XML: incipit

- XML document --- documento per la **rappresentazione di strutture di dati**, scritto usando un (beh, uno o piu' ...) **linguaggio di markup apposito per quei dati**
  - E chi ha definito quel linguaggio di markup?
  - Lo definisce chi progetta i dati, definendo una *DTD* o uno *Schema*
- Scrivere documenti XML ben formati ("**well formed**") e' il primo passo nella giusta direzione
- Se il **documento** rispetta la definizione del linguaggio e' "**valido**"
- Un **documento XML** puo' essere **processato** (mediante un'**applicazione XML**)per una varieta' di scopi: uno scopo puo' essere la sua presentazione (l'applicazione in tal caso sarebbe come il browser)
- Come e' fatta una DTD?

Slide discorsiva, ok? Molte cose vanno spiegate …

XML – eXtensible Markup Language

In un file `.xml` vengono usati elementi "inventati" (**definiti**) dall'autore. Tali elementi sono definiti specificandone i tag da cui sono delimitati, nonche' gli attributi ed i sottoelementi possibili … La definizione di questi elementi costituisce l'insieme delle regole da rispettare per usarli nel documento .xml

Il **(contenuto del) documento** `.xml` non fa altro che **rappresentare** un certo insieme di dati secondo la **struttura** decisa nella suddetta definizione degli elementi.

Quei **dati**, in base alla posizione in cui si trovano in tale struttura, assumono un **significato** (abbastanza) preciso: sono "**informazioni**" a tutti gli effetti.

Niente viene specificato, nel file `.xml` , a proposito della **presentazione** dei dati (aspetto che viene curato in separata sede: **nel file `.xml` ci si occupa solo di struttura e significato**)

NB in questa pagina ci sono due esempi di fie xml ...

In un file XHTML: dati messi in elementi predefiniti.
Interpretazione dei dati: presentazione (ev. di default, se non ci sono specifiche CSS).

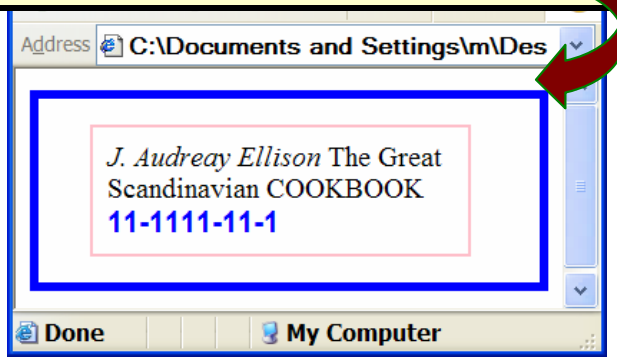
```
<div style="border: solid thick  blue;
padding: 5%; color: black; font-size: 12pt;">
 <div style= "border: solid thin pink;
margin-l ...
  <span style="font-style: oblique;">J.
Audreay Ellison</span>
  <span style="font-family:Times New Roman,
serif;">
   The Great Scandinavian COOKBOOK</span>
  <span style="font-family:Arial; color:blue;
font-size:smaller; font-weight:bold">
   11-1111-11-1</span></div></div>
```

In un file XML: informazioni su **strutture dati**, date in **elementi** di un apposito linguaggio, il cui scopo e' rappresentare quel tipo di strutture dati, per consentire un uso successivo, automatizzato, attraverso una applicazione XML, di quei dati (non necessariamente, ma non esclusa a priori, la presentazione).

```
<libri>
 <book>
  <author>J. A. Ellison
  </author>
  <title>The Great COOKBOOK
  </title>
  <isbn>11-1111-11-1</isbn>
 </book> </libri>
```

STRUTTURA - ogni libro e' specificato da **info su autore** (posta in un elemento author), **titolo** (…title) e **isbn** (…isbn)

Process



XML Vs. SGML

SGML e' un linguaggio per la definizione di *linguaggi-di-markup*.

Complicato e macchinoso, perche' ricco e molto espressivo (pensato per documenti complessi).

Dato un linguaggio definito con SGML, un'***applicazione*** per quel linguaggio lavora su un file scritto in quel linguaggio, per *trattare i dati in esso contenuti* (presentarli, cercarli e renderli disponibili per calcoli, etc …).

Essendo SGML "grosso", le applicazioni possono essere tanto complicate.

XML = sottoinsieme di SGML. XML e' una semplificazione di SGML, ma e' ancora molto ricco e permette di definire linguaggi di markup. Ad esempio … XHTML.

HTML, fu definito con SGML**;** XHTML e' stato definito mediante XML. Un *browser* e' *un'applicazione* per la gestione (tra l'altro) di file XHTML. Il browser si occupa di scorrere gli elementi contenuti nel file html e *presentare i dati contenuti in quegli elementi* (in base alle regole con cui gli elementi e i loro attributi sono associati ad una visualizzazione).

SEPARAZIONE tra struttura/semantica e presentazione del contenuto: ha **vantaggi**:

1) la stessa sorgente dati puo' essere **presentata in modi diversi da diverse applicazioni** (ad esempio dipendentemente dai media coinvolti) – o gestita per finalita' diverse da diverse applicazioni;
2) la **ricerca di informazioni** viene enormemente aiutata (es: selezione libri di kissinger da un file html di libri - e ci sono libri *di* kissinger e *su* kissinger);
3) **scambio dati tra applicazioni** che lavorano in ambienti diversi, s.o. diversi, software diversi, formati dati ev. diversi, il tutto reso compatibile tramite scambio di file XML (b2b!) localmente processati dalle opportune applicazioni xml

Analisi di un file XML:

e' un file testuale contenente la definizione di alcuni dati, basata su un linguaggio di specifica dei dati

Il file contiene un *elemento radice* `<libri>`: questa e' la categoria dei dati specificati nel documento

(... un file testuale contenente la definizione di alcuni dati...)

```
<?xml version="1.0" encoding="UTF-8"?>
<libri>
  <book>
    <author>J. Audreay Ellison</author>
    <title>The Great Scandinavian COOKBOOK</title>
    <isbn>0</isbn>
  </book>
  <book>
    <author> Laura Santos </author>
    <title>O Mestre Cozinheiro</title>
    <isbn>0-111111-4-11</isbn>
  </book>
  <book>
    <author>Anna Macmiadhachain, Mary
        Raynolds    </author>
    <title> The Mediterranean Cookbook  </title>
    <isbn>0-07-093536 X</isbn>
  </book>
</libri>
```

(... linguaggio di specifica dei dati ...)

I *libri* sono specificati come una sequenza di elementi `<book>`

Ogni *book* e' specificato come un insieme di dati suddivisi in tre gruppi:

- **i dati sull'*autore*, specificati nell'elemento `<author>` del *book*,**
- **i dati sul titolo del book, specificati nell'elemento `<title>` del *book*,**
- **i dati sul numero isbn, specificati nell'elemento `<isbn>` del *book*.**

