XML

Dato il proliferare senza regole di linguaggi markup, cominciò a delinearsi la necessità di un linguaggio di markup che offrisse **maggiore libertà nella definizione dei tag** pur rimanendo nell'ambito del rispetto di uno standard. Fu così che nel 1996 si costituì l'XML Working Group nell'ambito del W3C. Lo scopo del gruppo di lavoro era quello di definire un linguaggio che salvasse gli standard e offrisse libertà di estensione.

Esso è un meta-linguaggio di markup, cioè un linguaggio che permette di definire altri linguaggi di markup. A differenza di HTML, XML non ha tag predefiniti e non serve per definire pagine Web né per programmare. Esso serve esclusivamente per definire altri linguaggi.

In realtà, XML di per sé non è altro che un insieme standard di regole sintattiche per modellare la struttura di documenti e dati. Questo insieme di regole, dette più propriamente specifiche, definiscono le modalità secondo cui è possibile crearsi un proprio linguaggio di markup.

Anche l'HTML viene ridefinito secondo le regole di XML. Le differenze più evidenti per chi vuole passare da HTML a XHTML consistono in alcune regole sintattiche che possiamo così riassumere brevemente:

- tutti i tag e i loro attributi sono espressi in minuscolo
- è obbligatorio inserire il tag di chiusura (ad esempio, se usiamo dobbiamo chiudere con)
- i valori degli attributi devono essere specificati tra doppi apici o singoli apici (ad esempio, <table width="30%">)
- i tag vuoti seguono la cosiddetta sintassi minimizzata (per esempio, il tag <br> diventa <br/>)
- utilizzare l'attributo id al posto di name per identificare gli elementi di un documento

Intorno ad XML ruotano una serie di tecnologie e linguaggi che supportano l'uso di questo meta-*linguaggio e lo rendono affidabile e flessibile per gli usi più disparati.*

Ad esempio, tramite le definizioni di **Dtd** e **XML Schema** possiamo creare grammatiche che definiscono formalmente la struttura dei dati e ne consentono di verificare la correttezza; tramite CSS, XSL, XSL-FO possiamo controllare la presentazione dei dati e la loro trasformazione in altri formati; con XLink e XPointer possiamo collegare documenti XML mentre con XQuery e XQL possiamo estrarre informazioni secondo determinati criteri.

STRUTTURA documento

XML è dunque un meta-linguaggio per definire la struttura di documenti e dati. Il termine documento ricorre spesso nella terminologia XML. Anche se esso può far pensare pagine Web o altri prodotti dell'elaborazione di testo, è utilizzato nella sua accezione più ampia di **contenitore di informazioni**.

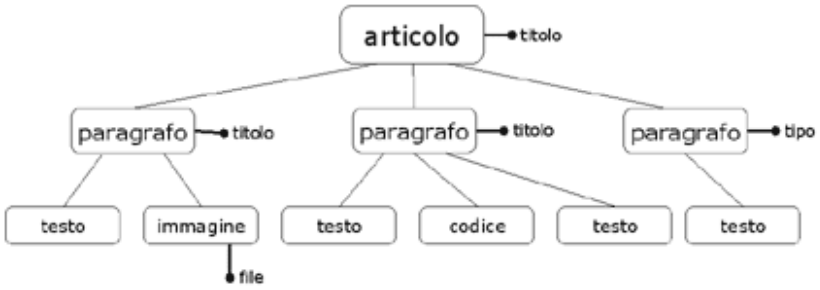
> ***Concretamente, un documento XML è un*** *file di testo* ***che contiene una serie di tag, attributi e testo secondo regole sintattiche ben definite.***

Analizziamo ora XML dal punto di vista logico e sintattico e i documenti che con esso si possono creare dando uno sguardo alla **struttura logica**. Introduciamo alcuni concetti fondamentali per la corretta comprensione di questo meta-linguaggio.

Un documento XML è intrinsecamente caratterizzato da una **struttura gerarchica**. Esso è composto da componenti denominati **elementi**. Ciascun elemento rappresenta un componente logico del documento e può contenere altri elementi (sottoelementi) o del testo.

Agli elementi possono avere associate altre informazioni che ne descrivono le proprietà. Queste informazioni sono chiamate **attributi**.

L'organizzazione degli elementi segue un ordine gerarchico o arboreo che prevede un elemento principale, chiamato **root element** o semplicemente **root** o **radice**. Possiamo rappresentare graficamente la struttura di un documento XML tramite un albero, generalmente noto come **document tree**.



