition:
  `<xs:attribute name="lang" type="xs:string"/>`
- Attributes can have default or fixed values. If the attribute is required, add `use="required"`

XML Schemas

- When an XML element or attribute has a data type defined, it puts restrictions on the element's or attribute's content.
- If an XML element is of type "xs:date" and contains a string like "Hello World", the element will not validate.
- With XML Schemas, you can also add your own restrictions to your XML elements and attributes.

XML Schemas

- The following example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 120:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```


XML Schemas

- The example below defines an element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW:

```
<xs:element name="car" type="carType"/>
  <xs:simpleType name="carType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
```

- **Note:** In this case the type "carType" can be used by other elements because it is not a part of the "car" element.

Schema example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.books.org"
           xmlns=http://www.books.org>
  <xs:element name="BookStore">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Book" minOccurs="1"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Title" type="xs:string" />
        <xs:element name="Author" type="xs:string" />
                ...
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The elements and datatypes used to construct schemas come from the XMLSchema namespace. This statement specifies that items from this namespace will be prefixed with xs:

Schema example (cont´d)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.books.org"
           xmlns=http://www.books.org>
  <xs:element name="BookStore">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Book" minOccurs="1"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Title" type="xs:string" />
        <xs:element name="Author" type="xs:string" />
                ...
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The elements defined by this schema go in the http://www.books.org namespace

The default namespace is http://www.books.org which is the targetNamespace

This reference does not specify a namespace, so it refers to the Book element declared in the default namespace

Using the schema

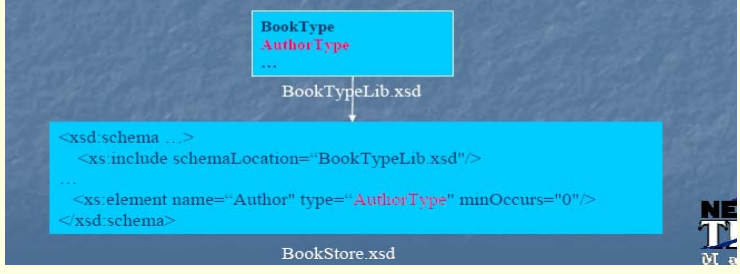
```
<?xml version="1.0"?>
<BookStore xmlns ="http://www.books.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.books.org BookStore.xsd">
      <Book>
            <Title>My Life and Times</Title>
            <Author>Paul McCartney</Author>
            <Date>July, 1998</Date>
            <ISBN>94303-12021-43892</ISBN>
            <Publisher>McMillin Publishing</Publisher>
      </Book>
      ...
</BookStore>
```

The default namespace is www.books.org – elements in the file come from this schema

Identifies the location of the www.books.org schema

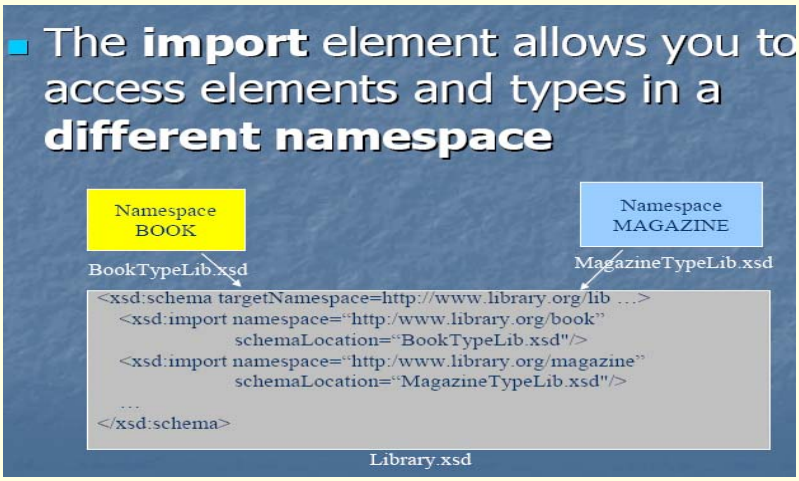
Combining Schemas

- The include element allows you to reference components in other schemas
  - All the schemas you include **must** have the **same namespace** as the including schema



Combining Schemas

- The **import** element allows you to access elements and types in a **different namespace**



Types and Type Derivation

- For purposes of discussion, consider only the content type aspects of types (attributes are analogous)
- A content type definition (simple or complex) consists of a set of constraints on what's allowed as content.

