XML
(eXtensible Markup Language)

1. Cosa è XML?

- XML (Extensible Markup Language [sic!]) è un meta-linguaggio di markup, progettato per lo scambio e la interusabilità di documenti strutturati su Internet.
- XML prevede una sintassi semplificata rispetto a SGML, e definisce contemporaneamente una serie piuttosto lunga di linguaggi associati: uno per i link, uno per i nomi di tag, uno per i fogli di stile, uno per la descrizione di meta-informazioni, ecc.
- XML si propone di integrare, arricchire e, nel lungo periodo, sostituire HTML come linguaggio di markup standard per il World Wide Web.

Perché XML?

Oggi nel Web tutte le pagine sono di tipo HTML - HyperText Markup Language.

Hypertext: testo con links ad altri testi.
Markup Language: annotazione che descrive la struttura di un testo.

Ma HTML ha alcuni limiti

- HTML contiene un numero fissato di tags la cui semantica è definita e non può essere modificata; inoltre tali tags hanno solo scopo di visualizzazione.
- CSS (Cascading Style Sheets) è uno strumento che consente la separazione della struttura logica dal layout della pagina.

I vantaggi di XML (1)

- Documenti auto-descrittivi
  - La scelta dei nomi degli elementi può essere fatta per facilitare la comprensione del ruolo strutturale dell'elemento.
  - Inoltre, l'uso di un DTD può esplicitare le regole di composizione ed i rapporti possibili tra le varie parti dei documenti.
- Struttura navigabile dei documenti
  - La rigida struttura ad albero e l'assenza di regole di minimizzazione rendono semplice la visualizzazione e l'analisi della struttura del documento, e la possibilità di visualizzare il documento è indipendente dal foglio di stile che vi si applica.

I vantaggi di XML (2)

- Platform-independence
  - XML è uno standard aperto, e chiunque può realizzare strumenti che lo usino come formato di dati.
- Facile convertibilità a formati Web
  - La totale interdipendenza tra XML, SGML, HTML etc. fa sì che la conversione tra formati interni e formati per il Web sia facile.

Cosa si fa con XML?

- Data Interchange
  - Ogni volta che più programmi si debbono scambiare dati, ci sono problemi di compatibilità. Ogni programma ha le proprie assunzioni in termini di caratteri, separatori, ripetibilità di elementi, differenza tra elementi vuoti e assenti, ecc.
  - XML si propone come la sintassi intermedia più semplice per esprimere dati anche complessi in forma indipendente dall'applicazione che li ha creati.
- Document publishing
  - XML è ideale come linguaggio per esprimere documenti strutturati o semi strutturati, e per esprimerli in maniera indipendente dalla loro destinazione finale.
  - Lo stesso documento XML può essere preso e trasformato per la stampa, il Web, il telefonino, l'autoradio.

Cosa si fa con XML?

- Applicazioni che richiedono che il client Web si ponga a mediare tra due o più database eterogenei
- Applicazioni che distribuiscono una parte significativa del carico computazionale dal server al client
- Applicazioni che richiedono che il client Web presenti view diverse degli stessi dati agli utenti
- Applicazioni in cui agenti Web intelligenti adattano la scoperta di informazioni alle esigenze degli specifici utenti.

Da J. Bosak, *XML, Java, and the future of the Web*, **`http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm`**

Cosa si fa con XML?

- **Accesso a database eterogenei**
  - Ogni volta che è necessario trasferire dei dati da un database all'altro, la soluzione più economica a tutt'oggi è stampare i dati dal primo DB su carta e ribatterli a mano sul secondo.
  - Idealmente io vorrei accedere via Web ai dati del primo DB, selezionare quelli che voglio in una cartella, e sbattere la cartella sul secondo DB, che si preoccupa di adattarli alle sue esigenze.
  - Il secondo DB, dunque, deve essere in grado di comprendere la sintassi dei dati, di interpretare la struttura (eventualmente, in parte, aiutato da un essere umano) e di isolare le informazioni di suo interesse.
  - Per questo potrebbe essere aiutato da un formato di interscambio tipo XML, che permetterebbe di etichettare i dati esplicitamente ed in maniera generale e comprensibile agli esseri umani.

Cosa si fa con XML?