In particolare,

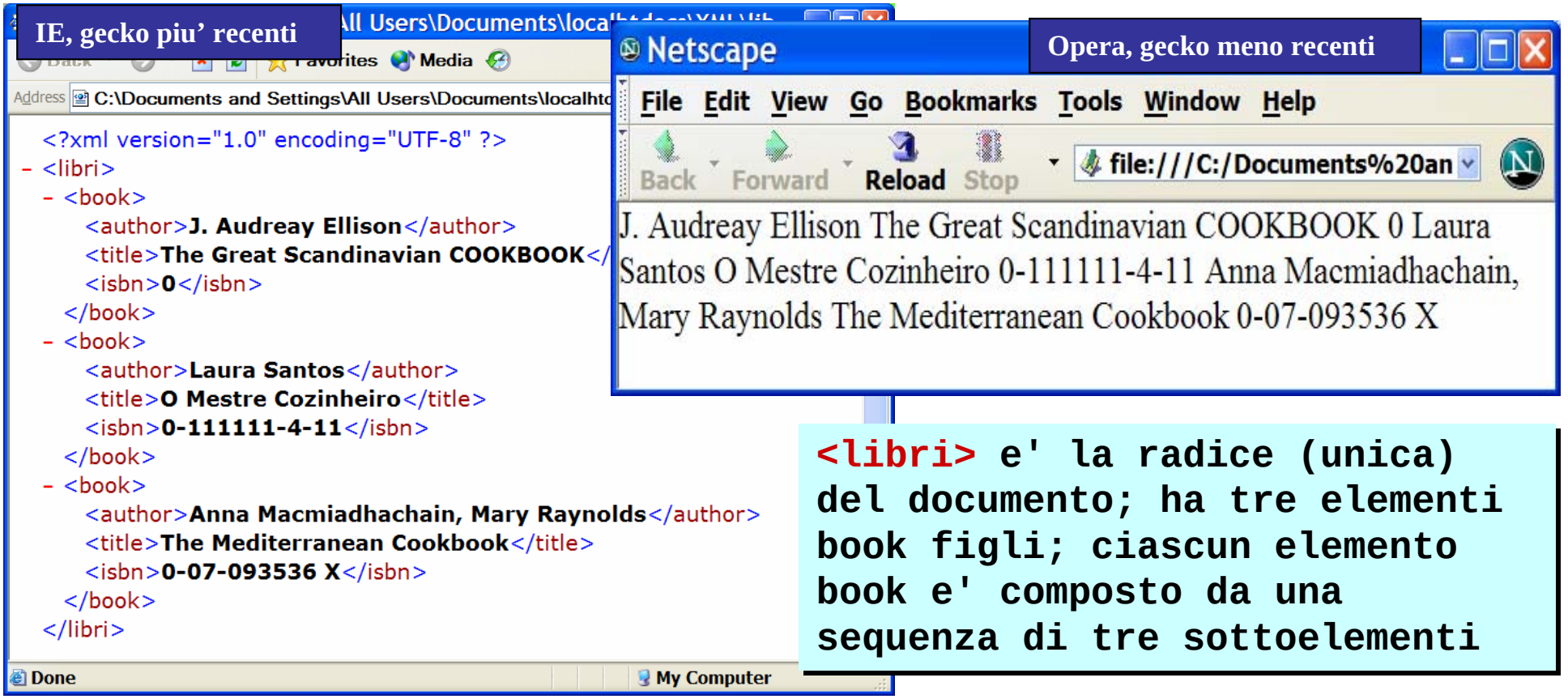
- ❑ **l'elemento `<libri>` e' composto da una sequenza di book;**
- ❑ **l'elemento `<book>` e' composto da una sequenza di: un el. `<author>` , un el. `<title>` e un el. `<isbn>`**
- ❑ **l'elem. `<author>` contiene caratteri**
- ❑ **l'elem. `<title>` contiene caratteri**
- ❑ **l'elemento `<isbn>` contiene caratteri**

XML – presentazione

Vedi libri.0.xml

Dato che gli elementi di un documento xml sono inventati dall'autore, un browser, o un'applicazione di presentazione, deve avere istruzioni su come visualizzare il contenuto.
Come dare queste istruzioni?

A) **non le diamo**: allora il browser decide se e come visualizzare i nostri tag



B) specifichiamo una **style sheet** (usando css, o xsl) - tra poco

XML technology

Nucleo fondamentale
sintassi, DTD,
Schema, Namespace

Specifiche di completamento
XPath, XSL-XSLT, **DOM,...**

Software di supporto

SAX / DOM,
Expat / Xerces / JAXP / MSXML,
ambienti di sviluppo (come XMLSpy) ...

Linguaggi XML su cui si basano Applicazioni XML

XHTML, MathML (vedi con Amaya), WML (mobile), VoiceXML, SMIL/SVG (multimedia), SOAP/XML-RPC(computer2computer comm.), ebXML (business2business comm.), linguaggi per la rappresentazione di informazioni in varie discipline (legge, chimica, biologia), ...

```
<SOAP-ENV:Body>
<proc:CompAvgValue
xmlns:proc="path.to/the/resource" />
```

```
<libri>
 <book>
  <author>J. Audreay Ellison</author>
  <title>The Great... </title>
<isbn>11-1111-1-1</isbn> </book></libri>
```

```
<dsc type="combined"><head>Inventory</head>
   <c01>    <did> <unittitle>Correspondence</unittitle> </did>
      <c02> <did>   <unittitle>Adams, Martha</unittitle>
                    <unitdate normal="1962/1967">1962-1967</unitdate>
                    <container type="box">1</container>
                    <container type="folder">1</container>
            </did>
      </c02>
…
```

```
<molecule convention="Mol" id="dopamine"
                  title="DOP">
 <date day="22" mon="1" year="1995" />
 <atomArray>
  <atom id="1">
   <string builtin="eltype">C</string>
   <float builtin="x">0.021</float>
  </atom>
```

XML – documenti well formed (1/4)

Vedi libri.0.sb.xml

Un documento che rispetta l'insieme di regole sintattiche della specifica XML (1.0 e succ.) e' *ben formato*.

