Anno Accademico 2007-2008

Corso di Tecnologie Web
Introduzione a XML

http://www-lia.deis.unibo.it/Courses/TecnologieWeb0708/

Il linguaggio XML

- Linguaggio di markup aperto
  - eXtensible Markup Language
- Metalinguaggio creato e gestito dal W3C
  - sviluppato dall'XML Working group costituito nel nel 1996
- Derivato dal SGML (Standard Generalized Markup Language)
- Fornisce informazioni sui dati di tipo strutturale e semantico

XML vs HTML

> HTML nasce per la rappresentazione standardizzata delle informazioni per il web

> HTML è rivolto alla *rappresentazione* dei dati

> In HTML non è presente alcuna informazione sul tipo o la struttura dei dati

> In HTML si sono nel tempo creati marcatori non standard per introdurre nuove funzionalità di presentazione che erano dipendenti dal browser

> XML nasce per l'interscambio dei dati tra sistemi diversi (non è, quindi, in generale, un linguaggio rivolto alla presentazione delle informazioni)

> XML è *estensibile*, cioè possono essere definiti nuovi tag e attributi

> La struttura di un documento XML può essere vista in modo gerarchico nidificando i tag

> Ogni documento XML può contenere una opzionale descrizione della sua grammatica, in modo che possa essere utilizzata da applicazioni che richiedono una validazione della struttura del documento

Obiettivi progettuali del XML

> XML deve essere utilizzabile in modo semplice su Internet: in primo luogo XML deve operare in maniera efficiente su Internet e soddisfare le esigenze delle applicazioni eseguite in un ambiente di rete distribuito

> XML deve supportare un gran numero di applicazioni

> XML deve essere compatibile con SGML: questo obiettivo è stato definito sulla base del presupposto che un documento XML valido debba anche essere un documento SGML valido, in modo tale che gli strumenti SGML esistenti possano essere utilizzati con l'XML e siano in grado di analizzare il codice XML

> Deve essere facile lo sviluppo di programmi che elaborino documenti XML: l'adozione del linguaggio è proporzionale alla disponibilità di strumenti e la proliferazione di questi ultimi è la dimostrazione che questo obiettivo è stato raggiunto

> Il numero di caratteristiche opzionali deve essere ridotto al minimo possibile

Obiettivi progettuali del XML

> I documenti XML dovrebbero essere leggibili da un utente: poiché utilizza testo per descrivere i dati e le relazioni tra i dati, XML è più semplice da utilizzare e da leggere del formato binario che esegue la stessa operazione

> La progettazione di strutture dati XML dovrebbe essere rapida: XML è stato sviluppato per soddisfare l'esigenza di un linguaggio estensibile per il Web

> La progettazione di XML deve essere formale e concisa: questo obiettivo deriva dall'esigenza di rendere il linguaggio il più possibile conciso, formalizzando la formulazione della specifica

> I documenti XML devono essere facili da creare: i documenti XML possono essere creati facendo ricorso a strumenti di semplice utilizzo, quali editor di testo

Sintassi XML

> Simile a quella HTML ma più rigida e rigorosa

> La caratteristica sopra citata aumenta la leggibilità del documento e consente di creare *parser* che abbiano comportamenti univoci e deterministici

> Consente la definizione di elementi e attributi personalizzati

> E' possibile creare un modello, chiamato Document Type Definition (DTD), che descrive la struttura e il contenuto di una classe di documenti

> Diversi documenti possono utilizzare i medesimi tag (necessità di un NameSpace)

Struttura XML

Struttura XML: Struttura logica

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

Processing Instruction

```
<!DOCTYPE Wildflowers SYSTEM "Wldflr.dtd">
```

Dichiarazione tipo documento (DTD)

```
<PLANT>
      <COMMON>Columbine</COMMON>
<BOTANICAL>Aquilegia canadensis</BOTANICAL> </PLANT>
```

DOCUMENTO

Struttura XML: Le entità in XML

> Rappresentano unità di memorizzazione

> Necessitano di un nome univoco

> Possono essere *esterne* o *interne*

> I riferimenti ad una entità indicano al *parser* di recuperarne il valore definito

Struttura XML: Le entità in XML

> Entità analizzabili (parsed)

- viene letta e sostituita con il corrispondente oggetto richiamato

```
<!ENTITY myName "Paolo Rossi">
```

> Entità NON analizzabili (unparsed)

- dati binari o testi non conformi alle regole XML. L'oggetto di riferimento viene letto da un'applicazione in grado di farlo

```
<!ENTITY MyImage SYSTEM "image.gif" NDATA GIF>

<!NOTATION GIF SYSTEM "gifviewer.exe">
```

> Il contenuto di ogni entità viene aggiunto al documento ogni volta che viene fatto riferimento a quell'entità. Il riferimento ha la funzione di segnaposto per l'autore del contenuto e l'elaboratore di XML colloca il contenuto effettivo nei punti di riferimento. Per includere un riferimento si inserisce il simbolo "&" seguito dal nome dell'entità e da ";".

```
<Name>Nome: &myName;</Name>
```

Struttura XML: Le entità in XML

- Esistono, definite dal linguaggio, un set di entità predefinite
  - < < (parentesi angolare di apertura)
  - > > (parentesi angolare di chiusura)
  - &amp ; & (e commerciale)
  - ' ‘ (apostrofo)
  - " " (virgolette doppie)