- **Computazioni client-side**
  - Esistono molte esigenze di testing e computazione su oggetti descrivibili parametricamente:
    - Caratteristiche e funzionalità di chip, semilavorati, e prodotti industriali
    - Scheduling in aerei, treni, ecc.
    - Shopping on-demand, e user-tailoring
    - Applicazioni per il customer suppot
  - In tutti questi casi, attualmente si creano applicazioni server-side che interrogano i database per i parametri e usano cicli del server per le computazioni, mentre i client sono in attesa.
  - Poter esprimere in Java o altri linguaggi client-side la logica della computazione, che scarica i parametri dal sito giusto ed esegue le computazioni indipendentemente, sarebbe molto comodo, e permetterebbe confronti incrociati e ogni altro tipo di valutazione ottimale per le esigenze di chi compra.

Cosa si fa con XML?

- **Viste selettive**
  - L'esempio tipico è l'indice sommario dinamico di un documento: interrogo una base documentaria e ottengo il primo livello di indice di un documento. Seleziono una voce e ri-interrogo la base dati per avere il secondo livello dell'indice.
  - Ogni espansione richiede un passaggio al server, con ovvi problemi di latenza. Sarebbe possibile fare tutto client-side con Javascript, ma o si fa l'indice a mano del documento HTML, oppure bisogna ricorrere a documenti ben strutturati, come XML.
  - Altri esempi:
    - Un grafico che si trasforma in una tabella
    - Un documento annotato in cui vedo il contenuto, o le annotazioni, o tutti e due
    - Un manuale di due versioni dello stesso sistema, con testi e immagini che cambiano a seconda di quale specifica versione si sta esaminando.

Cosa si fa con XML?

- **Agenti Web**
  - Mattew Fuchs (Disney Imagineering): "Data needs to know about itself, and data needs to know about me"
  - Agenti di filtro, selezione, rilevamento hanno bisogno di sapere le caratteristiche dei dati che stanno filtrando in maniera vendor-independent, ben strutturata e flessibile (nuove esigenze, categorie, comunità virtuali, sub-società si formano continuamente)
  - Ad esempio, bot personalizzati, la guida dei canali TV, i sistemi di classificazione del contenuto delle pagine Web, ecc.
  - Su questo specifico tema esistono argomenti di tesi di laurea.

Cosa c'è con XML?

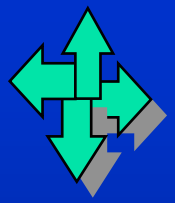
- XML è in realtà una famiglia di linguaggi, alcuni già definiti, altri in corso di completamento. Alcuni hanno l'ambizione di standard, altri sono solo proposte di privati o industrie interessate. Alcuni hanno scopi generali, altri sono applicazioni specifiche per ambiti più ristretti.

- Noi di occupiamo, tra gli altri, di:
  - XML 1.0: un meta-linguaggio di markup, sottoinsieme di SGML
  - XML-Namespace: un meccanismo per la convivenza di nomi di tag appartenenti a DTD diversi
  - XPath, XPointer e XLink: tre linguaggi per la creazione di link ipertestuali
  - XSL e XSLT: due linguaggi di stylesheet per XML
  - XML schema: un linguaggio per la specifica di criteri di validazione di documenti XML
  - RDF: un linguaggio per l'espressione di metainformazioni su documenti XML.

Overview: XML Architecture

XML : Extensible Markup Language

XSL : Extensible Stylesheet Language



XLL : Extensible Linking Language

DOM : Document Object Model

Criteri di progettazione di XML (1)

- Nel documento ufficiale di XML si elencano i seguenti obiettivi progettuali di XML:
  1. XML deve essere utilizzabile in modo diretto su Internet.
     - Non significa che deve essere possibile usarlo sul browser del giorno.
     - Significa che si dovevano tenere in conto le esigenze di applicazioni distribuite su reti a larga scala.
  2. XML deve supportare un gran numero di applicazioni.
     - Cioè XML non si limita al supporto di documenti in rete, ma a una larga classe di applicazioni che non c'entrano con la rete. Specificamente: deve essere possibile creare applicazioni come tool di authoring, filtri, formattatori, e traduttori.