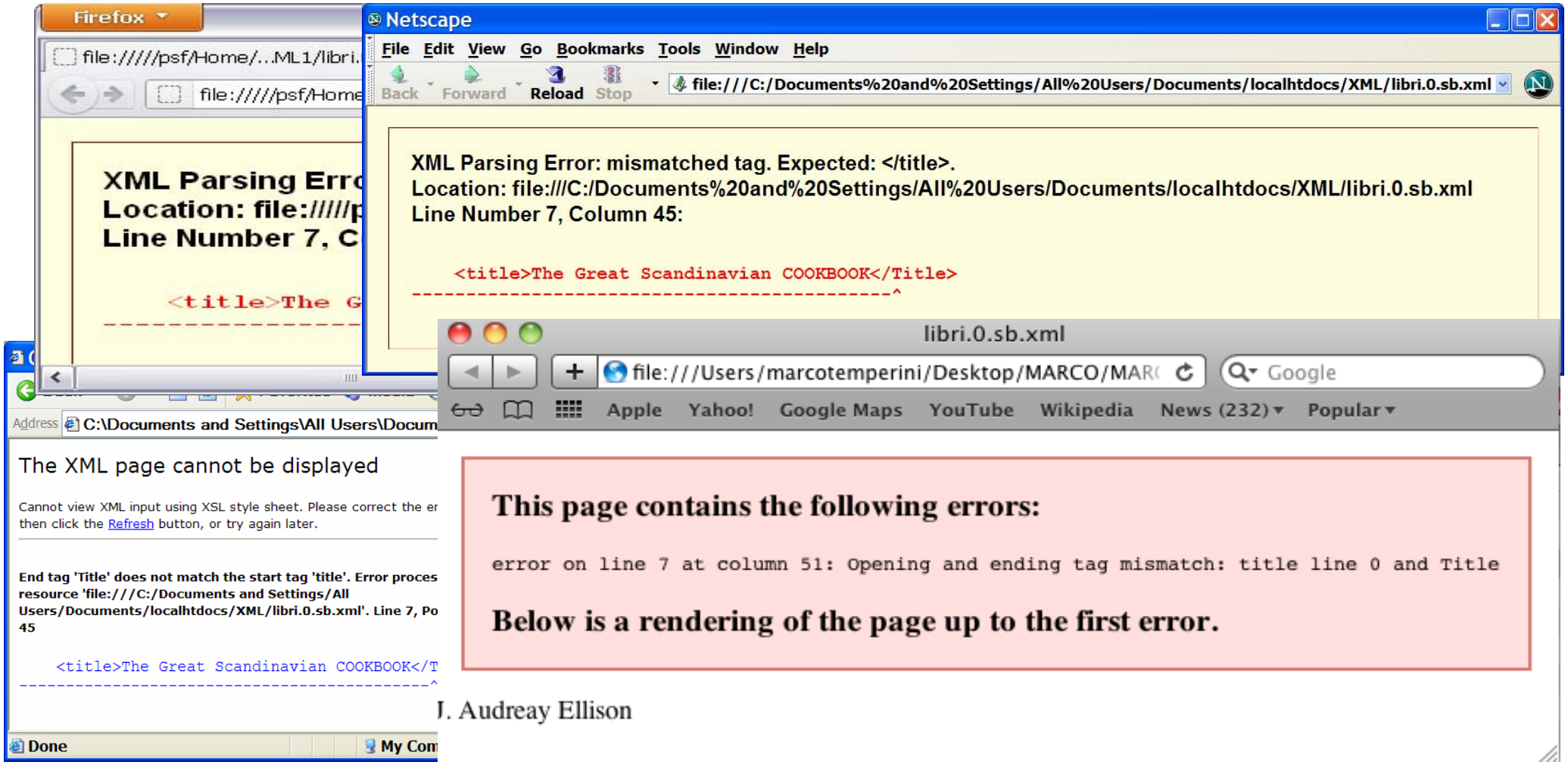
1) una sola radice

2) tag case sensitive

3) ogni elemento deve essere chiuso; es. chiusura el. vuoti: `<hr />` o `<hr></hr>`

4) nidificazione elementi corretta (`<NO><strong> <em> </strong> </em></NO>`)

5) attributi solo nello start tag; del tipo **`nome="valore"`** e appaiono al piu' una volta nel tag;



XML – documenti well formed (2/4)

Vedi libri.0.sb.xml

1) una sola radice 2) tag case sensitive 3) ogni elemento deve essere chiuso 4) nidificazione elementi corretta 5) attributi solo nello start tag; …

```
<?xml version="1.0" encoding="UTF-8"?>
<libri>
  <book>
    <author>J. Audreay Ellison</author>
    <title>The Great Scandinavian COOKBOOK</Title>
    <isbn>0</isbn>
  </book>
  <book>
    <AUTHOR> Laura Santos </AUTHOR>
    <title>O Mestre Cozinheiro</title>
    <isbn>0-111111-4-11</isbn>
  </book>
  <book>
    <author> A. Macmiadhachain, M. Raynolds</author>
    <title>The Mediterranean Cookbook</title>
    <isbn>0-07-093536 X</isbn>
  </book>
</libri>
```

Cosa non va?

XML – documenti well formed (2/4)

Vedi libri.0.sb.xml

1) una sola radice 2) tag case sensitive 3) ogni elemento deve essere chiuso 4) nidificazione elementi corretta 5) attributi solo nello start tag; …

```
<?xml version="1.0" encoding="UTF-8"?>
<libri>
  <book>
    <author>J. Audreay Ellison</author>
    <title>The Great Scandinavian COOKBOOK</Title>
    <isbn>0</isbn>
  </book>
  <book>
    <AUTHOR> Laura Santos </AUTHOR>
    <title>O Mestre Cozinheiro</title>
    <isbn>0-111111-4-11</isbn>
  </book>
  <book>
    <author> A. Macmiadhachain, M. Raynolds</author>
    <title>The Mediterranean Cookbook</title>
    <isbn>0-07-093536 X</isbn>
  </book>
</libri>
```

tsk

ok (se si parla di w.form.)

Ma qui un po' stiamo barando: in effetti l'uso di caratteri maiuscoli, in un contesto che quasi sempre ha sottolineato il fatto che si usano solo minuscoli per i tag, e' fuorviante

XML – documento well formed (3/4)

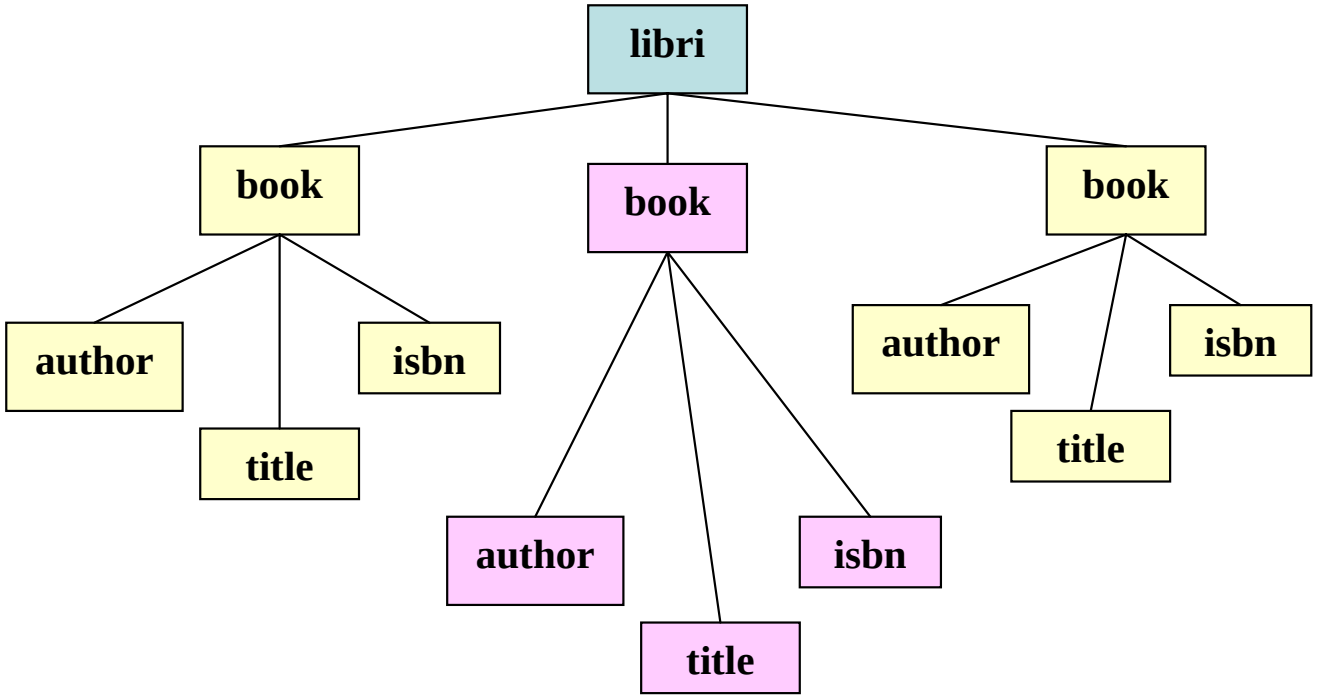
Il documento XML deve essere well formed, senno' il parser e l'applicazione XML non potrebbero nemmeno *iniziare* a processarlo (…nemmeno si riesce a visitarlo, figurarsi se si puo' pretendere di cercare informazioni, aggiornarle, etc … !).

Un documento well formed ha una struttura abbastanza coerente da poter essere resa come "albero del documento" (che e' poi un ottimo strumento per l'elaborazione delle informazioni contenute nel documento).

```
<?xml version="1.0"?>
<libri>
  <book>
    <author>...</author>
    <title>...</title>
    <isbn>...</isbn>
  </book>

  <book>
    ...
  </book>

  <book>
    ...
  </book>
</libri>
```



XML – documento well formed (4/4)

Il documento XML deve essere well formed, senno' il parser e l'applicazione XML non potrebbero nemmeno *iniziare* a processarlo (…nemmeno si riesce a visitarlo, altro che cercare informazioni, aggiornarle, etc … !).