Struttura XML: Le entità in XML

> Entità interne (dichiarate all'interno del documento)

```
<!ENTITY LR1 "light requirement: mostly shade">
```

> Entità esterne (fanno riferimento ad un file esterno al documento)

```
<!ENTITY MyImage SYSTEM "http://www.mySite.com/image.gif"
NDATA GIF>
```

Sintassi XML: gli elementi TAG

> Tutti i tag DEVONO essere chiusi (HTML meno rigido al riguardo)

> I Nomi dei tag devono iniziare con una lettera o underscore seguito da un numero qualsiasi di lettere, numeri, trattini (dashes), punti e underscore

> I nomi dei tag sono *case sensitive*

> I nomi non possono contenere spazi

> Il carattere ":", anche se tecnicamente legale, dovrebbe essere evitato perché riservato per i Namespace

> TAG di elemento Vuoto

```
<EMPTY></EMPTY>

<EMPTY/>
```

Sintassi XML: gli elementi Attributi

> Gli attributi forniscono informazioni addizionali relativamente ad un elemento aggiungendo caratteristiche all'elemento in questione

> Consistono di coppie chiave=valore

> Gli attributi sono contenuti all'interno di *start tag element*

> Un elemento può avere molteplici attributi

> I valori degli attributi devono essere racchiusi da *single* o *double quote*

```
<A HREF = "http://java.sun.com">
java Home Page
</A>
```

```
<?xml version="1.0"?>
<!DOCTYPE Wildflowers SYSTEM "Wldflr.dtd">

<PLANT ZONE="3">
<COMMON>Columbine</COMMON>
<BOTANICAL>Aquilegia canadensis</BOTANICAL> </PLANT>
```

Sintassi XML: *well formed VS valid*

- Documento **VALIDO**: deve rispettare le regole imposte dal DTD
- Documento *WELL FORMED*
  - La dichiarazione XML (se c'è) deve comparire come primo elemento del documento
  - Elementi che contengono dati devono avere sia il tag di apertura che quello di chiusura
  - Elementi che non contengono dati e utilizzano un tag singolo devono terminare con "/>"
  - il documento deve contenere solamente un elemento che contiene completamente tutti gli altri (un solo root element)
  - Gli elementi possono essere innestati ma non sovrapposti
  - I valori degli attributi devono essere messi tra apici
  - i caratteri < and & possono essere utilizzati solamente come inizio di un tag e per le entity

Sintassi XML: esempi

```
<list>
<item>Item 1</item>
<item>Item 2</item>
<item>Item 3</item>
</list>
```

Solo un elemento di root è ammesso

```
???
<item>Item 1</item>
<item>Item 2</item>
<item>Item 3</item>
???
```

```
<list>
<item>Car</item>
<ITEM>Plane</ITEM>
<Item>Train</Item>
</list>
```

```
<list>
 <item>Car</itm>
<item>Plane</ITEM>
<item>Train</item>
<list>
```

```
<forbidenNames>
  <xmlTag/>
  <XMLTag/>
  <XmLTag/>
  <xMlTag/>
  <xmLTag/>
</forbidenNames
>
```

```
<text>
  <bold><italic>XML</bold></italic>
</text>
```

Sintassi XML: esempi

```
<example>
  <isLower>
     23 < 46
  </isLower>
  <ampersand>
    Willey & sons
  </ampersand>
</example>
```

```
<!-- doc A -->
<example>
  <!-- <HEAD>  -->
  <!-- Characters <&<  -->
</example>
```

```
<example>
<right-bracket> both > and &gt; permitted</right-bracket>
<double-quote> both " and &quot; permitted</double-quote>
<apostrophe> both ' and &apos; permitted</apostrophe>
Useful in: <el value=" &apos; &quot; &apos; "/>
</example>
```

Sintassi XML: esempi

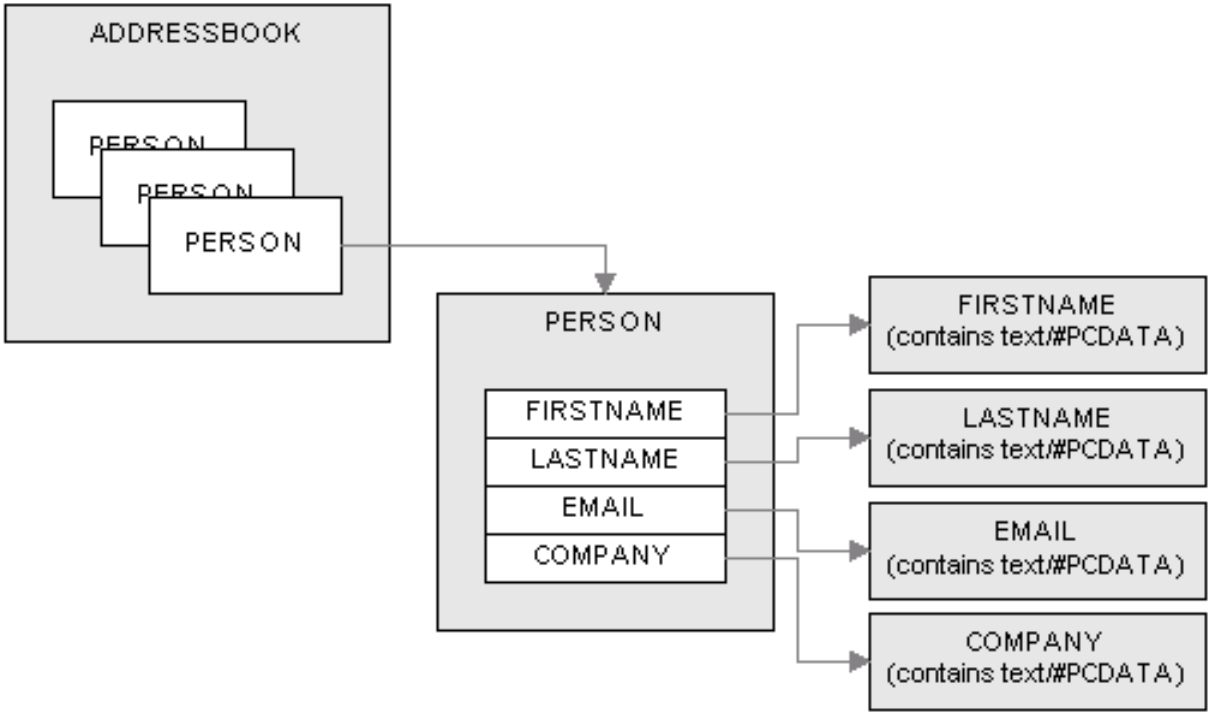
> CDATA: sezioni di testo che il parser XML non tenta di interpretare

```
<example>
  <![CDATA[ <aaa>bb&cc<<<]]>
</example>
```