La struttura logica di un documento XML viene tradotta in una corrispondente struttura fisica composta di elementi sintattici chiamati **tag**. Questa struttura fisica viene implementata tramite un file di testo creato con un qualsiasi editor. La rappresentazione fisica del documento XML visto prima può essere la seguente:

```
<?xml version="1.0" ?>
<articolo titolo="Titolo dell'articolo">
  <paragrafo titolo="Titolo del primo paragrafo">
    <testo>
      Blocco di testo del primo paragrafo
    </testo>
    <immagine file="immagine1.jpg">
    </immagine>
  </paragrafo>
  <paragrafo titolo="Titolo del secondo paragrafo">
    <testo>
      Blocco di testo del secondo paragrafo
    </testo>
    <codice>
      Esempio di codice
    </codice>
    <testo>
      Altro blocco di testo
    </testo>
  </paragrafo>
  <paragrafo tipo="bibliografia">
    <testo>
      Riferimento ad un articolo
    </testo>
  </paragrafo>
</articolo>
```

Documento ben formato

Perché un documento XML sia ben formato deve rispettare le seguenti regole:

- Ogni documento XML deve contenere **un unico elemento di massimo livello** (**root**) che contenga tutti gli altri elementi del documento. Le sole parti di XML che possono stare all'esterno di questo elemento sono i commenti e le direttive di elaborazione (per esempio, la dichiarazione della versione di XML)
- Ogni elemento deve avere un **tag di chiusura** o, se vuoti, possono prevedere la forma abbreviata (`/>`)
- Gli elementi devono essere opportunamente nidificati, cioè i tag di chiusura devono seguire l'**ordine** inverso dei rispettivi tag di apertura
- XML fa **distinzione tra maiuscole e minuscole**, per cui i nomi dei tag e degli attributi devono coincidere nei tag di apertura e chiusura anche in relazione a questo aspetto
- I valori degli attributi devono sempre essere racchiusi tra **singoli o doppi** apici

La violazione di una qualsiasi di queste regole fa in modo che il documento risultante non venga considerato ben formato. Anche se queste regole possono sembrare semplici, occorre prestarvi molta attenzione se si usa un semplice editor di testo. Soprattutto se si è abituati a lavorare con HTML. Codice come questo:

```
<articolo titolo=test>
        <!-- ... -->
</Articolo>
```

darà qualche problema, e lo stesso dicasi per situazioni analoghe alla seguente:

```
<paragrafo>
<testo>abcdefghi...
</paragrafo>
</testo>
```

Anche la **scelta dei nomi** dei tag deve seguire alcune regole. Un tag può iniziare con un lettera o un underscore (_) e può contenere lettere, numeri, il punto, l'underscore (_) o il trattino (-). Non sono ammessi spazi o altri caratteri. XML è sensibile all'uso di maiuscolo e minuscolo, quindi i tag `<prova>` e `<Prova>` sono considerati diversi.

Per quanto riguarda il **contenuto**, un documento XML può contenere potenzialmente qualsiasi carattere dell'alfabeto latino, cifre e punteggiatura. Normalmente vengono accettati come caratteri validi in un documento XML i primi 128 caratteri della codifica ASCII (lettere dell'alfabeto latino minuscole e maiuscole, cifre, segni di punteggiatura, etc.).

Se un documento contiene caratteri che non rientrano tra questi (es.: lettere accentate, simboli di valuta, ecc.) è necessario specificare lo schema di codifica utilizzato, ad esempio, la seguente direttiva di elaborazione:

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

abilita l'uso del set di caratteri noto come UTF-8 contenente le lettere accentate ed altri simboli.

Oltre alle direttive di elaborazione, in un documento XML possiamo trovare i **commenti**, cioè informazioni rivolte agli esseri umani ed ignorate dai software che lo elaborano. I commenti XML seguono la stessa

sintassi dell'HTML, sono cioè racchiusi tra le sequenze di caratteri `<!--` e `-->` e possono trovarsi in qualsiasi punto del documento. Potrebbe essere necessario inserire in un documento XML dei **caratteri particolari** che potrebbero renderlo non ben formato. Ad esempio, se dobbiamo inserire del testo che contiene il simbolo `<`, corriamo il rischio che possa venire interpretato come l'inizio di un nuovo tag, come nel seguente esempio:

```
<testo>
   il simbolo < indica minore di
</testo>
```

Per evitare situazioni di questo tipo, XML prevede degli oggetti speciali detti **entità** che consentono di sostituire altri caratteri. Cinque entità sono predefinite e consentono l'uso di altrettanti caratteri riservati all'interno di un documento:

Entità	Carattere corrispondente
`&`	**&**
`<`	**<**
`>`	**>**
`&quot;`	**"**
`&apos;`	**'**

Sfruttando le entità, l'esempio precedente diventa:

```
<testo>
   il simbolo &lt; indica minore di
</testo>
```

CDATA

In determinate situazioni gli elementi da sostituire con le entità possono essere molti, il che rischia di rendere illeggibile il testo ad essere umano. Si consideri il caso in cui un blocco di testo illustri proprio del codice XML:

```
<codice>
    <libro>
        <capitolo>
        </capitolo>
    </libro>
</codice>
```