Criteri di progettazione di XML (2)

3. XML deve essere compatibile con SGML
   - Tool SGML esistente debbono essere in grado di leggere e scrivere documenti XML
   - Istanze XML debbono essere istanze SGML così come sono, senza traduzioni, per quanto semplici.
   - Dato un documento XML, deve essere possibile generare un DTD SGML tale per cui un tool SGML esegue lo stesso parsing di un tool XML.
   - XML deve avere essenzialmente lo stesso potere espressivo di SGML.

Questi goal sono stati sostanzialmente raggiunti.

Criteri di progettazione di XML (3)

4. Deve essere facile lo sviluppo di programmi che elaborino documenti XML
   - Deve essere possibile creare applicazioni XML utili che non dipendano dal leggere ed interpretare il DTD
   - Obiettivo dichiarato: un diplomato in informatica deve essere in grado di scrivere un processore minimale XML in meno di una settimana.
5. Il numero di caratteristiche opzionali deve essere mantenuto al minimo possibile, idealmente a zero.
   - SGML, per generalità, aveva adottato un numero molto alto di caratteristiche opzionali, di dubbia utilità, o molto specifiche
   - Risultato: ogni processore SGML implementava solo una parte delle caratteristiche opzionali, e quindi documenti SGML conformi che potevano essere letti da un processore SGML non venivano letti da un altro, e viceversa.

Criteri di progettazione di XML (4)

6. I documenti XML dovrebbero essere leggibili da umani e ragionevolmente chiari.
   - Formati testuali sono più aperti, più utili, più gradevoli da lavorarci che formati binari.
   - Inoltre, per quanti capricci possa fare il tuo editor specializzato XML, puoi sempre aprire il documento con un editor di testi e rimettere a posto le cose.
7. La specifica del linguaggio XML deve avvenire rapidamente.
   - La paura era che le esigenze di estensibilità del Web potessero essere soddisfatte da una qualche combinazione di complicati formati binari e di accrocchi proprietari.
     Es: DHTML!

Criteri di progettazione di XML (5)

8. La progettazione XML deve essere formale e concisa.
   - La specifica di SGML è composta di un documento di oltre 300 pagine in testo, ottuso e burocratico. Il manuale SGML ne richiede più di 600, e comunque non è leggibile.
   - Inoltre non è neanche immediatamente utilizzabile da un programmatore per realizzare tool.
   - La scelta di formalismi nitidi e pochi commenti ha permesso la creazione di una specifica notevolmente più corta e immediatamente utilizzabile dai realizzatori di tool.
9. I documenti XML devono essere facili da creare.

   In particolare, deve essere facile creare tool di authoring di documenti XML.

XML e Unicode

- XML (come Java) abbandona completamente ASCII e le codifiche ad un byte, e si basa direttamente su Unicode.
- Questo porta a due vantaggi nei riguardi dell'internazionalizzazione:
  - È possibile scrivere documenti misti, senza ricorrere a trucchi strani per identificare la parte che usa un alfabeto dalla pare che ne adopera un altro.
  - Un documento scritto in un linguaggio non latino non deve basarsi su parametri esterni per essere riconosciuto come tale, ma la codifica stessa dei caratteri lo identifica.

Documenti ben formati o validi

- XML distingue due tipi di documenti rilevanti per le applicazioni XML: i documenti ***ben formati*** ed i documenti ***validi***.
- In SGML, un DTD è necessario per la validazione del documento. Anche in XML, un documento è **valido** se presenta un DTD ed è possibile validarlo usando il DTD.
- Tuttavia XML permette anche documenti **ben formati**, ovvero documenti che, pur essendo privi di DTD, presentano una struttura sufficientemente regolare e comprensibile da poter essere controllata.

Documenti XML ben formati

- Un documento XML si dice ben formato se:
  - Tutti i tag di apertura e chiusura corrispondono e sono ben annidati
  - Esiste un elemento radice che contiene tutti gli altri
  - I tag vuoti (senza contenuto) utilizzano un simbolo speciale di fine tag: <vuoto/>
  - Tutti gli attributi sono sempre racchiusi tra virgolette
  - Tutte le entità sono definite.