Un documento well formed ha una struttura abbastanza coerente da poter essere resa come "albero del documento" … magari sbilenco ma albero …

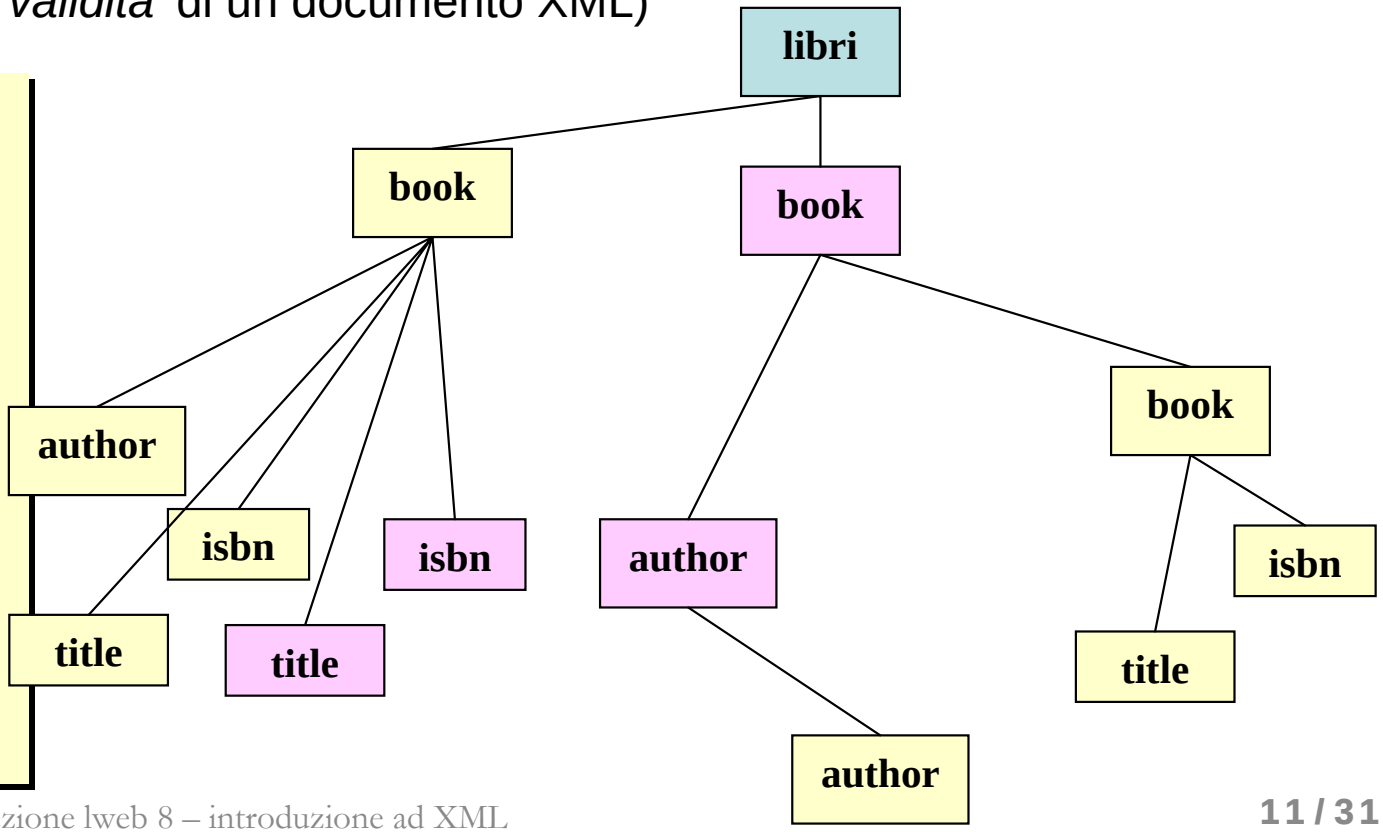
PERO'

Il documento seguente e' well formed ma sembra avere qualcosa che non va ... in effetti la sua struttura sembra un po' erratica, contraddittoria: un libro contiene un altro libro; un altro contiene due titoli ... un'applicazione (cioe' un programma) che dovesse gestire questo documento avrebbe un compito ingrato

in altre parole il documento non sembra essere basato su una buona organizzazione / strutturazione dei dati (e quindi non e' molto appetibile per un'applicazione xml che dovesse usarlo - vedi dopo il concetto di *validita*' di un documento XML)

```
<?xml version="1.0"?>
<libri>
  <book>
    <author>...</author>
    <title>...</title
    <isbn>...</isbn>
    <title>...</title>
    <isbn>...</isbn>
  </book>

  <book>
    ...
  </book>
</libri>
```



XML – altro (1/2)

- **dichiarazione XML**: obbligatoria;
  - una linea in testa ad ogni documento,
  - encoding,
  - optional standalone document declaration.
  - (e' una *"processing instruction" – vedi dopo ? "no, non lo e'!", "si'", "no"…*)
    `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`

- **commenti** come XHTML

- **nomi tag**: case sensitive / iniziano con lettera/underscore (1.0) o anche altri caratteri Unicode consentiti anche all'interno (1.1 – `w3…/TR/xml11`) / non iniziano per cifra / non contengono spazi / lunghezza a piacere / non iniziamoli con "`xml`" / non usiamo ':' ("riservato") se non stiamo definendo namespaces

- **nomi attributi**: restrizioni come per tag; alcuni riservati `(xml:lang="en/it/…"`

- **entita'**: simboli che permettono di includere caratteri speciali o confondibili in un documento (`< > &apos; &quote; &` sono quelli standard, corrispondenti ai caratteri speciali intuibili)

- **riferimenti ad altri caratteri**: per includere caratteri non standard nel documento;
  - formato: **`&#NNN;`** oppure **`&#xXXX;`** dove **`NNN`** e' il codice decimale (**`XXX`** quello esadecimale) del carattere nella tabella Unicode

XML – altro (2/2)

- **processing instructions** (direttive per l'applicazione XML, cioe' il programma in azione sul documento xml): passate direttamente all'applicazione dal parser; possono apparire dovunque

```
<?target      istr/param ?>
```
(il programma) (specifiche o dati utili)

Esempio

```
<?xml-stylesheet href="libri.css" type="text/css"?>
```

- **sezioni CDATA** (character data): sezioni non interpretate dal parser xml

```
<![CDATA[ non interpr. ]]>
```