In questo caso, al posto di sostituire tutte le occorrenze dei simboli speciali con le corrispondenti entità è possibile utilizzare una **sezione CDATA**. Una sezione CDATA (*Character DATA*) è un blocco di testo che viene considerato sempre come testo, anche se contiene codice XML o altri caratteri speciali. Per indicare una sezione CDATA è sufficiente racchiuderla tra le sequenze di caratteri `<![CDATA[` e `]]>`. Il nostro esempio diventerà come segue:

```
<codice>
        <![CDATA[
    <libro>
      <capitolo>
      </capitolo>
    </libro>
        ]]>
</codice>
```

In certe situazioni non si conosce a priori il contenuto che può essere inserito in un blocco di testo e pertanto l'utilizzo delle sezioni CDATA risulta obbligatorio.

Documenti validi

XML offre la libertà di definire i tag a seconda delle necessità, ma perché non si generi confusione è necessario un meccanismo che ne vincoli l'utilizzo all'interno dei documenti. Si può stabilire quali tag possono essere utilizzati e come per rispecchiare una struttura logica predefinita.

In altre parole abbiamo bisogno di definire una **grammatica** per il linguaggio di markup che abbiamo ideato. Una grammatica è un insieme di regole che indica quali vocaboli (elementi) possono essere utilizzati e con che struttura è possibile comporre frasi (documenti).

Una grammatica definisce uno specifico linguaggio linguaggio di markup. Dunque se un documento XML rispetta le regole definite da una grammatica è detto **valido** per un particolare linguaggio.

La caratteristica di documento valido si affianca a quella di documento ben formato per costruire documenti XML adatti ad essere elaborati automaticamente.

Grammatica XML

Ma come si definisce una grammatica per descrivere un linguaggio di markup? Attualmente due sono gli approcci più diffusi alla creazione di grammatiche per documenti XML:

1. **Dtd (**Document Type Definition) e
2. **XML Schema**.

Tutti i possibili impieghi di XML, si fondano su due tipi di elaborazione preliminare:

- la **verifica** che un documento sia **ben formato** e
- la sua **validità** rispetto ad una **grammatica**.

I software che si occupano di queste elaborazioni sono detti **parser** e sono degli strumenti standard disponibili sulle diverse piattaforme. Possiamo suddividere i parser in due categorie (talvolta può essere lo stesso parser che assume due ruoli):

- **parser non validante** è un parser che verifica soltanto se un documento è ben formato
- **parser validante** è un parser che, oltre a verificare che un documento è ben formato, verifica se è corretto rispetto ad una data grammatica

La maggior parte degli editor XML più recenti hanno un parser integrato o si appoggiano su parser esterni per effettuare la convalida dei documenti.

Dtd (Document type definition)

Un Dtd è un documento che descrive **i tag utilizzabili** in un documento XML, **la loro reciproca relazione** nei confronti della struttura del documento **e altre informazioni sugli attributi di ciascun tag**.

La **sintassi di un Dtd** si basa principalmente sulla presenza di due dichiarazioni:

1. **<!ELEMENT>** e
2. **<!ATTLIST>**.

La prima definisce gli elementi utilizzabili nel documento e la struttura del documento stesso, la seconda definisce la lista di attributi per ciascun elemento. Ad esempio, la dichiarazione:

```
<!ELEMENT articolo(paragrafo+)>
```

indica che l'elemento <articolo> ha come sottoelemento uno o più elementi <paragrafo>. Il carattere '+', dopo il nome del sottoelemento, indica il relativo numero di occorrenze.

Un insieme di caratteri speciali ha appunto lo scopo di indicare il **numero di occorrenze** di un elemento. In particolare:

- \+ indica che l'elemento è presente **una o più** volte
- * indica che l'elemento è presente **zero o più** volte
- ? indica che l'elemento è presente **zero o una** sola volta

Per esempio, la definizione

```
<!ELEMENT paragrafo(immagine*, testo+)>
```

indica che l'elemento <paragrafo> contiene la sequenza di elementi <immagine> e <testo>. L'elemento <immagine> può essere presente zero o più volte, mentre <testo> deve essere presente almeno una volta.

Per la definizione dei tag che non contengono sottoelementi dobbiamo distinguere il caso dei tag vuoti dai tag che racchiudono testo. Nel caso di tag vuoto, come accade per <immagine>, la definizione è

```
<!ELEMENT immagine EMPTY>
```

Nel caso di elementi che racchiudono testo abbiamo una definizione analoga alla seguente:

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

Esiste la possibilità di definire elementi il cui contenuto non è definito a priori, possono cioè essere vuoti o contenere altri elementi senza vincoli particolari. Per definire questo tipo di elementi si utilizza la seguente dichiarazione:

```
<!ELEMENT elemento ANY>
```