Built-In Data Types

- Data types defined by the schema namespace

xs:string	"Hello World"
xs:boolean	{true, false, 1, 0}
xs:decimal	7.08
xs:dateTime	format: *CCYY-MM-DDThh:mm:ss*
xs:time	format: *hh:mm:ss.sss*
xs:date	format: *CCYY-MM-DD*
xs:anyURI	http://www.myweboage.com
xs:integer	456
xs:int	-2147483648 to 2147483647

Permissions and obligations

- You can think of the type itself as the set of strings/EIIs its constraints allow. It's helpful to think of constraints as composed of obligations and permissions:

  `(\d )?(\d{3}-)?\d{3}-\d{4}`

  - regexp definition facet for [US] 'phone number type
- the ? and the \d can be seen as permissions, the - and the {3} as obligations
  - 1 337-6818 and 207-422-6240 belong to this type

Simple DataType example

- Type declaration

```
< xs:simpleType name="CustomerNameType">
  <xs:annotation>
    <xs:documentation>Subscriber name associated
with the Calling Party Number.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

Data Type Name

Data Type Description

Data Type Definition

- Using the new type

```
<xs:element name="CustomerName"
type="tns:CustomerNameType"/>
```

Name attribute value is the XML tag

Type attribute value must be a valid schema datatype

- Example XML

```
<CustomerName>Gomer Pyle</CustomerName>
```

NENA

Datatype Extensions

- The built in datatypes may have additional restrictions added to create a 'new' datatype.
- A 'typical' example:

```
<xs:simpleType name="AlsoRingsAtAddressType">
  <xs:annotation>
    <xs:documentation>Additional textual address
for the CPN</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:maxLength value="60"/>
  </xs:restriction>
</xs:simpleType>
```

Data Type Name

Data Type Description

"Base" type

May not be longer than 60 characters

Restricting values in na Element

```
<xs:simpleType name="ClassOfServiceCodeType">
   <xs:restriction base="xs:string">
         <xs:length value="1"/>
         <xs:enumeration value="1"/>
         <xs:enumeration value="2"/>
         <xs:enumeration value="3"/>
         <xs:enumeration value="4"/>
         <xs:enumeration value="5"/>
         <xs:enumeration value="6"/>
         <xs:enumeration value="7"/>
         <xs:enumeration value="8"/>
         <xs:enumeration value="9"/>
         <xs:enumeration value="0"/>
         <xs:enumeration value="A"/>
         <xs:enumeration value="B"/>
         <xs:enumeration value="G"/>
         <xs:enumeration value="H"/>
   </xs:restriction>
</xs:simpleType>
```

Data Type Name
"Base" type
Must be 1 char in length
May only be one of these enumerated values

An integer extension

```
<xs:simpleType name="ConfidencePercentType">
  <xs:annotation>
    <xs:documentation>Confidence percentage that
calling party lies within the
shape.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:integer">
    <xs:maxInclusive value="100"/>
    <xs:minInclusive value="0"/>
  </xs:restriction>
</xs:simpleType>
```

Data Type Name
"Base" type
Must be an integer with a value between 0 and 100

Specifying Attributes

```
<xs:complexType name="GeneralUseType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ID" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Types with attributes must be complexType
No child elements
GeneralUse is a string
Attribute is required; ID is the attribute tag name
The attribute ID must contain a string value

- Example XML

```
<GeneralUse ID="HBFUSOC">Valid USOC Code</GeneralUse>
```


XML Schemas

- **What is a Complex Element?**
  - A complex element is an XML element that contains other elements and/or attributes.
  - There are four kinds of complex elements:
    - empty elements
    - elements that contain only other elements
    - elements that contain only text
    - elements that contain both other elements and text
- **Note:** Each of these elements may contain attributes as well!

XML Schemas

- A complex XML element, "food", which contains only text:

```
<food tye="dessert">Ice cream</food>
```

- A complex XML element, "description", which contains both elements and text:

```
<description> It happened on
    <date lang="norwegian">03.03.99</date> ....
</description>
```


XML Schemas

- A complex XML element, "product", which is empty:

```
<product pid="1345"/>
```

- A complex XML element, "employee", which contains only other elements:

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

XML Schemas

- A complex XML element, "food", which contains only text:

```
<food tye="dessert">Ice cream</food>
```

- A complex XML element, "description", which contains both elements and text:

```
<description> It happened on
    <date lang="norwegian">03.03.99</date> ....
</description>
```


XML Schemas

- A complex XML element, "description", which contains both elements and text:

```
<description> It happened on
   <date lang="norwegian">03.03.99</date> ....
</description>
```

Complex types

```
(title?,forename*,surname)
```

- (shorthand for) content model for **name**

- the ? can be seen as permission, the , and the 'surname' as obligations (at the end of the day, each component involves both permission AND obligation, but the balance of impact is as suggested)