Parser validanti e non validanti

- Il cuore di un applicazione XML è il parser, ovvero quel modulo che legge il documento XML e ne crea una rappresentazione interna utile per successive elaborazioni (come la visualizzazione).
- Un parser validante, in presenza di un DTD, è in grado di verificare la validità del documento, o di segnalare gli errori di markup presenti.
- Un parser non validante invece, anche in presenza di un DTD è solo in grado di verificare la buona forma del documento.
- Un parser non validante è molto più semplice e veloce da scrivere, ma è in grado di fare meno controlli. In alcune applicazioni, però, non è necessario validare i documenti, solo verificare la loro buona forma.

2. XML : dichiarazione

<?XML version="1.0" encoding="UTF-16" standalone="yes" ?>

- Un documento XML può includere una dichiarazione XML. Questa specifica le caratteristiche opzionali del documento in questione. Poiché esse sono ridotte al minimo, la dichiarazione XML è brevissima.
- La sintassi usata per la dichiarazione XML è quella delle Processing Instructions,
- La non obbligatorietà della dichiarazione XML è dovuta a motivi di convenienza, per poter usare la grande quantità di documenti HTML e SGML che sono ben formati senza richiedere modifiche anche stupide. In assenza di dichiarazione XML, si assume la forma:

  <?XML version="1.0" ?>

Dichiarazione XML (2)

- Esistono esattamente tre valori che possono essere messi in una dichiarazione XML:
  - Il parametro "version" identifica quale versione di XML si sta usando. Per il momento, l'unico valore possibile è "1.0". Necessario.
  - Il parametro "encoding" permette di specificare, se il dubbio può sorgere, quale codifica di caratteri viene usata per il documento. Facoltativo.
  - Il parametro "standalone" permette di specificare se le informazioni necessarie per valutare e validare il documento sono interne o se ne esistono anche di esterne. Facoltativo.

2. XML

- Un documento XML contiene una varietà dei seguenti componenti
  - Elementi
  - Attributi
  - Entità
  - Testo (o #PCDATA)
  - Commenti
  - Istruzioni di Processo
  - Sezioni CDATA

Elementi

- Gli elementi sono le parti di documento dotate di un senso proprio.
- Il titolo, l'autore, i paragrafi del documento sono tutti elementi.
- Un elemento è individuato da un tag iniziale, un contenuto ed un tag finale.
- **Non confondere i tag con gli elementi!**

<TITOLO>Tre uomini in barca</TITOLO>

Peculiarità nella definizione di elementi

- **Elementi vuoti**: un elemento con content model EMPTY ha il carattere di chiusura tag ‘/>’.

  <EMPTY/>

- **Case sensitivity**: in XML tutto il markup è case-sensitive (il maiuscolo è diverso dal minuscolo). È quindi l’elemento <para> è diverso dall’elemento <PARA>.

Attributi

- Gli attributi sono informazioni aggiuntive sull'elemento che non fanno effettivamente parte del contenuto (meta-informazioni).
- Essi sono posti dentro al tag iniziale dell'elemento. Tipicamente hanno la forma nome=valore
- **Valori tra virgolette**: tutti i valori di tutti gli attributi debbono avere le virgolette (semplici o doppie, ma in maniera coerente), anche se numeri o appartenenti ad una lista di valori predefiniti.

<romanzo file='miofile'>…</romanzo>

<capitolo N=1>Capitolo primo</capitolo>

Entità

- Le entità sono frammenti di documento memorizzati separatamente e richiamabili all'interno del documento.
- Esse permettono di riutilizzare lo stesso frammento in molte posizioni garantendo sempre l'esatta corrispondenza dei dati, e permettendo una loro modifica semplificata.

Testo

- Rappresenta il contenuto vero e proprio del documento.
- Esso corrisponde alle parole, gli spazi e la punteggiatura che costituiscono il testo.
- Viene anche detto #PCDATA (Parsed Character DATA) perché SGML usa questo nome per indicare il contenuto di elementi di testo.

Commenti

- I documenti SGML possono contenere commenti, ovvero note da un autore all'altro, da un editore all'altro, ecc.
- Queste note non fanno parte del contenuto del documento, e le applicazioni SGML li ignorano.
- Sono molto comodi per passare informazioni tra un autore e l'altro, o per trattenere informazioni per se stessi, nel caso le dimenticassimo.