Per la definizione degli attributi di ciascun tag facciamo uso della dichiarazione <!ATTLIST>. Ad esempio, la dichiarazione:

```
<!ATTLIST articolo titolo CDATA #REQUIRED>
```

indica che l'elemento <articolo> prevede un attributo titolo che può avere come valore una qualsiasi combinazione di caratteri (CDATA). L'indicazione **#REQUIRED** indica che la presenza dell'attributo è obbligatoria. Valori alternativi a #REQUIRED sono:

- #IMPLIED Indica che l'attributo è opzionale
- #FIXED valore Indica che il valore dell'attributo è fisso ed equivale al valore specificato

Se un attributo prevede valori alternativi predefiniti è necessario specificarli al posto di CDATA, come accade per l'attributo tipo del tag <paragrafo>

```
<!ATTLIST paragrafo
    titolo   CDATA                          #REQUIRED
    tipo     (abstract|bibliografia|note)   #IMPLIED
>
```

In questo caso vengono definiti due attributi per l'elemento <paragrafo> facendo seguire alla definizione del primo attributo (titolo) quella del secondo (tipo). L'attributo tipo, opzionale, può assumere uno tra i valori abstract, bibliografia o note.

Il seguente codice riporta il Dtd completo per un documento che descrive un articolo analogo a quello visto negli esempi:

```
<!ELEMENT       articolo(paragrafo+)>
<!ELEMENT       paragrafo (immagine*, testo+, codice*)>
<!ELEMENT       immagine EMPTY>
<!ELEMENT       testo (#PCDATA)>
<!ELEMENT       codice (#PCDATA)>
<!ATTLIST       articolo titolo CDATA #REQUIRED>
<!ATTLIST       paragrafo
    titolo   CDATA                          #IMPLIED
    tipo     (abstract|bibliografia|note)   #IMPLIED
>
<!ATTLIST immagine file CDATA #REQUIRED>
```

Entità e documenti

Abbiamo visto come XML preveda degli elementi, detti entità, che consentono di sostituire caratteri speciali. Più in generale, una entità consente di sostituire sequenze di caratteri con **nomi speciali** della forma &nome;. È possibile definire entità personalizzate all'interno di un Dtd in modo da sostituire qualsiasi sequenza di caratteri.

Per definire un'entità personalizzata si utilizza la dichiarazione **<!ENTITY>**. Il seguente esempio mostra la definizione di un'entità &html; che rappresenta un'abbreviazione per la stringa HyperText Markup Language:

```
<!ENTITY html "HyperText Markup Language">
```

Grazie a questa dichiarazione possiamo utilizzare l'entità &html; al posto dell'intera stringa all'interno del documento XML che fa riferimento a questa grammatica.

Definito un Dtd abbiamo definito la grammatica per un linguaggio di markup. A questo punto dobbiamo mettere in relazione un documento **XML con il suo Dtd**, in modo che un parser XML possa verificare non soltanto la struttura ben formata del documento, ma anche la sua validità rispetto alla grammatica specificata.

Esistono **due modi** per indicare il Dtd cui un documento XML fa riferimento. **Il primo modo** prevede la presenza del Dtd all'interno del documento XML, come nel seguente esempio:

```
<?xml version="1.0">
<!DOCTYPE articolo[
…Definizioni del Dtd…
]>
<articolo>
…Contenuto del documento XML…
</articolo>
```

La dichiarazione **<!DOCTYPE>** indica che il documento individuato dall'elemento root <articolo> segue le regole definite tra le parentesi quadre.

Il secondo modo prevede che il Dtd sia definito in un file esterno ed il documento XML abbia un riferimento a tale file, come nel seguente esempio:

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

In questo caso si fa riferimento all'URL del Dtd definito nel file articolo.dtd. L'indicazione del file contenente il Dtd può essere espressa come URL assoluto o relativo. Ad esempio, se il Dtd viene pubblicato su un sito web è possibile specificare il riferimento al Dtd come nel seguente esempio:

```
<!DOCTYPE articolo SYSTEM "http://www.myXML.it/articolo.dtd">
```

Utilizzando i Dtd, quindi, abbiamo un maggior controllo sulla struttura e sull'uso dei tag in un documento XML, evitando che la libertà nella definizione dei tag possa far perdere il controllo sui contenuti.

Tuttavia l'uso dei Dtd per definire la grammatica di un linguaggio di markup **non sempre è del tutto soddisfacente**. A parte il fatto che la sintassi utilizzata per definire un Dtd non segue le regole stesse di XML, i Dtd non consentono di specificare un tipo di dato per il valore degli attributi, né di specificare il numero minimo o massimo di occorrenze di un tag in un documento o altre caratteristiche che in determinati contesti consentirebbero di ottenere un controllo ancora più accurato sulla validità di un documento XML.

Queste limitazioni hanno spinto alla definizione di approcci alternativi per definire grammatiche per documenti XML. Tra questi approcci il più noto è **XML Schema**.