Definizione Tipo Documento (DTD)

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

<!DOCTYPE ADDRESSBOOK [
<!ELEMENT ADDRESSBOOK (PERSON)*>
<!ELEMENT PERSON (LASTNAME, FIRSTNAME, EMAIL, COMPANY)>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
<!ELEMENT COMPANY (#PCDATA)> ]>
```



Definizione Tipo Documento (DTD)

DTD

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

il documento può contenere solamente l'elemento di root `tutorial` che a sua volta può contenere del testo

corretto

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

<tutorial>This is an XML document</tutorial>
```

errato

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

<text>This is an XML document</text>
```

Definizione Tipo Documento (DTD)

DTD

```
<!ELEMENT XXX (AAA , BBB)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
```

l'elemento di root `XXX` deve contenere esattamente un elemento `AAA` seguito da un elemento `BBB`. `AAA` e `BBB` possono contenere testo ma non altri elementi

corretto

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">
        <XXX>
        <AAA>Start</AAA>
        <BBB>End</BBB>
        </XXX>

 <!DOCTYPE tutorial SYSTEM "tutorial.dtd">
        <XXX> <AAA/> <BBB/> </XXX>
```

errato

BBB mancante

```
        <XXX> <AAA/> ___ </XXX>
```

BBB deve seguire **AAA**:

```
        <XXX> <BBB/> <AAA/> </XXX>
```

XXX può contenere solo un elemento **BBB**:

```
        <XXX> <AAA/> <BBB/> <BBB/> </XXX>
```

XXX non può contenere testo

```
        <XXX> Elements: <AAA/> <BBB/>  </XXX>
```

Definizione Tipo Documento (DTD)

DTD

```
<!ELEMENT XXX (AAA* , BBB)>
        <!ELEMENT AAA (#PCDATA)>
        <!ELEMENT BBB (#PCDATA)>
```

L'elemento **XXX** *può* contenere uno o più elementi **AAA** seguiti da un elemento **BBB**. **BBB** deve essere sempre presente

corretto

```
        <XXX> <AAA/> <BBB/> </XXX>
```

AAA non è obbligatorio

```
        <XXX> <BBB/> </XXX>
```

diverse ricorrenze di **AAA** sono ammesse

```
        <XXX> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <BBB/> </XXX>
```

errato

BBB mancante

```
        <XXX> ___ </XXX>
```

BBB deve seguire **AAA**

```
        <XXX> <BBB/> <AAA/> </XXX>
```

AAA non può venire dopo **BBB**:

```
        <XXX><AAA/> <AAA/> <AAA/> <AAA/> <BBB/> <AAA/> <AAA/></XXX>
```

Definizione Tipo Documento (DTD)

DTD

```
<!ELEMENT XXX (AAA+ , BBB)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
```

XXX *deve* contenere uno o più elementi **AAA** seguiti da esattamente un elemento **BBB**. **BBB** deve essere sempre presente

corretto

```
<XXX> <AAA/> <BBB/> </XXX>
<XXX> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <BBB/> </XXX>
```

errato

AAA e **BBB** mancanti

```
<XXX> ___ ___ </XXX>
```

almeno un elemento **AAA** deve essere presente

```
<XXX> ___<BBB/> </XXX>
```

BBB deve essere dopo **AAA**:

```
<XXX> <BBB/> <AAA/> </XXX>
```

AAA non può essere dopo **BBB**:

```
<XXX> <AAA/> <AAA/> <AAA/> <AAA/> <BBB/> <AAA/> <AAA/> </XXX>
```

Definizione Tipo Documento (DTD)

DTD

```
<!ELEMENT XXX (AAA? , BBB)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
```

XXX *può* contenere un elemento **AAA**

corretto

```
<XXX> <AAA/> <BBB/> </XXX>
```

AAA non è obbligatorio

```
<XXX> <BBB/> </XXX>
```

errato

BBB mancante

```
<XXX> ___ </XXX>
```

al massimo un elemento **AAA:**

```
<XXX> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <BBB/> </XXX>
```