Esempio

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
... Contenuto del documento ...
<![CDATA[
<html><head>...</head>
<body><p>contenuto html che potrebbe inficiare la buona formazione del
documento xml; infatti <p> non e` chiuso e adesso andiamo a capo con
<br> senza chiuderlo e poi aggiungiamo una riga scritta male con <hr>
</body></html>
 ]]>
```

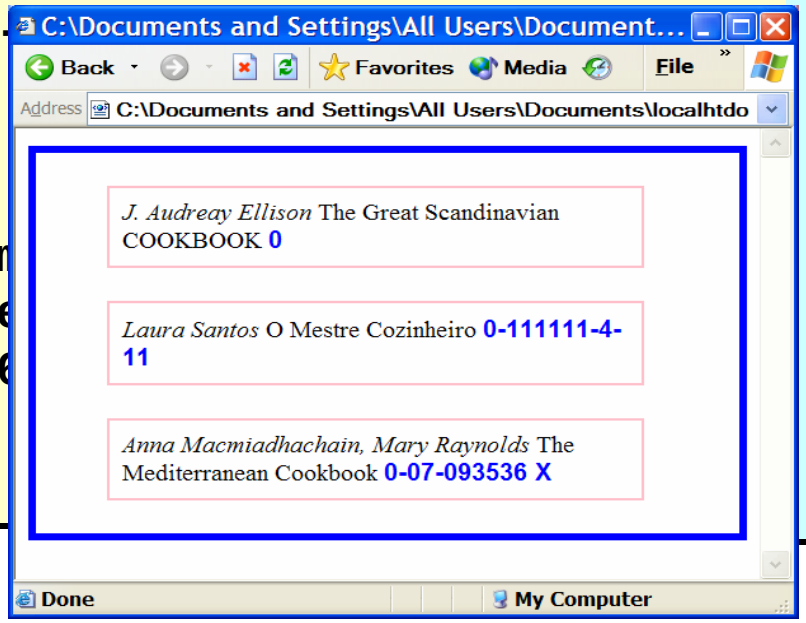
XML - presentazione con css

libri.1.xml / .css

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
                 href="libri.1.css"?>
<libri>
  <book>
    <author>J. Audreay Ellison</author>
    <title>The Great Scandinavian COOKBOOK</titl
    <isbn>0</isbn>
  </book>

  <book>
    <author> Laura Santos </author>
    <title>O Mestre Cozinheiro</title>
    <isbn>0-111111-4-
  </book>

  <book>
    <author>Anna Macn
    <title>The Medite
    <isbn>0-07-093536
  </book>
</libri>
```




```
libri {
 display: block;
 border: solid thick  blue;
 padding: 5%;
 color: black;
 font-size: 12pt;
}

book  {
 display:block;
 width: 90%;
 border: solid thin  pink;
 margin-left: 5%;
 margin-top: 5%;
 padding: 10px;

}

isbn  {
 ...
}

title {
 font-family:Times New
Roman, serif;
}

author  {
 font-style: oblique;
}
```

XML - progettazione di un documento (1/2)

Progettare e scrivere un documento xml vuol dire pensare a quali informazioni vi dovranno essere conservate e a quale sara' la strutturazione/distribuzione di tali informazioni negli elementi e nei relativi attributi: quali elementi servono? Quali attributi essi hanno? Quali sottoelementi ne possono/devono discendere?

Es. Varie possibilita' per definire un documento che contenga una sequenza di libri, disponibili in una libreria; ogni libro e' dato dal suo autore, titolo e numero isbn.

```
<?xml version="1.0"?>
<libri>
  <book>
    <author>J. A. Ellison</author>
    <title>The Great ...</title>
    <isbn>777-8-999-9-765</isbn>
  </book>
...
</libri>
```

```
<?xml version="1.0"?>
<libri>
 <book numISBN="777-8-999-9-7">
  <author>J. A. Ellison</author>
  <title>The Great ...</title>
 </book>
...
</libri>
```

```
<?xml version="1.0"?>
<libri>
  <book numISBN="777-8" auth="J.A.Ell..." title="The Great ..." />
...
</libri>
```

distribuzione delle informazioni tra gli elementi e gli attributi: dipende dal progettista, dalla struttura dei dati su cui si vuole che le applicazioni lavorino, ha qualche aspetto di scelta personale;

? dati brevi, assimilabili a valori/identificatori ---> attributo

? dati complessi/che possono essere ev. suddivisi (titolo/sottotitolo, nome/cognome), descrizioni ---> elemento

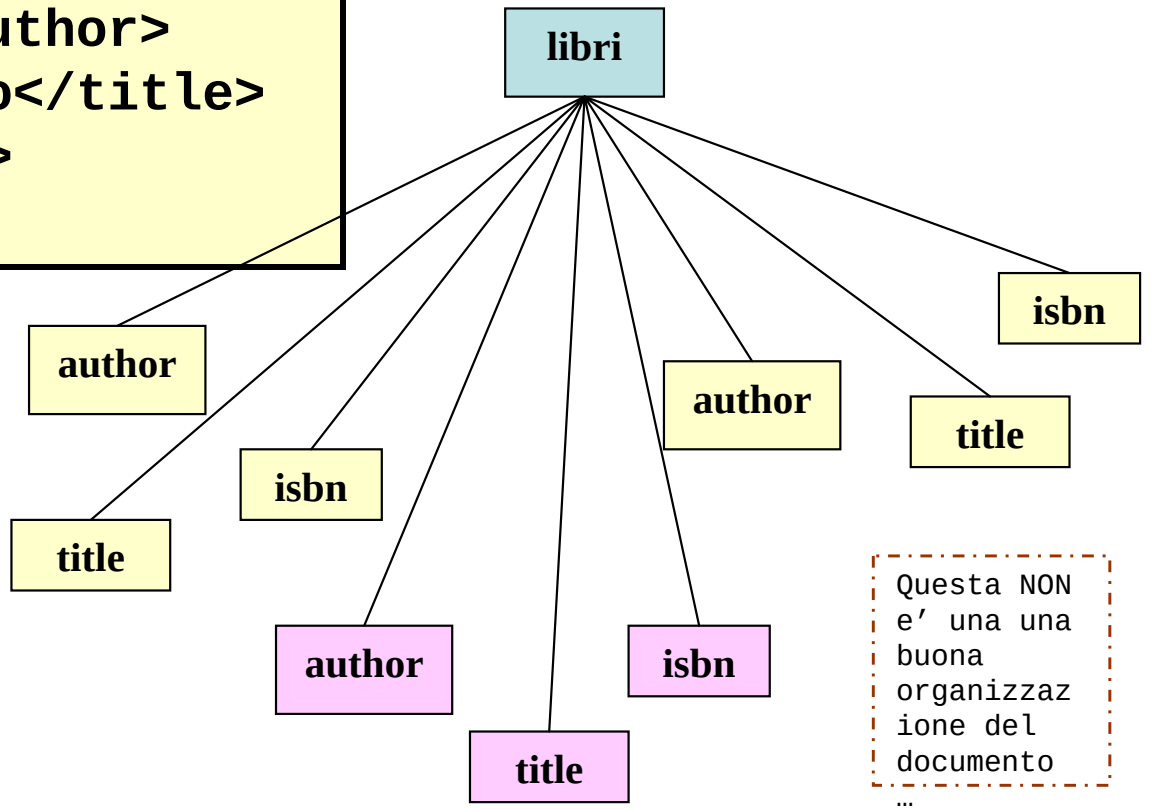
XML - progettazione di un documento (2/2)

Pero' comunque bisogna pensarci … se non si sta attenti e' possibile costruire documenti male strutturati (e quindi e meno "utilizzabili“)

Es. documento che contiene una sequenza di libri, disponibili in una libreria; ogni libro e' caratterizzato dal suo autore, titolo e numero isbn.

```
<?xml version="1.0"?>
<libri>
  <author> J. A. Ellison</author>
  <title>The Great ...</title>
  <isbn>777-8-999-9-765</isbn>
  <author> Laura Santos </author>
  <title>O Mestre Cozinheiro</title>
  <isbn>0-111111-4-11</isbn>
</libri>
```

Una buona organizzazione del documento (una buona strutturazione), lo rende piu' appetibile ad una applicazione che debba usarlo (rende piu' semplice costruire l'applicazione)



E' piu' facile far usare questo documento ad un'applicaziuone che deve aggiornare il documento, aggiungendo e togliendo libri?

E' piu' facile estendere questo documento in modo che ogni libro abbia anche l'indicazione del numero di copie disponibili in libreria?

E' piu' facile interfacciare questo documento con l'applicazione di gestione degli ordini di libri?