XML schema

Analogamente ad un Dtd, un XML Schema è una descrizione formale di una grammatica per un linguaggio di markup basato su XML. L'approccio basato sui Dtd ci consente di specificare la struttura del nostro documento XML e di ciascun tag utilizzabile al suo interno con una precisione a prima vista accettabile.

Tuttavia, se abbiamo bisogno di un maggiore controllo sugli elementi che possono trovarsi all'interno di uno specifico tipo di documenti XML, i Dtd non risultano più sufficienti.

Ad esempio, i Dtd non mettono a disposizione un meccanismo immediato per indicare che un elemento può contenere al massimo un numero predefinito di sottoelementi, né è possibile specificare che un attributo può assumere valori di un certo tipo di dato, ad esempio valori numerici.

A differenza di un Dtd, che utilizza una propria sintassi specifica, un XML Schema utilizza la stessa sintassi XML per definire la grammatica di un linguaggio di markup.

> ***Quindi uno XML Schema è un documento XML che descrive la grammatica di un linguaggio XML utilizzando un linguaggio di markup specifico.***

In quanto documento XML, uno XML Schema ha un root element che contiene tutte le regole di definizione della grammatica.

La **struttura generale** di uno schema XML è la seguente:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        … Definizione della grammatica …
</xs:schema>
```

L'elemento root del documento è rappresentato dal tag <xs:schema>. Esso indica al parser che in questo documento saranno utilizzati dei tag definiti dal namespace standard del W3C.

Vedremo successivamente più in dettaglio cosa sono i namespace e come possiamo utilizzarli e definirli. Per il momento ci basti sapere che essi rappresentano un meccanismo per identificare tag appartenenti ad una specifica grammatica. Nel nostro caso questi **tag speciali** sono caratterizzati dal prefisso **xs:**.

XML Schema prevede il tag **<xs:element>** per la definizione degli elementi utilizzabili in un documento XML, specificando nell'attributo name il nome del relativo tag. All'interno di ciascun tag <xs:element> possiamo indicare il tipo di dato dell'elemento e possiamo definire gli eventuali attributi.

Ad esempio, la seguente definizione specifica l'elemento testo che può contenere soltanto stringhe:

```
<xs:element name="testo" type="xs:string" />
```

Questa dichiarazione corrisponde alla seguente dichiarazione Dtd:

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

Ma per comprendere meglio ed apprezzare la potenza degli XML Schema occorre analizzare il concetto di tipo di dato. Esistono due **categorie di tipi di dato**: semplici e complessi.

Dati semplici

XML Schema introduce il concetto di **tipo di dato semplice** per definire gli elementi che non possono contenere altri elementi e non prevedono attributi. Si possono usare tipi di dato semplici predefiniti oppure è possibile personalizzarli.

Sono previsti numerosi **tipi di dato predefiniti**, alcuni dei quali sono riportati nella seguente tabella:

Tipo di dato	Descrizione
xs:string	Stringa di caratteri
xs:integer	Numero intero
xs:decimal	Numero decimale
xs:boolean	Valore booleano
xs:date	Data
xs:time	Ora
xs:uriReference	URL

Ad esempio, la seguente dichiarazione:

```
<xs:element name=”quantita” type=”xs:integer” />
```

permette l'utilizzo dell'elemento quantita in un documento XML consentendo soltanto un contenuto di tipo intero. In altre parole, sarà considerato valido l'elemento <quantita>123</quantita> mentre non lo sarà l'elemento <quantita>uno</quantita>.

XML Schema prevede anche la possibilità di definire **tipi di dato semplici personalizzati** come derivazione di quelli predefiniti. Se, ad esempio, abbiamo bisogno di limitare il valore che può essere assegnato all'elemento <quantita> possiamo definirlo nel seguente modo:

```
<xs:element name=”quantita” >
  <xs:simpleType>
    <xs:restriction base=”xs:integer”>
      <xs:minInclusive value=”1” />
      <xs:maxInclusive value=”100” />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

In altre parole, la dichiarazione indica che l'elemento <quantita> è di tipo semplice e prevede una restrizione sul tipo di dato intero predefinito accettando valori compresi tra 1 e 100.

Dati complessi

I **tipi di dato complessi** si riferiscono ad elementi che possono contenere altri elementi e possono avere attributi. Definire un elemento di tipo complesso corrisponde a definire la relativa struttura.