BBB deve seguire **AAA**:

```
<XXX> <BBB/> <AAA/> </XXX>
```

Definizione Tipo Documento (DTD)

DTD

```
<!ELEMENT XXX (AAA? , BBB+)>
<!ELEMENT AAA (CCC? , DDD*)>
<!ELEMENT BBB (CCC , DDD)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
```

XXX può contenere un elemento **AAA** seguito da uno o più elementi **BBB**. **AAA** può contenere un **CCC** e diversi **DDD**. **BBB** deve contenere un elemento **CCC** e un elemento **DDD**

corretto

```
<XXX>
   <AAA>
      <CCC/><DDD/>
   </AAA>
   <BBB>
      <CCC/><DDD/>
   </BBB>
</XXX>
```

AAA non è obbligatorio

```
<XXX>
   <AAA/>
   <BBB>
      <CCC/><DDD/>
   </BBB>
</XXX>
```

errato

BBB deve contenere **CCC** e **DDD**:

```
<XXX>
   <AAA/>
   <BBB/>
</XXX>
```

AAA può contenere al massimo un elemento **CCC**:

```
<XXX>
   <AAA>
      <CCC/><CCC/>
      <DDD/><DDD/>
   </AAA>
   <BBB>
      <CCC/><DDD/>
   </BBB>
</XXX>
```

Definizione Tipo Documento (DTD)

DTD

```
<!ELEMENT XXX (AAA , BBB)>
<!ELEMENT AAA (CCC , DDD)>
<!ELEMENT BBB (CCC | DDD)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
```

BBB deve contenere un elemento **CCC** oppure un elemento **DDD**

corretto

```
<XXX>
  <AAA>
    <CCC/> <DDD/>
  </AAA>
  <BBB>
    <CCC/>
  </BBB>
</XXX>
```

```
<XXX>
  <AAA>
    <CCC/> <DDD/>
  </AAA>
  <BBB>
    <DDD/>
  </BBB>
</XXX>
```

errato

Non possono essere presenti entrambi:

```
<XXX>
   <AAA>
      <CCC/> <DDD/>
   </AAA>
   <BBB>
      <CCC/> <DDD/>
   </BBB>
</XXX>

<XXX>
   <AAA>
      <CCC/> <DDD/>
   </AAA>
   <BBB>
      <DDD/> <CCC/>
   </BBB>
</XXX>
```

Definizione Tipo Documento (DTD)

DTD

```
<!ELEMENT attributes (#PCDATA)>
<!ATTLIST attributes
      aaa CDATA #REQUIRED
      bbb CDATA #IMPLIED>
```

Un attributo di tipo CDATA può contenere qualsiasi stringa ben formata.. Un attributo **required** deve essere sempre presente, mentre uno **implied** è opzionale.

corretto

```
<attributes aaa="#d1" bbb="*~*">
   Text
</attributes>
```

L'ordine degli attributi non è importante

```
<attributes bbb="$25" aaa="13%">
   Text
</attributes>
```

l'attributo **bbb** può essere omesso essendo *implied*

```
<attributes aaa="#d1" />
```

errato

L'attributo **aaa** è *required*. Deve essere SEMPRE presente.

```
<attributes ___ bbb="X24"/>
```

Definizione Tipo Documento (DTD)

bbb e **ccc** devono essere sempre presenti, **aaa** è opzionale

DTD

```
<!ELEMENT attributes (#PCDATA)>
<!ATTLIST attributes
        aaa CDATA #IMPLIED
        bbb NMTOKEN #REQUIRED
        ccc NMTOKENS #REQUIRED>
```

Gli attributi **id**, **code** e **X** devono essere unici.

DTD

```
<!ELEMENT XXX (AAA+ , BBB+ , CCC+)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ELEMENT CCC (#PCDATA)>
<!ATTLIST AAA id ID #REQUIRED>
<!ATTLIST BBB
     code ID #IMPLIED
     list NMTOKEN #IMPLIED>
<!ATTLIST CCC
     X ID #REQUIRED
     Y NMTOKEN #IMPLIED>
```

errato

Un attributo ID non può partire con un numero o contenere caratteri non ammessi da NMTOKEN:

```
<XXX>
   <AAA ="L12"/>
   <BBB code="#QW" list="L12"/>
   <CCC X="12" Y="QW" />
</XXX>
```

L'attributo ID deve avere valore UNICO

```
<XXX>
  id   <AAA id="L12"/>
   <BBB code="QW" list="L12"/>
   <CCC X="ZA" Y="QW" />
   <CCC X="ZA" Y="QW" />
</XXX>

<XXX>
   <AAA id="L12"/>
   <BBB code="QW" list="L12"/>
   <CCC X="L12" Y="QW" />
</XXX>
```

CDATA – qualsiasi stringa ben formata

NMTOKEN – lettere, numeri e i caratteri **. - _ :**

NMTOKENS – NMTOKEN e whitespaces (space characters, carriage returns, line feeds, or tabs)

Definizione Tipo Documento (DTD)

DTD

```
<!ELEMENT XXX (AAA+ , BBB+, CCC+, DDD+)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
<!ATTLIST AAA  mark ID      #REQUIRED>
<!ATTLIST BBB  id   ID      #REQUIRED>
<!ATTLIST CCC  ref  IDREF   #REQUIRED>
<!ATTLIST DDD  ref  IDREFS  #REQUIRED>
```