Questa NON e' una una buona organizzazione del documento …

XML – documento valido

documento XML, scritto con un linguaggio di markup

```
<?xml version="1.0"?>
<libri>
  <book>
    <author>...</author>
    <title>...</title>
    <isbn>...</isbn>
  </book>
 <book>
    <AUTHOR>...</AUTHOR>
    <title>...</title>
    <isbn>...</isbn>
  </book>
...
</libri>
```

linguaggio = insieme di regole che stabiliscono

- **quali elementi e loro combinazioni si possono/devono usare,**
- **quali attributi sono obbligatori / opzionali,**
- **quali valori per gli attributi sono ammissibili,**
- **…**

Document type definition (DTD) = specifica regole linguaggio

documento XML associato ad una DTD (*interna/esterna*): DOCTYPE!

se la DTD, cui e' associato il documento XML e' rispettata, questo e' VALIDO

Se l'elemento AUTHOR non e' definito nella DTD associata, e/o non e' previsto come sottoelemento di <book> in prima posizione, questo documento non e' valido

XML – DTD: Document Type Definition

Un **documento xml** contiene e rende disponibili dei dati ed e' **scritto** usando un **linguaggio di markup.**

Pertanto i dati contenuti nel documento sono racchiusi/distribuiti/disponibili all'interno di elementi, sottoelementi, attributi, …, dotati di certe proprieta' *per definizione:*

- gli **elementi** si chiamano in un certo modo prestabilito e non in un altro!
- e, in base alla loro definizione possono contenere certi **sottoelementi** ma non altri;
- in base alla sua definizione, un elemento puo'avere **attributi** con certi nomi ma non altri;
- gli attributi possono avere certi **valori** piuttosto che altri
- ...

La definizione di tali proprieta' e' data nella **DTD** associata al documento (e progettata per esso): una DTD specifica le **regole del linguaggio di markup** usato per scrivere il documento.

Se il documento rispetta le regole date nella sua DTD, allora e' **valido**.

Un documento valido e' buono per essere processato da un'applicazione, con speranze di successo.

- Se il documento xml non e' collegato ad una DTD, possiamo discutere se sia well formed o meno, ma non sappiamo ufficialmente quali regole e' previsto esso segua nel riportare le informazioni.
- Il che significa che non possiamo scrivere un'applicazione che usi quelle informazioni su basi rigorose (cioe' essendo sicura di quel che fa su quali dati).

- Un documento valido puo' essere scambiato tra due poli di un sistema di applicazioni, ciascuno dei quali fa affidamento su una definizione comune del formato dei dati e usa applicazioni proprie su quei dati

Una DTD puo' essere **interna** al documento, oppure **esterna** (scritta in un file separato).
Diversi documenti possono essere scritti facendo riferimento alla medesima DTD esterna, che quindi costituisce un ottimo modo per specificare un formato comune di dati, processabile automaticamente. Ad esempio due societa' che scambiano dati, possono scrivere e processare ciascuna i propri documenti. Se c'e' una una DTD comune, i documenti possono essere scambiati, per essere usati ad entrambi i capi. Se ci sono modifiche nel formato dei dati, basta cambiare la DTD comune e poi ognuno se la vede per conto proprio con il suo software.

XML – esempio di DTD – libri.2.xml

libri.2.xml

libri.2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE libri SYSTEM "libri.2.dtd">

<libri>
  <book>
    <author>J. Audreay Ellison</author>
    <title>The Great Scand...</ti
    <isbn>0</isbn>
  </book>

  <book>
    <author> Laura Santos </autho
    <title>O Mestre Cozinheiro</t
    <isbn>0-111111-4-11</isbn>
  </book>

  <book>
    <author>Anna Macmiadhachain, Mary
    <title>The Mediterranean Cookbook<
    <isbn>0-07-093536 X</isbn>
  </book>
</libri>
```

- elemento <libri> = radice del documento xml
- def. del tipo di doc.:libri.2.dtd

SYSTEM: riferimento specifico (URI) alla .dtd;
PUBLIC: rif. non specifico (responsabile l'appl.) ad una "nota" dtd (+ rif. secondario specifico)

```
<?xml version="1.0" encoding="UTF-8"
<!ELEMENT libri (book*)>
<!ELEMENT book (author, title, isbn)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
```

libri.2.dtd

Elemento libri: composto da 0/+ book

- elemento book: composto da un elemento author, seguito da un title e da un isbn (nell'ordine e tutti!)
- elemento author: composto da *Parsed Character Data*

Vedi libri.2.sb.xml

XML – esempio di DTD – estensione

libri.3.dtd / .xml

Vogliamo arricchire la struttura delle informazioni librarie, permettendo info

- su editore,anno,numero ediz. e autore della prefazione (nome e cognome);
- su piu' autori distinti (nome e cognome)
- attributi isbn e rating: rating puo' essere *sufficiente* (default), *buono* oppure *ottimo*;

```
<book isbn="0-111111-4-11">
   <author>
     <name>Anna</name>
     <surname>Macmi…</surname>
   </author>
   <author>
     <name>Mary</name>
     <surname>Raynolds</surname>
   </author>
   <title>O The Medit...</title>
   <year>1979</year>
   <publisher>McGraw</publisher>
   <edition>0-07-0936!!</edition>
   <preface>
     <name>Remy</name>
     <surname>Fougere</surname>
   </preface>
 </book>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT libri (book*)>
<!ELEMENT book (author+, title,
year?,publisher,edition?,preface)>

<!ATTLIST book
   isbn CDATA #REQUIRED
   rating (sufficiente | buono |
                ottimo) "sufficiente">

<!ELEMENT author (name, surname)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
<!ELEMENT preface (name,surname)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
```

vedi anche libri.3.sb.xml

XML – DTD - espressioni per gli elementi

E = elemento

```
E*      qualsiasi numero di occorrenze di E
E?      E opzionale (1 o 0)
E+      1 o piu' occorrenze di E
E, F    E seguito da F
a|b     uno dei due
(…)     gruppo di elementi
#PCDATA     testo (interpretato)
```

almeno 1 autore (ev. piu')

anno e numero edizione facoltativi

isbn e rating sono attributi di book: la lista di tali due attributi e' def in una clausola ATTLIST, in cui e' specificato che isbn e' di caratteri ed e' obbligatorio e rating puo' avere uno tra i valori specificati in OR, con "suff…" dato come default.

author e preface sono composti da due sottoelementi: nome e cognome (obbligatori e nell'ordine)

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT libri (book*)>
<!ELEMENT book (author+, title, year?,
   publisher, edition?, preface)>
<!ATTLIST book
          isbn CDATA #REQUIRED
          rating (sufficiente | buono |
   ottimo) "sufficiente">

<!ELEMENT author (name, surname)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
<!ELEMENT preface (name, surname)>

<!ELEMENT name (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
```

XML – DTD - keywords applicabili, e dove

keywords usate per definire parti di documento

!ELEMENT (vediamo il content model)
!ATTLIST (vediamo I tipi di attributo che si possono sfruttare)
!ENTITY
!NOTATION

keywords usabili in ATTLIST

ENTITY in ATTLIST	(definizione del posizionamento di entity come valori per gli attributi)
NOTATION in ATTLIST	(definizione di attributi i cui valori hanno NOTATION definita appositamente a livello di documento)
ID/IDREF in ATTLIST	(definizione di attributi che sono usati per dare e fare riferimento ad elementi del documento xml)

XML – DTD - content model/categorie per ELEMENT

Per *content model* si intende la forma su cui gli elementi sono modellati nella definizione: un elemento puo' essere qualificato secondo le seguenti categorie:

- *element-only* composto da sottoelementi (*childs*) e eventuali attributi

```
<!ELEMENT book (author+, title, year?, publisher, edition?)>
```