XML Schemas

- A complex XML element, "food", which contains only text:

```
<food tye="dessert">Ice cream</food>
```

- A complex XML element, "description", which contains both elements and text:

```
<description> It happened on
    <date lang="norwegian">03.03.99</date> ....
</description>
```

XML Schemas

- A complex XML element, "description", which contains both elements and text:

```
<description> It happened on
   <date lang="norwegian">03.03.99</date> ....
</description>
```


XML Schemas

- **How to Define a Complex Element**
  - Look at this complex XML element, "employee", which contains only other elements:

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
 </employee>
```

XML Schemas

2. The "employee" element can have a type attribute that refers to the name of the complex type to use:

```
<xs:element name="employee" type="personinfo"/>
 <xs:complexType name="personinfo">
   <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname"  type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
```


XML Schemas

- If you use the 2nd method, several elements can refer to the same complex type, like this:

```
<xs:element name="employee" type="personinfo"/> <xs:element
 name="student" type="personinfo"/> <xs:element
 name="member" type="personinfo"/>

<xs:complexType name="personinfo">
 <xs:sequence>
  <xs:element name="firstname" type="xs:string"/>
  <xs:element name="lastname" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Enforcing Structure

- XML Schemas enforce child/parent relationships.
- XML Schemas can also specify whether elements are required or optional.
- The <xs:complexType> in conjunction with the <xs:sequence> or <xs:all> tags are used to specify structure.

Parent/child example

```
<xs:complexType name="TNInfoType">
  <xs:annotation>
    <xs:documentation>Type definition for set of
elements related to the TN and
Customer</xs:documentation>
  </xs:annotation>
  <xs:all>
    <xs:element name="CallingPartyNum"
type="tns:CallingPartyNumType"/>
    <xs:element name="CustomerName"
type="tns:CustomerNameType" minOccurs="0"/>
    <xs:element name="CustomerCode"
type="tns:CustomerCodeType"
    <xs:element name="ESN" type="tns:ESNType"
minOccurs="0"/>
</xs:all>
</xs:complexType>
```

Data Type Name

Data Type Description

Specifies that child elements follow in any order

Specifies an element with a tag "Customer Name" and data type CustomerNameType

Specifies that this is an optional element

- Using the data type

```
<xs:element name="TNInfo" type="NENA-SO-TNInfoType"/>
```

- Example XML

```
<TNInfo>
  <CallingPartyNum>5124810911</CallingPartyNum>
  <CustomerName>Gomer Pyle</CustomerName>
  <CustomerCode>abc</CustomerCode>
  <ESN>00305</ESN>
</TNInfo>
```

Children specified in the TNInfoType declaration

Specifying repeating elements

```
<xs:complexType name="ErrorsType">
  <xs:sequence>
    <xs:element name="ErrorCode" type="NENA-
SO:ErrorCodeType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

<xs:sequence> used for repeating elements

May have 0 to n ErrorCode elements

- Using the data type

```
<xs:element name="ReturnErrors" type="NENA-
SO:ErrorsType" minOccurs="0"/>
```

ReturnErrors is an optional element

- Example XML

```
<ReturnErrors>
  <ErrorCode>010</ErrorCode>
  <ErrorCode>011</ErrorCode>
</ReturnErrors>
```

Children specified in ErrorsType – may have 0 to n ErrorCodes within the optional ReturnErrors parent

Complex types, cont'd

```
(title?,forename*,surname)
 <name>
    <forename>...</forename>
    <surname>...</surname>
  </name>
```

- and

```
 <name>
    <title>...</title>
    <surname>...</surname>
  </name>
```

- are both members of this type

Restriction

- A type definition may be a restriction of another type's definition if it reduces permissions, sometimes to the point of inducing obligations:

  `\d[01]\d-\d{3}-\d{4}` (a restriction
  `(\d )?(\d{3}-)?\d{3}-\d{4}` of US p#)
- The membership of this type, which includes
  - 207-422-6240 but not 1 337-6818
- is a (proper) subset of the membership of the original type,
- because by construction every member of the new type is a member of the original.

Restriction, cont'd

- Similarly,

  `(forename+,surname)`
- is a restriction of the original type definition for **name**

  `(title?,forename*,surname)`
- and the same relation holds.

Restriction, cont'd

- Note first that

```
(forename+,surname)
```

```
<name>
   <forename>...</forename>
   <surname>...</surname>
  </name>
```

- is a member of the new type, but

```
<name>
   <title>...</title>
   <surname>...</surname>
  </name>
```

- is not.

Extension

- Now consider

```
(title?, forename*, surname, genMark?)
```

- This type extends the original type definition for **name**.