Gli attributi **id** e **mark** determinano unicamente i loro elementi l'attributo **ref** referenzia uno di questi elementi

corretto

```
<XXX>
  <AAA mark="a1"/>
  <AAA mark="a2"/>
  <AAA mark="a3"/>
  <BBB id="b001" />
  <CCC ref="a3" />
  <DDD ref="a1 b001 a2" />
</XXX>
```

errato

Non ci sono attributi di tipo ID con valore a3 o b001:

```
<XXX>
   <AAA mark="a1"/>
   <AAA mark="a2"/>
   <BBB id="b01" />
   <CCC ref="a3" />
   <DDD ref="a1 b001 a2" />
</XXX>
```

Definizione Tipo Documento (DTD)

Possibilità di definire un insieme finito di elementi

```
<!ELEMENT XXX (AAA+, BBB+)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ATTLIST AAA  true ( yes | no ) #REQUIRED>
<!ATTLIST BBB  month (1|2|3|4|5|6|7|8|9|10|11|12) #IMPLIED>
```

Attributi facoltativi con valore di default

```
<!ELEMENT XXX (AAA+, BBB+)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ATTLIST AAA  true ( yes | no ) "yes">
<!ATTLIST BBB  month NMTOKEN "1">
```

L'elemento **AAA** può contenere attributi ma non un valore.

```
<!ELEMENT XXX (AAA+)>
<!ELEMENT AAA EMPTY>
<!ATTLIST AAA true ( yes | no ) "yes">
```

Definizione Tipo Documento (DTD)

Le entità vanno definite nel DTD

```
<!ENTITY EntityName EntityDefinition>
```

entità interna

```
<!ENTITY SIGNATURE "Alessio">
```

&SIGNATURE;

entità esterna

```
<!ENTITY IMAGE1 SYSTEM "Xmlquot.gif" NDATA GIF>
```

Definizione Tipo Documento (DTD)

Le entità vanno definite nel DTD

```
<!ENTITY EntityName EntityDefinition>
```

entità interna

```
<!ENTITY SIGNATURE "Alessio">
```

&SIGNATURE;

entità esterna

```
<!ENTITY IMAGE1 SYSTEM "Xmlquot.gif" NDATA GIF>
```

annotazioni (NOTATION) per entità non analizzabili (NDATA)

```
<!ENTITY IMAGE1 SYSTEM "Xmlquot.gif" NDATA GIF>
```

Un riferimento a tale enitià produrrebbe un errore:

Declaration 'Xmlquot' contains reference to undefined notation 'GIF'.

```
<!NOTATION GIF SYSTEM "Iexplore.exe">
```

Definizione Tipo Documento (DTD)

Entità di parametro

```
<!ENTITY % ENCRYPTION "40bit CDATA #IMPLIED 128bit CDATA #IMPLIED">
```

```
<!ELEMENT EMAIL (TO+, FROM, CC*, BCC*, SUBJECT?, BODY?)>
<!ATTLIST EMAIL LANGUAGE(Western|Greek|Latin|Universal) "Western"
ENCRYPTED %ENCRYPTION;
PRIORITY (NORMAL|LOW|HIGH) "NORMAL">
```

Definizione Tipo Documento (DTD)

namespace

spazio dei nomi: raccolta di nomi identificata da un URI.

```
<?xml version="1.0"?>
<?xml:namespace ns=http://inventory/schema/ns prefix="inv"?>
<?xml:namespace ns=http://wildflowers/schema/ns prefix="wf"?>
<PRODUCT>
  <PNAME>Test1</PNAME>
  <inv:quantity>1</inv:quantity>
  <wf:price>323</wf:price>
  <DATE>6/1</DATE>
</PRODUCT>
```

nomi qualificati = prefix : localPart

```
<CATALOG>
<INDEX>
<ITEM>Trees</ITEM>
</INDEX>
<PRODUCT xmlns:wf="urn:shemas-wildflowers-
com”>          <NAME>Bloodroot</NAME>
<QUANTITY>10</QUANTITY>
</PRODUCT>
</CATALOG>
```

Namespace predefinito

Definizione Tipo Documento (DTD)

DICHIARAZIONE DELLO SPAZIO DEI NOMI COME URI

```
<wf:product xmlns:wf="urn:shemas-wildflowers-com”>
<wf:name>Bloodroot</wf:name>
<QUANTITY>10</QUANTITY>
<PRICE>$2.44</PRICE>
</wf:product>
```

L’URI che definisce il namespace è puramente formale: non c’è nessuna garanzia che il documento all’URI specificato contenga la descrizione della sintassi utilizzata o che esista effettivamente un documento.
Per un processore XML un documento che utilizzi namespace o no non fa differenza e deve essere sempre well-formed.

NON E’ POSSIBILE APPLICARE LO STESSO DTD SE QUALIFICHIAMO I NOMI

`<!ELEMENT DIVISION (DIVISION_NAME, TEAM+)>` → `<!ELEMENT bb:DIVISION (bb:DIVISION_NAME, bb:TEAM+)>`

XML schema

Limitazione dei DTD

i DTD nascono con lo scopo di trattare una tipologia ben specifica di documenti come libri, brochures, manuali, pagine web. E' facile infatti imporre regole che definiscano ad esempio che in un libro esistano uno o più autori, oppure che una canzone abbia un solo titolo. I DTD dimostrano la loro inefficienza in tutte quelle applicazioni di XML che si differenziano da questi ambienti (scambio di dati computer-to-computer).