- *text-only* solo #PCDATA

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

- *mixed* elemento composto sia da occorrenze di sottoelementi che di #PCDATA

La definizione di questo tipo di elemento e' peculiare: deve essere solo come segue:

```
<!ELEMENT elem (#PCDATA | elem1 | elem2)*>
```

cioe' il termine PCDATA deve apparire come primo, seguito dagli eventuali sottoelementi, in un gruppo di scelte alternative.
Il gruppo deve essere quantificato con * (quindi riproponibile all'infinito nel documento). Cosi' e' possibile costruire legalmente qualunque sequenza di elementi, inframmezzata da testo. Vedi `wrox.xml`

- *empty* (sottocaso di element-only ... al piu' dotato di attributi)

```
<!ELEMENT Mr EMPTY>
```

- *any* (tutti i precedenti …) elemento contenente qualunque well formed xml

```
<!ELEMENT AnyThingGoesInHere ANY>
```

XML – DTD - definizione e tipi di attributi (uso di ATTLIST)

Il valore di un attributo, corrispondentemente alla definizione, puo' essere di vari tipi. In

```
<!ATTLIST book     isbn CDATA #REQUIRED
                   rating (suff | buono | ottimo) "suff">
```

si vedono in azione due possibili tipi:
- **CDATA** (*character data*: testo che non viene analizzato dal parser ma riportato fedelmente)
- **ENUMERATED** (una sequenza di opzioni mutuamente esclusive);

Anticipando quel che succede tra qualche slide, diciamo che altri tipi possibili sono

- **ID** un identificatore che permette di “dare nome univoco” ad un’istanza dell’elemento, in modo da potervisi riferire univocamente da altri elementi
- **IDREF** e' l'attributo che permette di fare riferimento da un elemento ad un altro elemento (per cui e' specificato un ID)
- **ENTITY** entita' definita per permettere di includere "testo predefinito" nel doc xml;
- **NOTATION** un’indicazione su tipi di dati non xml usati in un file (in questo caso in attributi), e sulle appl.che li possono gestire
- **NMTOKEN** nameToken: testo con qualche restrizione (ad es. solo lettere e cifre e niente spazi)

Per ogni attributo e' definibile un valore di **default** (come `"suff"`) e sono usabili i qualificatori

- **#REQUIRED** obbligatorio
- **#IMPLIED** opzionale
- **#FIXED** con un valore specificato (come per default) e non assegnabile

XML – DTD - keyword: ENTITY (a livello «alto» di DTD)

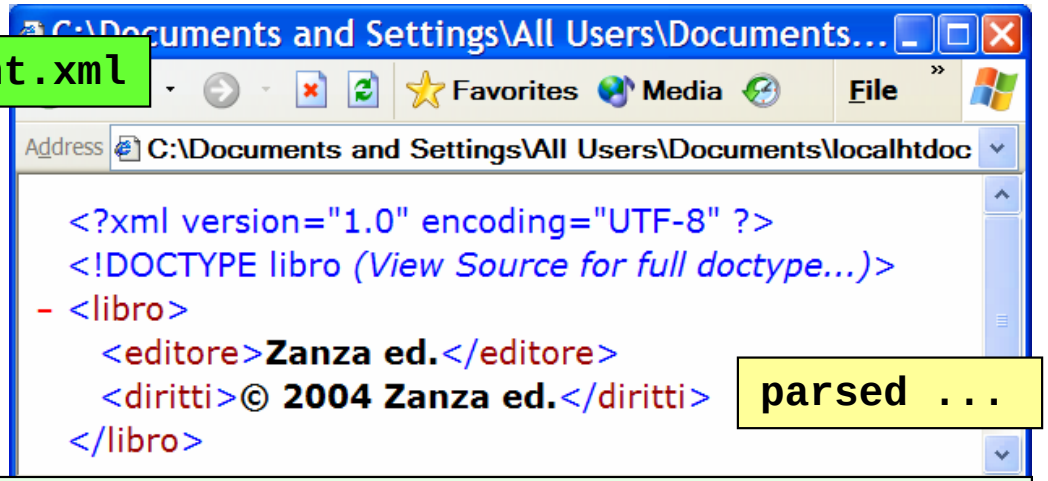
(Oltre a ELEMENT e ATTLIST, ci sono altre due keyword: ENTITY e NOTATION)

- **General** entities (entita' definite nella dtd ed usate nei file xml)
  - parsed entities: **internal** o **external**.

```
<!DOCTYPE libro
[
 <!ENTITY editore "Zanza ed.">
 <!ENTITY testoCopyright "&#169; 2004
                          &editore;">
]>
<libro>
  <editore>&editore;</editore>
  <diritti>&testoCopyright;</diritti>
</libro>
```

entities.int.xml

parsed ...



(DTD interna e uso di "character references" unicode)

  - unparsed entities: solo external.

```
<!ENTITY logo SYSTEM
"http://www.editore.com/images/logo.p
ng" NDATA png>
```

unparsed = "risorsa da usare con un'appl. di supporto/plug-in per il tipo di dato coinvolto" **= "c'e' poco da scorrere".** Uso: &logo;
(NB NOTATION sul tipo png deve essere definita!)

- **Parameter** entities: usate per definire abbreviazioni da usare all'interno di una DTD

```
<!ENTITY % CDTreq "CDATA #REQUIRED">
<!ENTITY % CDTopt "CDATA #IMPLIED">
```

```
<!ATTLIST book isbn %CDTreq
    ...
```

XML – DTD - keyword: NOTATION (a livello di DTD)

Usata per suggerire come gestire una unparsed entity o qualunque dato non xml (per esempio il formato di un'immagine).

```
<!NOTATION png SYSTEM "http://this.site.boh/Software/PNG_Viewer.exe" >
```

```
I dati qualificati di tipo png verranno trattati con questa applicazione
ad esempio
<!ENTITY logo SYSTEM "http://www.editore.com/images/logo.png" NDATA png>
```

XML – DTD - NOTATION in ATTLIST

NOTATION e' uno degli "attribute type" usati in una definizione di attributi (ATTLIST)

```
<!ELEMENT img EMPTY >
<!ATTLIST img
   src CDATA #REQUIRED
   type NOTATION (png | gif | jpg) #IMPLIED
```

```
Questa potrebbe essere una def di
   attributi per l'elemento xhtml
   img: si vuole che nella DTD
   sia data una definizione
   NOTATION per ciascuno dei tre
   tipi di immagine (con la
   specifica del relativo plugin)
```

XML – DTD - ENTITY in ATTLIST

ENTITY e' uno degli "attribute type" usati in una definizione di attributi (ATTLIST)

```
<!ELEMENT person ... >
<!ATTLIST person
   ...
   photo ENTITY #IMPLIED>
```

Questa potrebbe essere una def di attributi per una persona, in cui esiste un'attributo photo. L'attributo photo, nel documento xml, puo' essere assegnato come una entita' (che, se la cosa e' seria, dovrebbe essere a sua volta definita come rif. ad una immagine)

DTD del documento xml

```
<!NOTATION png SYSTEM "http://this.site.boh/Software/PNG_Viewer.exe" >
   ...     jpg       ...
<!ENTITY immagine1 SYSTEM "…/facce/anna.mac.png NDATA png>
<!ENTITY immagine2 SYSTEM "…/facce/mary.raynolds.jpg NDATA jpg>
...
```

external entities (un'altra DTD, o magari la stessa)

```
 <staff ...>
 <person ... photo="immagine1">
      <name>Anna</name>
      <surname>Macmiadhachain...
...
 <person ... photo="immagine2">
      <name>Mary</name>
      <surname>Raynolds</surname>
</staff>
```

documento xml

XML – DTD - ID/IDREF in ATTLIST

Servono per gestire riferimenti incrociati tra elementi.
L'elemento cui e' assegnato un determinato valore per l'attributo di tipo ID deve essere unico nel documento (in modo che quando si fa riferimento a quel valore di attributo si trova solo quell'unico elemento).

```
<book isbn="0-111111-4-11">
 <author authId="ANNMCMDHC">
   <name>...</name>
   <surname>...</surname>
 </author>
 <author authId="MRYRYNLDS">
   <name>...</name>
   <surname>...</surname>
 </author>
 <title>...</title>
 <year>1979</year>
 <publisher>...</publisher>
 <edition refAuth="ANNMCMDHC">
   2
 </edition>
 <preface>... </preface>
</book>
```

libri.4.xml

identificatore xml, case sensitive

```
<!ELEMENT author (name, surname)>
<!ATTLIST author
          authId ID #IMPLIED>
```

libri.4.dtd

```
...
<!ATTLIST edition
          refAuth IDREF #IMPLIED>
```

IDREFS per poter elencare nell'attributo diversi ID (blank-separati)

```
<edition
refAuth="ANNMCMDHC MRYRYNLDS">
```

In questo modo, nell'elemento **<edition>,** oltre specificare il numero dell'edizione del libro, e' possibile fare riferimento ad uno degli autori (che si suppone sia stato responsabile per questa edizione).