<!-- Questa parte è ignorata da SGML -->

Istruzioni di processo: <? … >

- Le **processing instructions** (PI) sono elementi particolari (spesso di senso esplicitamente procedurale) posti dall'autore o dall'applicazione per dare ulteriori indicazioni su come gestire il documento SGML nel caso specifico
- Per esempio, in generale è l'applicazione a decidere quando cambiare pagina. Ma in alcuni casi può essere importante specificare un comando di cambio pagina (oppure tutti i cambi pagina di un documento già impaginato).
- In questi casi, l'applicazione può inserire elementi di tipo PI: `<?NEWPAGE>` . Da notare che in XML la sintassi richiede un altro '?' alla fine: `<?NEWPAGE?>`

Sezioni CDATA

- A volte può essere comodo inserire un blocco di caratteri comprendenti anche ‘&’ e ‘<‘, senza preoccuparsi di nasconderli dentro ad entità.
- Si usa allora la sezione CDATA, che ha la seguente sintassi:
  <![CDATA[dati liberi comprendenti & e <]]>
- L’unica sequenza di caratteri non accettabile è la sequenza ‘]]>’, che definisce la fine della sezione CDATA
- Il parser XML passa all’applicazione finale tutti i caratteri che trova fino alla sequenza]]>

Un esempio

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<CATALOG>
  <CD year="1985">
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
  </CD>
    ....
</CATALOG>
```

Un esempio

```
<?xml version="1.0" standalone="no"?>
<radice>
    <curriculum>
            <dati>
                <nome>Luca</nome>
                <cognome>Rossi</cognome>
            </dati>
            <laurea>   <votol>110</votol>   </laurea>
     </curriculum>
      ....
</radice>
```

3. DTD: Document Type Definition

<!DOCTYPE nome TIPO [markup] >

- La dichiarazione del tipo del documento serve a specificare le regole che permettono di verificare la correttezza strutturale di un documento.
- Vengono cioè elencati [i file che contengono] gli elementi ammissibili, il contesto in cui possono apparire, ed altri eventuali vincoli strutturali.
- Nella terminologia SGML, si parla di modellare una classe (cioè una collezione omogenea) di documenti attribuendogli un tipo.

Inclusione di un DTD

- La dichiarazione di un tipo di documento deve essere la prima istruzione di un documento XML dopo l'opzionale intestazione o eventuali commenti.

```
<?XML version="1.0" rmd="all"?>
<!DOCTYPE chapter SYSTEM "dbook.dtd" [
<!ELEMENT ulink (#PCDATA)*>
<!ATTLIST ulink
   xml-link          CDATA  #FIXED "SIMPLE"
   xml-attributes    CDATA  #FIXED "HREF URL"
   URL               CDATA  #REQUIRED>
]>
<chapter>...</chapter>
```

Tipologie di inclusione

```
1 <!DOCTYPE mydoc SYSTEM "document.dtd">
2 <!DOCTYPE mydoc [
                    <!ELEMENT …
        ]>
3 <!DOCTYPE mydoc SYSTEM "document.dtd" [
                    <!ELEMENT …
        ]>
```

- La prima forma di dichiarazione indica che il DTD è contenuto in un file esterno (per esempio, condivisa con altri documenti). Il DTD viene chiamato *external subset* perché è posto in un file esterno.
- La seconda forma precisa il DTD internamente (cioè nello stesso file), che quindi non può essere condiviso da altri file. Il DTD si chiama in questo caso *internal subset*.
- La terza forma precisa una parte del DTD come contenuta in un file esterno (e quindi condivisibile con altri documenti), e una parte come propria del documento, e non condivisibile.

Dichiarazione Element

- Identifica il nome degli elementi usati nei file XML e il loro tipo