I DTD non dispongono di Tipi di dato

I DTD non permettono di definire come debba essere composto il contenuto di un elemento

I DTD non supportano namespace

I DTD non hanno una sintassi XML

Non è possibile definire il numero di figli di un nodo senza imporne l'ordine

XML schema

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

<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name="GREETINGS" type="xsd:string"/>
</xsd:schema>
```

L'elemento di root di ogni schema è rappresentato da *schema* che deve essere all'interno del namespace specificato. Il prefisso xsd può essere cambiato se rimane lo stesso URI.

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

<GREETINGS
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance
  xsi:noNamespaceSchemaLocation="C:\unibo\greetings.xsd">HELLO! </GREETINGS>
```

XML VALIDO

```
<GREETNGIS> various random text but no markup</GREETINGS>

<GREETINGS>Hello!</GREETINGS>

<GREETINGS></GREETINGS>
```

XML NON VALIDO

```
<GREETINGS> <SOME_TAG>various random text</SOME_TAG>
<SOME_EMPTY_TAG/> </GREETINGS>

<GREETINGS> <GREETINGS>various random text</GREETINGS> </GREETINGS>
```

XML schema

Complex Types

A differenza dei tipi semplici, permettono di definire tipi di elementi con attributi e elementi figli.

Root element

Solo un titolo

Diversi produttori o nessuno

Solo un anno

Potrebbe esserci o no.

```
<?xml version="1.0"?>
<SONG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="song.xsd">
  <TITLE>Yes I Am</TITLE>
  <COMPOSER>Melissa Etheridge</COMPOSER>
  <PRODUCER>Hugh Padgham</PRODUCER>
  <PUBLISHER>Island Records</PUBLISHER>
  <LENGTH>4:24</LENGTH>
  <YEAR>1993</YEAR>
  <ARTIST>Melissa Etheridge</ARTIST>
  <PRICE>$1.25</PRICE>
</SONG>
```

namespace

Potrei avere diversi compositori

Uno o nessuno

Almeno uno.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="SongType">
  <xsd:sequence>
    <xsd:element name="TITLE" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="COMPOSER" type="xsd:string" minOccurs="1"
maxOccurs="unbounded"/>
    <xsd:element name="PRODUCER" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="PUBLISHER" type="xsd:string" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="LENGTH" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="YEAR" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="ARTIST" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
    <xsd:element name="PRICE" type="xsd:string" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

XML schema

Complex Types

Aumentiamo la profondità della struttura

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="ComposerType">
    <xsd:sequence>
      <xsd:element name="NAME" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ProducerType">
    <xsd:sequence>
      <xsd:element name="NAME" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      <xsd:element name="TITLE"     type="xsd:string"/>
      <xsd:element name="COMPOSER"  type="ComposerType"  maxOccurs="unbounded"/>
      <xsd:element name="PRODUCER"  type="ProducerType" minOccurs="0" MaxOccurs="unbounded"/>
      <xsd:element name="PUBLISHER" type="xsd:string" minOccurs="0"/>
      <xsd:element name="LENGTH" type="xsd:string"/>
      <xsd:element name="YEAR"   type="xsd:string"/>
      <xsd:element name="ARTIST" type="xsd:string"  maxOccurs="unbounded"/>
      <xsd:element name="PRICE" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```
<?xml version="1.0"?>
<SONG xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="24-10.xsd">
  <TITLE>Hot Cop</TITLE>
  <COMPOSER><NAME>Jacques Morali</NAME></COMPOSER>
  <COMPOSER><NAME>Henri Belolo</NAME></COMPOSER>
  <COMPOSER><NAME>Victor Willis</NAME></COMPOSER>
  <PRODUCER><NAME>Jacques Morali</NAME></PRODUCER>
  <PUBLISHER>PolyGram Records</PUBLISHER>
  <LENGTH>6:20</LENGTH>
  <YEAR>1978</YEAR>
  <ARTIST>Village People</ARTIST>
</SONG>
```

XML schema

Nell'esempio precedente i due tipi definiti sono identici. E' possibile creare un solo tipo e condividerlo.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="PersonType">
    <xsd:sequence>
      <xsd:element name="NAME" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      <xsd:element name="TITLE"     type="xsd:string"/>
      <xsd:element name="COMPOSER"  type="PersonType"  maxOccurs="unbounded"/>
      <xsd:element name="PRODUCER"  type="PersonType"  minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="PUBLISHER" type="xsd:string" minOccurs="0"/>
      <xsd:element name="LENGTH" type="xsd:string"/>
      <xsd:element name="YEAR"   type="xsd:string"/>
      <xsd:element name="ARTIST" type="xsd:string" maxOccurs="unbounded"/>
      <xsd:element name="PRICE" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

XML schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="SONG" type="SongType"/>

 <xsd:complexType name="NameType">
  <xsd:sequence>
     <xsd:element name="GIVEN"  type="xsd:string"/>
      <xsd:element name="FAMILY" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="PersonType">
    <xsd:sequence>
      <xsd:element name="NAME" type="NameType"/>
    </xsd:sequence>
  </xsd:complexType>


  <xsd:complexType name="SongType">
    <xsd:sequence>
      …
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

In questo modo posso utilizzare lo stesso elemento in posizioni diverse con tipi diversi. Per esempio è possibile che NAME di PERSON contenga i figli GIVEN e FAMILY, mentre NAME di MOVIE contenga una stringa.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="PersonType">
    <xsd:sequence>
      <xsd:element name="NAME">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="GIVEN"  type="xsd:string"/>
            <xsd:element name="FAMILY" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      ...
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

XML schema

XML Schema mette a disposizione tre construtti che permettono di specificare come e se l'ordine degli elementi è importante