XML – DTD - Conclusioni

Una DTD permette di definire rigorosamente la struttura sintattica (linguaggio) del documento. Una DTD puo' essere messa in comune tra diversi documenti, dando loro una comune struttura sintattica e lessicale. L'esistenza di una DTD permette di validare i documenti che fanno ad essa riferimento e di passarli "seriamente" ad un'applicazione che ne processi le informazioni.

Ci sono delle limitazioni nel meccanismo d'uso delle DTD, comunque …

- **tutto in una DTD** (si usa una sola DTD per documento; e' difficile mischiare piu' DTD, cioe' vocabolari, in uno stesso documento)
- **una DTD non e' eXtensible!** (non si scrive in XML - semmai EBNF - e non si processa usando qualcuna delle molte applicazioni xml fiorite)

Nota: Con entita' esterne si puo' fare riferimento a DTD esterne, per arricchire la DTD, e si possono anche usare definizioni DTD nel documento xml, in aggiunta (e copertura!!) della DTD riferita. Di base queste macchinose flessibilita' si pagano in termini pesanti di complessita' e performance delle applicazioni.

- **definizioni interne al doc.** ricoprono quelle della DTD esterna (nessuna garanzia che il doc. segua le regole della DTD cui e' associato)
- **non si specificano i tipi dei dati che un elemento o un attributo puo' contenere, o intervalli di valori: i valori sono stringhe**

Sono state intraprese diverse iniziative per definire meccanismi alternativi alle DTD: la specifica di XML Schema e' quella adottata dal W3C, per cui ...

risorse

XML spec (the original) http://www.w3.org/TR/REC-xml/
(1.1. at http://www.w3.org/TR/2008/REC-xml-c14n11-20080502/)

XML Copy Editor http://xml-copy-editor.sourceforge.net/ e' un editor con check di buona formazione e validaione

xml-SPY *ambiente di programmazione per xml; supporto a DTD, schema, validazione e molto altro (www.altova.com, download di prova gratuito)*

esercizi provenienti dal testo consigliato *Nella dir.XML1/ ci sono file con esercizi e soluzioni tratti dal sito del libro di testo consigliato (alcune directory con nome num1.num2 dove num1 e' il capitolo e num2 l'esercizio).*

Testi di approfondimento *Beginning XML, D.Hunter, Wrox Press Ltd.*

When to use elements vs. attributes
http://www.ibm.com/developerworks/library/x-eleatt/

"XML galaxy" http://www.zvon.org

w3schools http://www.w3schools.com/xml/default.asp

Attivita' in laboratorio / prodotti individuali

XML-1

Ripercorrere le slides della lezione: usando un editor, per esempio textpad (o magari XML Copy, un browser, e un XML validator (c'e' in XML Copy, comunque), sperimentare e modificare gli esempi menzionati, leggere i commenti contenuti nei file di esempio. Copiare ciascun file con suffisso .sb (contenente errori di cattiva formazione del documento) in uno con lo stesso nome e suffisso .corr, che poi deve essere corretto. Ogni tanto ci si ferma per fare uno dei prossimi esercizi …

XML-2

Progettare un documento che risponda alle seguenti specifiche:

Bisogna rappresentare le informazioni sulle offerte di appartamenti di una agenzia immobiliare. Ogni offerta contiene la data di disponibilita' dell'appartamento, le informazioni su prezzo/numerocamere/numerobagni, il luogo dove si trova (indirizzo, citta'), informazioni sull'agente che ha in carico questo appartamento, e informazioni sugli appuntamenti presi per visitare l'appartamento (nome e cognome visitatore, data, se effettivamente svolto, se confermato). Scrivere il documento well formed e disegnarne l'albero. Visualizzarlo con IE e poi aggiungere un file di stile che ne permetta una visualizzazione piu' elegante (inserendo la relativa istruzione di processazione nel documento). Un suggerimento in `apartments/apartments2.xml`; i file `apartments.xml` e . `apartments.css` sono le soluzioni ad un esercizio simile del libro. Guardare anche `apartments2.sb.xml`.

Attivita' in laboratorio / prodotti individuali

XML-3

Partendo da libri.2.xml, ignorando per il momento la DTD, e rimuovendone il riferimento dal documento, aggiungere informazioni sui prezzi dei libri. Costruire una .css nuova, che permetta la visulaizzazione su uno schermo dei libri memorizzati. Costruire una seconda css, che visualizzi le informazioni sullo schermo di un telefonino (bisogna agire sulle dimensioni delle font e dell'output in generale). Notare come la sequenza di informazioni visualizzate e' sempre governata dall'ordine con cui le informazioni appaiono nel file xml (questa e' una limitazione dell'approccio alla presentazione mediante css).

Lavorando su questo esercizio, pu' essere di spunto quello numero 13 nel capitolo 3 del testo di riferimento.

XML-4

Partendo da libri.3.xml, libri.3.dtd aggiungere informazioni sui prezzi dei libri, in modo che l'elemento prezzo abbia un attributo che specifica la valuta (euro, magari come entita', e' il valore di default).

XML-5

L'esercizio 3.19 del testo di riferimento e' carino e istruttivo.

XML-6

Eccoci qui. Ci serviva un documento xml contenente una collezione di dischi (libera scelta se cd o vinile …).

Inizialmente ne abbiamo scritto una porzione, sufficiente a farci capire quali elementi e attributi servissero. Durante questa fase iniziale abbiamo usato spesso il browser per vedere se scrivevamo roba well formed.

Poi, con qualche disco completamente scritto, abbiamo steso la DTD. A volte abbiamo cancellato un elemento perche' quell'informazione pareva meglio stesse in un attributo. Il viceversa e' stato piu' raro. Certo che le definizioni della DTD rassomigliano proprio alla EBNF. Ah sono scritte in EBNF? Ma vah!

Durante la stesura della DTD abbiamo usato spesso un XML validator. Ce n'era uno in mezzo al materiale didattico; ma poi ci siamo accorti che era anche in XML Copy, e la vita e' sstata piu' faile. A volte abbiamo scoperto che i tag erano scritti male (maiuscole, minuscole) e abbiamo visto ancora una volta che un documento puo' essere well formed e non valido.

Poi abbiamo fatto una .css per permetterci di visualizzare il contenuto del documento su schermo. Veramente l'abbiamo fatta prima, anche se il prof. voleva che la facessimo dopo. E' andata bene lo stesso. Certo che gli errori scoperti mentre facevamo la DTD davano fastidio anche alla visualizzazione.

Anche se non era richiesto, anzi sconsigliato al momento, siamo riusciti pure a fare una versione della .css che usa le tabelle.