```
<!ELEMENT oldjoke       (burns+, allen, applause?)>
<!ELEMENT burns         (#PCDATA | quote)*>
<!ELEMENT allen         (#PCDATA | quote)*>
<!ELEMENT quote         (#PCDATA)*>
<!ELEMENT applause      EMPTY>
```

Specifica di elementi <!ELEMENT …>

<!ELEMENT nome ST ET content-model >

- ST & ET: minimizzazione del tag iniziale (ST) e finale (ET): può assumere i valori ‘-’ (*obbligatorio*) o ‘o’ (*omissibile*). Per noi sono sempre obbligatori.
- Content-model: la specificazione formale del contenuto permesso nell’elemento, secondo una sintassi specifica di gruppi di modelli.

Content model

- Tramite il content model posso specificare quali sono gli elementi leciti all'interno di un elemento, in quale numero e quale posizione rispetto agli altri.
- Un content model (CM) è '`ANY`', '`EMPTY`', oppure un gruppo di CM più elementari.
- Un gruppo di CM è sempre circondato da parentesi. Può contenere la specifica `#PCDATA`, la specifica di un elemento SGML, o di un altro gruppo di CM più elementare. Ogni specifica è separata da un separatore. Alla fine ci può essere un operatore di ripetizione.

ANY, EMPTY, #PCDATA

- ANY: significa che qualunque content model è ammesso. Ogni elemento definito nel DTD può comparire qui dentro in qualunque ordine e numero.
- EMPTY: Questo è un elemento vuoto, o senza contenuto. In questo caso nel documento esso appare come tag semplice, senza tag finale.

  Def.: <!ELEMENT HR - - EMPTY>
  Uso: <HR>

- #PCDATA: (Parsed Character Data): il contenuto testuale del documento. Include caratteri ed entità generali. È naturalmente in numero multiplo.

Separatori

- Separano specifiche determinando l'ordine o l'obbligatorietà:
  - ',' **(virgola)**: richiede la presenza di entrambe le specifiche nell'ordine precisato.
    Es.: **(a , b)**: ci devono essere sia a che b, e prima ci deve essere a e poi b.
  - '|' **(barra verticale)**: ammette la presenza di una sola delle due specifiche.
    Es.: **(a | b)**: ci può essere o a, oppure b, ma solo uno di essi.

Operatori di ripetizione (1)

- Permettono di specificare se un gruppo può comparire esattamente una volta, almeno una volta, oppure zero o più volte.
  - ***Niente***: la specifica precedente deve comparire esattamente una volta.
    Es.: c, (a, b): a e b devono comparire in quest'ordine esattamente una volta. È lecito solo: cab. Si dice che è una specifica *richiesta* e *non ripetibile*.
  - ***? (punto interrogativo)***: la specifica precedente può e può non comparire, ma solo una volta.
    Es.: c, (a , b)?: a e b possono comparire una volta, ma possono non comparire. Sono lecite: c, cab. Si dice che è una specifica *facoltativa* e *non ripetibile*.

Operatori di ripetizione (2)

- **+ *(più)***: la specifica precedente deve comparire almeno una volta. Es.: c, (a , b)+: a e b devono comparire almeno una volta, ma possono comparire anche più di una. Sono lecite: cab, cabab, cababababab, ma non c, ca, cb, cba, cababa. Si dice che è una specifica *richiesta* e *ripetibile*.
- ***** ***(asterisco)***: la specifica precedente deve comparire zero o più volte. Es.: c, (a, b)*: a e b possono comparire o no, a scelta e in numero libero. Sono lecite: c, cab, cabab, cababababab, ma non ca, cb, cba, cababa. Si dice che è una specifica *facoltativa* e *ripetibile*.

Element content semplice

<!ELEMENT sezione - - (titolo, abstract, para) >

- Un elemento contiene solo altri elementi, senza parti opzionali.
- Dentro all'elemento sezione ci deve essere un titolo, seguito da un abstract, seguito da un para.

```
<sezione>
        <titolo> … </titolo>
        <abstract> … </abstract>
        <para> … </para>
</sezione>
```

Element content complessi

<!ELEMENT sezione - - (titolo, abstract?, para+)>

- Un elemento contiene solo altri elementi, ma alcuni possono essere opzionali e altre in presenza multipla.
- Dentro all'elemento sezione ci deve essere un titolo, seguito facoltativamente da un abstract, seguito da almeno (ma anche più di) un para.

```
<sezione>
        <titolo> … </titolo>
        <para> … </para>
        <para> … </para>
</sezione>
```

Element content complesso

<!ELEMENT sezione - - (titolo, (abstract | para)+) >

- Un elemento contiene solo altri elementi, ma gli operatori di ripetizione e i separatori permettono sequenze complesse di elementi.
- Dentro all'elemento sezione ci deve essere un titolo, seguito da almeno uno di abstract o para, che poi possono ripetersi in qualunque ordine e in qualunque numero.