Lo schema generale per la definizione di un **elemento di tipo complesso** è il seguente:

```
<xs:element name=”NOME_ELEMENTO”>
  <xs:complexType>
    … Definizione del tipo complesso …
    … Definizione degli attributi …
  </xs:complexType>
</xs:element>
```

Se l'elemento può contenere altri elementi possiamo definire la sequenza di elementi che possono stare al suo interno utilizzando uno dei **costruttori di tipi complessi** previsti:

<xs:sequence> Consente di definire una sequenza ordinata di sottoelementi

<xs:choice> Consente di definire un elenco di sottoelementi alternativi

<xs:all> Consente di definire una sequenza non ordinata di sottoelementi

Per ciascuno di questi costruttori e per ciascun elemento è possibile definire il numero di occorrenze previste utilizzando gli attributi **minOccurs** e **maxOccurs**. Ad esempio, se l'elemento testo può essere presente una o infinite volte all'interno di un paragrafo possiamo esprimere questa condizione nel seguente modo:

```
<xs:element name="paragrafo">
  <xs:complexType>
    <xs:element name="testo" minOccurs="1"
    maxOccurs="unbounded"/>
  </xs:complexType>
</xs:element>
```

In questo caso il valore **unbounded** indica che non è stabilito un massimo numero di elementi testo che possono stare all'interno di un paragrafo.

La definizione della struttura di un elemento è **ricorsiva**, cioè contiene la definizione di ciascun elemento che può stare all'interno della struttura stessa.

Per gli **elementi vuoti** è prevista una definizione basata sul seguente schema:

```
<xs:element name="NOME_ELEMENTO">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="xs:anyType" />
        … Definizione degli attributi …
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

In altri termini, un elemento vuoto è considerato un elemento di tipo complesso il cui contenuto non si basa su nessun tipo predefinito.

La definizione degli attributi è basata sull'uso del tag **<xs:attribute>**, come nel seguente esempio:

```
<xs:attribute name="titolo" type="xs:string" use="required" />
```

L'attributo **use** consente di specificare alcune caratteristiche come la presenza obbligatoria (**required**) o un valore predefinito (**default**) in combinazione con l'attributo value. Ad esempio, la seguente definizione indica un attributo il cui valore di predefinito è test:

```
<xs:attribute name="titolo" type="xs:string" use="default" value="test" />
```

Bisogna tener presente che se non si specifica esplicitamente l'obbligatorietà dell'attributo, esso è considerato opzionale. Il seguente codice presenta uno XML Schema relativo al linguaggio di descrizione di articoli tecnici mostrato nei vari esempi.

```
<?xml version=”1.0”?>
<xs:schema xmlns:xs=”http://www.w3.org/2001/XMLSchema”>
 <xs:element name=”articolo”>
  <xs:complexType>
   <xs:sequence>
    <xs:element name=”paragrafo” maxOccurs=”unbounded”>
     <xs:complexType>
      <xs:all maxOccurs=”unbounded”>
       <xs:element name=”immagine” minOccurs=”0”>
        <xs:complexType>
         <xs:attribute name=”file”
          use=”required”>
          <xs:simpleType>
           <xs:restriction base=”xs:string”/>
          </xs:simpleType>
         </xs:attribute>
        </xs:complexType>
       </xs:element>
       <xs:element name=”testo”/>
       <xs:element name=”codice” minOccurs=”0”/>
      </xs:all>
      <xs:attribute name=”titolo” type=”xs:string”
      use=”optional”/>
      <xs:attribute name=”tipo” use=”optional”>
       <xs:simpleType>
        <xs:restriction base=”xs:string”>
         <xs:enumeration value=”abstract”/>
         <xs:enumeration
         value=”bibliografia”/>
         <xs:enumeration value=”note”/>
        </xs:restriction>
       </xs:simpleType>
      </xs:attribute>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name=”titolo” type=”xs:string”
   use=”required”/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Questo XML Schema è equivalente al Dtd che abbiamo visto prima “Dtd: Document Type Definition” .

Dichiarazioni tipi

XML Schema prevede la possibilità di rendere modulare la definizione della struttura di un documento XML tramite la dichiarazione di tipi e di elementi.

Nel corso della creazione di uno schema XML possiamo analizzare ciascun sottoelemento significativamente complesso e **fornire una definizione separata** come elemento o come tipo di dato.

Questo contribuisce a fornire una **struttura modulare** allo schema, più ordinata, più comprensibile e semplice da modificare. Sfruttando la struttura modulare delle dichiarazioni, il contenuto di uno XML Schema diventa una sequenza di dichiarazioni di tipi ed elementi.

Possiamo definire un tipo complesso in base al seguente schema:

```
<xs:complexType name=”nome_tipo”>
        …
</xs:complexType>
```

Il riferimento ad una dichiarazione di tipo viene fatta come se fosse un tipo predefinito, come mostrato nel seguente esempio:

```
<xs:element name=”nome_elemento” type=”nome_tipo” />
```

La possibilità di dichiarare elementi e tipi di dato implica l’esistenza di un **ambito di visibilità** o contesto dei componenti dichiarati. I componenti di uno schema dichiarati al livello massimo, cioè come sottoelementi diretti dell’elemento root, sono considerati dichiarati a livello globale e possono essere utilizzati nel resto dello schema.