```
<name>
   <forename>Al</forename>
   <surname>Gore</surname>
   <genMark>Jr</genMark>
</name>
```

- is an instance of this new type, but not of the original.

Any

- Finally note that the <any/> content model particle, in all of its forms, introduces particularly broad permissions into complex content types.

Where are we headed?

- A number of design decisions can now be stated:
- Should we make it easy to construct type definitions which restrict or extend other type definitions, by specifying only the method of derivation and the differences between the source and derived type definitions?
- The new proposal says 'yes', you do this by using the "`source`" and "`derivedBy`" attributes on your `<type>` or `<datatype>` element.

Datatype example

- Consider the simple type case first:

```
<datatype name='bodytemp'
          source='decimal'>
   <precision value='4'/>
   <scale value='1'/>
   <minInclusive value='97.0'/>
   <maxInclusive value='105.0'/>
</datatype>
<datatype name='healthyBodytemp'
        source='bodytemp'>
   <maxInclusive value='99.5'/>
</datatype>
```


Derived type

- The **healthyBodytemp** type definition is defined by closing down the permitted range of **bodytemp**. We say it 'inherits' the other facets of **bodytemp**, so the 'effective type definition' of **healthyBodytemp** is

Effective type

```
<datatype name='healthyBodytemp'
          source='decimal'>
  <precision value='4'/>
  <scale value='1'/>
  <minInclusive value='97.0'/>
  <maxInclusive value='99.5'/>
</datatype>
```

- Since it doesn't in general make sense to extend one simple type by another, the `"derivedBy"` attribute is actually redundant for `<datatype>`.

Extension for complex types

- The next simplest case is extension for complex types:

```
<type name='name'>
  <element name='title' minOccurs='0'/>
  <element name='forename'
           minOccurs='0' maxOccurs='*'/>
  <element name='surname'/>
</type>
```

- Derived type

```
<type name='fullName'
      source='name'
      derivedBy='extension'>
  <element name='genMark'
           minOccurs='0'/>
</type>
```

The effective type

```
<type name='fullName'>
   <element name='title'
            minOccurs='0'/>
   <element name='forename'
            minOccurs='0'
            maxOccurs='*'/>
   <element name='surname'/>
   <element name='genMark'
            minOccurs='0'/>
  </type>
```


Restriction for complex types

- Restriction for complex types is harder to handle syntactically, because of the significance of linear order in content models, but the semantics are completely parallel to the simple type case:

Restriction example

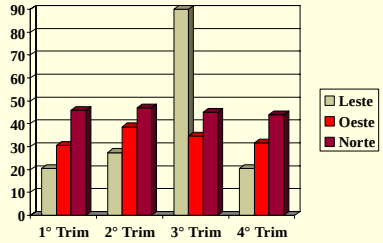
- Restriction for complex types is harder to handle syntactically, because of the significance of linear order in content models, but the semantics are completely parallel to the simple type case:

```
<type name='simpleName'
         source='name'

   derivedBy='restriction'>
      <restrictions>
       <element name='title'

   maxOccurs='0'/>
       <element
   name='forename'

   minOccurs='1'/>
      </restrictions>
     </type>
```



Restriction and Inheritance

- Just as in the `<datatype>` case, the content model aspects not mentioned are left alone, including the `"maxOccurs='*'"` on `<forename>` and the whole particle for `<surname>`, so the 'effective content model' of 'simpleName' is

Effective type

```
<type name='simpleName'>
   <element name='title'
            maxOccurs='0'
            minOccurs='0'/>
 <!-- i.e. forbidden -->
   <element name='forename'
            minOccurs='1'
            maxOccurs='*'/>
   <element name='surname'/>
  </type>
```


Instances

- Given all the example definitions above, all of

```
<name>
 <title>Ms</title>
 <surname>Steinem</surname>
</name>

<name xsi:type='simpleName'>
   <foreName>Harry</foreName>
   <foreName>S</foreName>
   <surname>Truman</surname>
 </name>
```

Another instance

```
<name xsi:type='fullName'>
   <forename>Al</forename>
   <surname>Gore</surname>
   <genMark>Jr</genMark>
 </name>
```

- all would be schema-valid per

```
<element name='name' type='name'/>
```


Connecting Instances and Schemas

- A schema is not a namespace
- The connection *cannot* be made rigid
- The draft identifies three layers, first is
  - **`schema-valid(EII,TypeName,ComponentSet)`**
- The TypeName is a (namespaceURI,NCName) pair
- The component set is made up of (namespaceURI,NCName,component) triples

fim