```
<sezione>
        <titolo> … </titolo>
        <para> … </para>
        <abstract> … </abstract>
        <para> … </para>
</sezione>
```

Character content

<!ELEMENT para - - #PCDATA >

- Un elemento contiene soltanto caratteri stampabili e entità. Nessun altro elemento è ammesso all'interno.

<para>Questo è lecito</para>

Mixed content

<!ELEMENT para - - (#PCDATA | bold)* >

- Un elemento contiene sia caratteri stampabili ed entità, sia altri elementi.
- <para>Questo è un paragrafo lecito con alcune <bold> parole in grassetto </bold> e poi <bold> ancora altre </bold>. </para>
- **Nota**: in XML la specifica #PCDATA deve sempre essere la prima di qualunque gruppo di mixed content. In SGML non è obbligatorio, ma è buono stile.

Dichiarazione Attribute

- Questa dichiarazione permette di specificare
  - quali elementi hanno attributi
  - quali sono i possibili valori degli attributi
  - eventuali valori di default

- Una dichiarazione è formata da tre parti:
  un nome, un tipo e un valore di default

```
<!ATTLIST nomeElemento
   name    ID                    #required
   label   CDATA                 #implied
   status  ( funny | notfunny )  'funny'>
```

Tipi di attributi (1)

- CDATA
  - Stringa, ossia un qualsiasi testo
- ID
  - Il valore deve essere un nome
  - I valori ID che compaioni in un file XML devono essere tutti diversi
  - Un elemento può avere un solo attributo ID
- IDREF or IDREFS
  - IDREF : il valore deve essere un ID di qualche elemento (corrisponde ad un puntatore)
  - IDREFS : contiene una lista di IDREF separati da spazio (corrisponde ad un puntatore a più elementi)

Tipi di attributi (2)

- ENTITY or ENTITIES
  - ENTITY : è il nome di una singola entità
  - ENTITIES : diverse ENTITY separate da spazio
- NMTOKEN or NMTOKENS
  - NMTOKEN : è una singola parola
  - NMTOKENS : lista di NMTOKEN

Valore di default letterale

- L'attributo assume quel valore se non ne viene specificato un altro.

```
<!ATTLIST doc
        stato    (bozza | impaginato | finale)  'bozza' >
```

l'uso

```
<doc stato='bozza'> Questo è un documento </doc>
```

e l'uso

```
<doc> Questo è un documento </doc>
```

sono uguali.

Valore di default #FIXED

- L'attributo assume automaticamente quel valore e non ne può assumere un altro. Il tentativo di assegnare un altro valore a quell'attributo produce un errore:

```
<!ATTLIST doc
        stato          CDATA          #FIXED 'bozza' >
```

l'uso

```
<doc stato='finale'> Questo è un documento </doc>
```

è un errore.

Valore di default #REQUIRED

- Non esiste valore di default: l'autore deve fornire ogni volta un valore.

```
<!ATTLIST doc
    stato      (bozza | impaginato | finale)          #REQUIRED
>
```

l'uso

```
<doc> Questo è un documento </doc>
```

è un errore.

Valore di default #IMPLIED

- Non è obbligatorio specificare un valore per questo attributo. Se esiste, verrà considerato il valore fornito, altrimenti l'applicazione deve fornirne un valore proprio.
- Gli attributi di tipo ID debbono avere un valore `#REQUIRED` (l'autore deve fornire un identificativo univoco ogni volta) o `#IMPLIED` (l'applicazione si preoccupa di fornire un identificativo interno).

Acronimi in XML

- CDF : Channel Definition Format
- CML : Chemical Markup Language
- CSS : Cascading Style Sheet
- DOM : Document Object Model
- DTD : Document Type Declarations
- HTML : HyperText Markup Language
- ICE : Internet Content and Exchange
- MCF : Meta Content Framework
- MathML : Mathematical Markup Language
- P3P : Platform for Privacy Preferences
- PICS : Platform for Internet Content
- RDF : Resource Description Framework
- SMIL : Synchronized Multimedia Integration Language
- SGML : Standard Generalized Markup Language
- WIDL : Web Interface Definition Language
- XLL : XML Link Language
- XML : eXtensible Markup Language
- XSL : XML Style Language