Nella dichiarazione di un tipo complesso è possibile fare riferimento ad elementi già esistenti dichiarati a livello globale oppure si possono **definire nuovi elementi**. La definizione di nuovi elementi all’interno di una definizione di tipo o di elemento costituisce una dichiarazione a livello locale. Ciò vuol dire che l’utilizzo di questi elementi è limitato alla definizione del tipo complesso in cui sono dichiarati e non possono essere utilizzati in altri punti dello schema.

I nomi degli elementi devono essere univoci nel contesto in cui compaiono. Questo significa, però, che in contesti diversi possiamo avere elementi con lo stesso nome ma con struttura diversa senza rischio di conflitti.

Per fare un’**analogia con i classici linguaggi** di programmazione, le dichiarazioni globali e locali di componenti di uno schema corrispondono alle dichiarazioni di variabili globali e locali in un’applicazione.

Il seguente codice riporta lo XML Schema per un linguaggio di descrizione di articoli visto prima, riorganizzato alla luce della possibilità di definire tipi di dato.

```
<?xml version=”1.0”?>
<xs:schema xmlns:xs=”http://www.w3.org/2001/XMLSchema”>
 <xs:element name=”articolo”>
  <xs:complexType>
   <xs:sequence>
    <xs:element name=”paragrafo” type=”paragrafoType”
    maxOccurs=”unbounded”/>
   </xs:sequence>
  <xs:attribute name=”titolo” type=”xs:string”
  use=”required”/>
  </xs:complexType>
 </xs:element>
 <xs:complexType name=”paragrafoType”>
  <xs:all maxOccurs=”unbounded”>
   <xs:element name=”immagine” type=”immagineType”
   minOccurs=”0”/>
   <xs:element name=”testo”/>
   <xs:element name=”codice” minOccurs=”0”/>
  </xs:all>
  <xs:attribute name=”titolo” type=”xs:string”
  use=”optional”/>
  <xs:attribute name=”tipo” use=”optional”>
   <xs:simpleType>
    <xs:restriction base=”xs:string”>
     <xs:enumeration value=”abstract”/>
     <xs:enumeration value=”bibliografia”/>
     <xs:enumeration value=”note”/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
 <xs:complexType name=”immagineType”>
  <xs:attribute name=”file” use=”required”>
   <xs:simpleType>
    <xs:restriction base=”xs:string”/>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
</xs:schema>
```

Come si può vedere, la **leggibilità dello schema** è molto maggiore rispetto alla prima versione. Inoltre, i tipi definiti a livello globale possono essere riutilizzati nel caso servisse modificare lo schema e quindi la struttura dei documenti XML risultanti.

Presentazione con CSS

A differenza di HTML, che è un linguaggio specifico di strutturazione e presentazione di documenti, XML è più generale e non ha una **semantica di presentazione**. Non è previsto alcun meccanismo predefinito per visualizzare i vari elementi di un documento.

Ad esempio, un documento **XML visualizzato in un browser** appare generalmente così com'è, al massimo con una indentazione e una colorazione dei tag impostata dal browser.

Un metodo per gestire la **presentazione del contenuto** di un documento XML consiste nell'utilizzare i Cascading Style Sheets (CSS).

È possibile **utilizzare i CSS** in modo analogo a come si utilizzano con HTML. Per ciascun elemento del documento XML che vogliamo formattare occorre definire una regola secondo lo schema:

```
selettore { proprietà: valore; proprietà: valore; … }
```

Il selettore specifica a quale elemento la regola deve essere applicata, mentre la parte racchiusa tra parentesi graffe elenca le caratteristiche da impostare e il relativo valore.

È opportuno evidenziare una **importante differenza** tra l'utilizzo dei CSS per formattare documenti HTML e il loro uso per i documenti XML. In HTML la maggior parte dei tag ha una formattazione predefinita e pertanto un foglio di stile CSS consente di ridefinire tali impostazioni.

In XML i tag non hanno alcun significato di formattazione, pertanto è necessario **specificare tutto**. Ad esempio, senza l'opportuna indicazione il testo contenuto nei diversi elementi di un documento XML verrebbe visualizzato come un'unica stringa.

Per **strutturare visivamente il documento** dobbiamo indicare la modalità di visualizzazione di ciascun elemento tramite la proprietà display di CSS. Ad esempio, per formattare l'elemento paragrafo di un articolo possiamo definire una regola come la seguente:

```
paragrafo {display: block; font-size: 12pt; text-align: left}
```

Generalmente un foglio di stile CSS da applicare ad un documento XML viene salvato in un file di testo con estensione .css (in realtà l'estensione usata è irrilevante). Nel documento XML possiamo quindi **inserire un riferimento** ad esso mediante unìapposita direttiva di elaborazione, come nel seguente esempio:

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

Questa dichiarazione fa in modo che un browser abilitato applichi le impostazioni del foglio di stile CSS specificato al documento XML.