```
<xsd:complexType name="PersonType">
  <xsd:sequence>
    <xsd:element name="NAME">
      <xsd:complexType>
        <xsd:all>
          <xsd:element name="GIVEN" type="xsd:string"
                       minOccurs="1" maxOccurs="1"/>
          <xsd:element name="FAMILY" type="xsd:string"
                       minOccurs="1" maxOccurs="1"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

xsd:all
Specifica che tutti gli elementi devono essere presenti in qualunque ordine.

```
<xsd:complexType name="SongType">
  <xsd:sequence>
    <xsd:element name="TITLE" type="xsd:string"/>
    <xsd:choice>
      <xsd:element name="COMPOSER" type="PersonType"/>
      <xsd:element name="PRODUCER" type="PersonType"/>
    </xsd:choice>
    <xsd:element name="PUBLISHER" type="xsd:string"
                 minOccurs="0"/>
    <xsd:element name="LENGTH" type="xsd:string"/>
    <xsd:element name="YEAR"   type="xsd:string"/>
    <xsd:element name="ARTIST" type="xsd:string"
                 maxOccurs="unbounded"/>
    <xsd:element name="PRICE" type="xsd:string"
minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

xsd:choice
Corrisponde ad un OR (esclusivo) logico. Nell'esempio significa che devono essere presenti o l'elemento COMPOSER o PRODUCER ma non contemporaneamente.

xsd:sequence
indica l'ordine nel quale devono comparire gli elementi. Il numero di volte che questi si possono ripetere viene controllato con gli attributi minOccours e maxOccours.

XML schema

Name:	Type:	Examples:
xsd:float	IEEE 754 32-bit floating point number, or as close as you can get using a base 10 representation; same as Java's float type	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
xsd:double	IEEE 754 64-bit floating point number, or as close as you can get using a base 10 representation; same as Java's double type	-INF, 1.401E-90, -1E4, -0, 0, 12.78E-2, 12, INF, NaN, 3.4E42
xsd:decimal	Arbitrary precision, decimal numbers; same as java.math.BigDecimal	-2.7E400, 5.7E-444, -3.1415292, 0, 7.8, 90200.76, 3.4E1024
xsd:integer	An arbitrarily large or small integer; same as java.math.BigInteger	-5000000000000000000000000, -9223372036854775809, -126789, -1, 0, 1, 5, 23, 42, 126789, 9223372036854775808, 45673498732498326498736249 58
xsd:nonPositiveInteger	An integer less than or equal to zero	0, -1, -2, -3, -4, -5, -6, -7, -8, -9, . . .
xsd:negativeInteger	An integer strictly less than zero	-1, -2, -3, -4, -5, -6, -7, -8, -9, . . .
xsd:long	An eight-byte two's complement integer such as Java's long type	-9223372036854775808, -9223372036854775807, . . . -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . ., 2147483645, 2147483646, 2147483647, 2147483648, . . .9223372036854775806, 9223372036854775807
xsd:int	An integer that can be represented as a four-byte, two's complement number such as Java's int type	-2147483648, -2147483647, -2147483646, 2147483645, . . . -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . ., 2147483645, 2147483646, 2147483647
xsd:short	An integer that can be represented as a two-byte, two's complement number such as Java's short type	-32768, -32767, -32766, . . ., -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, . . . 32765, 32766, 32767
xsd:byte	An integer that can be represented as a one-byte, two's complement number such as Java's byte type	-128, -127, -126, -125, . . ., -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, . . .121, 122, 123, 124, 125, 126, 127
xsd:nonNegativeInteger	An integer greater than or equal to zero	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ...
xsd:unsignedLong	An eight-byte unsigned integer	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, . . .18446744073709551614, 18446744073709551615
xsd:unsignedInt	A four-byte unsigned integer	0, 1, 2, 3, 4, 5, . . .4294967294, 4294967295
xsd:unsignedShort	A two-byte unsigned integer	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . .65533, 65534, 65535
xsd:unsignedByte	A one-byte unsigned integer	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . . 252, 253, 254, 255
xsd:positiveInteger	An integer strictly greater than zero	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . .

Simple Types: Numeric data types

XML schema

Simple Types

XML data types

Name:	Type:	Examples:
xsd:ID	XML 1.0 ID attribute type; any XML name that's unique among ID type attributes and elements	p1, p2, ss124-45-6789, _92, red, green, NT-Decl, seventeen
xsd:IDREF	XML 1.0 IDREF attribute type; any XML name that's used as the value of an ID type attribute or element elsewhere in the document	p1, p2, ss124-45-6789, _92, p1, p2, red, green, NT-Decl, seventeen
xsd:ENTITY	XML 1.0 ENTITY attribute type; any XML name that's declared as an unparsed entity in the DTD	PIC1, PIC2, PIC3, cow_movie, MonaLisa, Warhol
xsd:NOTATION	XML 1.0 NOTATION attribute type; any XML name that's declared as a notation name in the schema using xsd:notation	GIF, jpeg, TIF, pdf, TeX
xsd:IDREFS	XML 1.0 IDREFS attribute type; a white space-separated list of XML names that are used as values of ID type attributes or elements elsewhere in the document	p1 p2, ss124-45-6789 _92, red green NT-Decl seventeen
xsd:ENTITIES	XML 1.0 ENTITIES attribute type; a white space-separated list of ENTITY names	PIC1 PIC2 PIC3
xsd:NMTOKEN	XML 1.0 NMTOKEN attribute type	12 are you ready 199
xsd:NMTOKENS	XML 1.0 NMTOKENS attribute type, a white space-separated list of name tokens	MI NY LA CA p1 p2 p3 p4 p5 p6 1 2 3 4 5 6
xsd:language	Valid values for xml:lang as defined in XML 1.0	en, en-GB, en-US, fr, i-lux, ama, ara, ara-EG, x-choctaw
xsd:Name	An XML 1.0 Name, with or without colons	set, title, rdf, math, math123, xlink:href, song:title
xsd:QName	a prefixed name	song:title, math:set, xsd:element
xsd:NCName	a local name without any colons	set, title, rdf, math, tei.2, href

XML schema

Simple Types

String data types

Name:	Type:	Examples:
xsd:string	A sequence of zero or more Unicode characters that are allowed in an XML document; essentially the only forbidden characters are most of the C0 controls, surrogates, and the byte-order mark	p1, p2, 123 45 6789, ^*&^*&_92, red green blue, NT-Decl, seventeen; Mary had a little lamb, The love of money is the root of all Evil., Would you paint the lily? Would you gild gold?
xsd:normalizedString	A string that does not contain any tabs, carriage returns, or linefeeds	PIC1, PIC2, PIC3, cow_movie, MonaLisa, Hello World , Warhol, red green
xsd:token	A string with no leading or trailing white space, no tabs, no linefeeds, and not more than one consecutive space	p1 p2, ss123 45 6789, _92, red, green, NT Decl, seventeenp1, p2, 123 45 6789, ^*&^*&_92, red green blue, NT-Decl, seventeen; Mary had a little lamb, The love of money is the root of all Evil.

Derivazione di tipi semplici

A partire dai tipi semplici visti possono essere derivati altri tipi di dato.

- **xsd:restriction** per selezionare un subset dei valori ammessi dal tipo base
- **xsd:union** combinazione di tipi
- **xsd:list** lista di elementi di un tipo semplice di base

esempio