I fogli di stile CSS sono pensati principalmente per il Web e mancano pertanto di alcune caratteristiche che possono risultare utili in ambiti diversi. Ad esempio, per la presentazione su supporti cartacei occorrerebbero maggiori funzionalità per l'impaginazione. tra le principali **limitazioni**, non è prevista la possibilità di estrarre il valore degli attributi degli elementi in modo da poterli visualizzare.

ESEMPI

N.1

Codice xml

```
<note>
 <nome>Gianni</nome>
 <data>01-01-2019</data>
 <ora>15.30</ora>
 <causale>RECUPERO!</causale>

 <nome>Pippo</nome>
 <data>01-01-2019</data>
 <ora>15.30</ora>
 <causale>Recupero Info</causale>
</note>
```

File dtd: **note.dtd**

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

<!DOCTYPE note [
    <!ELEMENT note (nome,data,ora,causale)>

    <!ELEMENT nome      (#PCDATA)>
    <!ELEMENT data      (#PCDATA)>
    <!ELEMENT ora       (#PCDATA)>
    <!ELEMENT causale   (#PCDATA)>
]>
```

File xml-schema: **note.xsd**

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

<xs:schema elementFormDefault="qualified">

    <xs:element name="note">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="nome"     type="xs:string"/>
                <xs:element name="data"     type="xs:string"/>
                <xs:element name="ora"      type="xs:string"/>
                <xs:element name="causale"  type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

File css: **note-stile.css**

```
note {
        display: block;
        color: brown;
        border-style: ridge;
        border-bottom: solid;
        }
nome {
        display: block;
        font-weight: bold;
        font-size: 1.3em;
        }
data {
        display: block;
        color: red;
        font-style: normal;
        }
ora {
        display: block;
        color: red;
        font-style: normal;
        }
causale {
        display: block;
        color: red-brown;
        font-style: italic;
        }
```

File xml che richiama dtd e css: **note-dtd.xml**

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/css" href="note-stile.css" ?>

<!DOCTYPE note SYSTEM "note.dtd">

<note>
 <nome>Gianni</nome>
 <data>01-01-2019</data>
 <ora>15.30</ora>
 <causale>RECUPERO!</causale>

 <nome>Pippo</nome>
 <data>01-01-2019</data>
 <ora>15.30</ora>
 <causale>Recupero Info</causale>
</note>
```

File xml che richiama xsd e css: **note-xsd.xml**

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/css" href="note-stile.css" ?>

<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                        xsi:schemaLocation="note.xsd">
  <nome>Gianni</nome>
  <data>01-01-2019</data>
  <ora>15.30</ora>
  <causale>Recupero Info</causale>

  <nome>Pippo</nome>
  <data>01-01-2019</data>
  <ora>15.30</ora>
  <causale>Recupero TPS</causale>
</note>
```

N.2

File xml con dtd incorporato e richiamo di css: **libreria-dtd-css.xml**

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/css" href="libreria-stile.css" ?>

<!DOCTYPE biblioteca [
	<!ELEMENT biblioteca (libro*)>
	<!ELEMENT libro (autori, titolo, casaed, edizione)>
	<!ELEMENT autori		(autore*)>
	<!ELEMENT autore		(nome, cognome)>
	<!ELEMENT nome		(#PCDATA)>
	<!ELEMENT cognome		(#PCDATA)>
	<!ELEMENT titolo		(#PCDATA)>
	<!ELEMENT casaed		(#PCDATA)>
	<!ELEMENT edizione		(luogo, anno)*>
	<!ELEMENT luogo		(#PCDATA)>
	<!ELEMENT anno		(#PCDATA)>
	<!ATTLIST libro
		codice CDATA #IMPLIED
		lingua CDATA #REQUIRED
	>
]>

<biblioteca>
	<libro>
		<autori>
			<autore>
				<nome>P.</nome>
				<cognome>Ollari</cognome>
			</autore>
		</autori>
		<titolo>Corso di sistemi e reti</titolo>
		<casaed>Zanichelli</casaed>
		<edizione>
			<luogo>Padova</luogo>
			<anno>2019</anno>
		</edizione>
	</libro>

	<libro>
		<autori>
			<autore>
				<nome>A.</nome>
				<cognome>Lorenzi</cognome>
			</autore>
			<autore>
				<nome>A.</nome>
				<cognome>Colleoni</cognome>
			</autore>
		</autori>
		<titolo>Tecnologie e progettazione di sistemi informatici e di telecomunicazioni</titolo>
		<casaed>Atlas</casaed>
		<edizione>
			<luogo>Paova</luogo>
			<anno>2019</anno>
		</edizione>
	</libro>
</biblioteca>
```

File css: **libreria-stile.css**

```
libro {
        display: block;
        color: Brown;
        border-style: ridge;
        border-bottom: solid;
        }
autore {
        display: block;
        color: red;
        font-style: italic;
        }
titolo {
        display: block;
        font-weight: bold;
        font-size: 1.3em;
        }
```

Sommario