```
<xsd:simpleType name="phonoYear">
  <xsd:restriction base="xsd:gYear">
    <xsd:minInclusive value="1877"/>
    <xsd:maxInclusive value="2100"/>
  </xsd:restriction>
</xsd:simpleType>
```

FACETS

xsd:minExclusive: valore minimo del quale devono essere strettamente maggiori (>).

xsd:minInclusive: valore minimo del quale devono essere maggiori o uguali (>=).

xsd:maxInclusive: valore massimo del quale devono essere minori o uguali (<=).

xsd:maxExclusive: valore massimo del quale devono essere strettamente minori (<).

xsd:enumeration: lista di valori

xsd:whiteSpace: come vengono trattati i whitespace

xsd:pattern: expressione di confronto

xsd:length: num di caratteri ammessi

xsd:minLength: numero min di caratteri

xsd:maxLength: max num di caratteri

xsd:totalDigits: max numero di cifre

xsd:fractionDigits: max numero di decimali

XML schema

xsd:union permette di derivare un tipo semplice dall'unione di di altri tipi semplici: nell'esempio il tipo MoneyOrDecimal può contenere o un numerico decimale o un tipo "valuta" definito tramite un pattern.

```
<xsd:simpleType name="MoneyOrDecimal">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="\p{Sc}\p{Nd}+(\.\p{Nd}\p{Nd})?"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

xsd:list permette di derivare un tipo semplice come lista di valori. Nell'esempio il tipo YearList può contenere una lista di anni tipo:<YEAR>1992 1997 1980 1971</YEAR>.

E' possibile restringere il numero degli elementi ammessi nella lista.

```
<xsd:simpleType name="YearList">
  <xsd:list itemType="xsd:gYear"/>
</xsd:simpleType>

<xsd:simpleType name="DoubleYear">
  <xsd:restriction base="YearList">
    <xsd:length value="2"/>
  </xsd:restriction>
</xsd:simpleType>
```

XML schema

Anche con gli *schema* è possibile definire gli attributi. Rispetto ai DTD possiamo utilizzare anche per essi i vari data types.

```
<xsd:complexType name="PhotoType">
   <xsd:attribute name="SRC"    type="xsd:anyURI"/>
   <xsd:attribute name="WIDTH"  type="xsd:positiveInteger"/>
   <xsd:attribute name="HEIGHT" type="xsd:positiveInteger"/>
   <xsd:attribute name="ALT"    type="xsd:string"/>
 </xsd:complexType>
```

```
<TITLE ID="test">Yes I Am</TITLE>
```

```
<xsd:complexType name="StringWithID">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="ID" type="xsd:ID“ use=“required" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

```
<xsd:element name="TITLE" type="StringWithID"/>
```