XML

Indice

1. 1. Introduzione
   Cosa è e a cosa serve l'eXtensible Markup Language (XML)

Storia e applicazioni di XML

1. 2. Un po'di storia
   Storia di XML: dall'ideazione alla standardizzazione del W3C
2. 3. Il mondo di XML: XHTML e oltre
   L'evoluzione di HMTL e le applicazioni possibili di XML

Documenti, dati e rappresentazioni

1. 4. Struttura dei Documenti XML
   Come è rappresentata l'informazione in un documento XML
2. 5. Documenti ben formati
   Regole per una corretta stesura di documenti in formato XML
3. 6. Documenti validi
   Scrivere e convalidare documenti XML definiti secondo grammatiche

Definizione e applicazione di grammatiche

1. 7. Dtd: Document Type Definition
   Definire e descrivere grammatiche per la validazione di documenti XML
2. 8. Entità, documenti e considerazioni su XML
   Definire caratteristiche personalizzate dei documenti nel DTD
3. 9. XML Schema: elementi e struttura
   Cosa è un XML Schema e quali sono le principali differenze con DTD
4. 10. Tipi di dato
   Definire gli elementi con i tipi di dato principali di XML Schema
5. 11. Dichiarazione di tipi
   Semplificare ls struttura di un XML Schema e renderlo modulare.
6. 12. Integrazione di grammatiche e namespace
   Assegnare ad un documento XML la sua grammatica di riferimento
7. 13. Sintassi dei namespace
   Usare correttamente gli identificatori dei namespace nelle dichiarazioni

Modelli di presentazione dei documenti XML

1. 14. Presentazione di XML con CSS
   Definire caratteristiche presentazione per documenti XML con CSS
2. 15. XSL: eXtensible Stylesheet Language
   Presentare dati XML di natura diversa sotto forme diverse con XSL
3. 16. XPath: espressioni e funzioni
   Raggiungere le informazioni desiderate cercando percorsi in XML
4. 17. Fogli di stile XSLT
   Modelli per la elaborazione e trasformazione dei documenti XML

5. 18. Elaborazioni con XSLT
Costrutti XSLT per la elaborazione e la selezione dei contenuti

Conclusioni ed appendici

1. 19. Conclusioni
Breve riepilogo ed alcune considerazioni finali su XML
2. 20. Appendice A
Applicazioni e tecnologie basate su XML
3. 21. Appendice B
Alcune indicazioni sui software per la manipolazione di XML

Introduzione

Contrariamente a quanto comunemente si pensa, l'eXtensible Markup Language (XML) non è l'ennesimo linguaggio di markup né l'evoluzione dell'ormai vecchio, ma sempre vivo, HTML. 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. Le specifiche ufficiali sono state definite dal W3C (Worl Wide Web Consortium) e sono consultabili a partire dall'indirizzo http://www.w3.org/XML.

Ma perché si sente tanto parlare di XML? Perché tutti ne parlano quasi come una rivoluzione in ambito informatico? Cosa è possibile fare con questo meta-linguaggio? Come è possibile definire ed utilizzare un proprio linguaggio di markup?

A queste domande cercherà di dare una risposta la presente guida, il cui obiettivo consiste nel fornire le idee generali delle tecnologie che ruotano intorno ad XML. Ma andiamo per ordine e cominciamo a vedere perché si è sentita l'esigenza di introdurre XML.

Un po'di storia

Internet ed il Web, per la loro stessa natura non centralizzata ed aperta a macchine e persone di diversa natura, hanno bisogno di standard per poter essere utilizzabili e per poter evolvere senza forzature. A questo scopo è stato istituito il Worl Wide Web Consortium nel dicembre del 1994 con l'obiettivo di definire standard accettati dai maggiori produttori di software per il Web, primo fra tutti il linguaggio HTML.

Tuttavia, l'assalto commerciale ad Internet degli anni '90 e la rapida diffusione del Web ha scatenato una delle lotte più agguerrite sul piano tecnico e commerciale: la **guerra dei browser** tra Netscape e Microsoft. Ciascun contendente introduceva, con ogni nuova versione del proprio browser, una estensione proprietaria all'HTML ufficiale. Il risultato di tale battaglia era che un sito Web che voleva utilizzare le estensioni proprietarie di un browser rischiava di risultare inaccessibile agli altri browser. La situazione peggiorò con l'introduzione del Dynamic HTML, le cui implementazioni erano quasi totalmente arbitrarie.

In questo panorama il W3C era costretto a rincorrere le evoluzioni de facto dell'HTML e doveva scegliere quali caratteristiche standardizzare e quali invece lasciare fuori dalle specifiche ufficiali dell'HTML.

In questo contesto 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.

La ricerca partì, come era già accaduto in passato per l'HTML, dal linguaggio **SGML** (Standard Generalized Markup Language), un meta-linguaggio per la definizione di linguaggi di markup. Questo linguaggio risultava però troppo complesso per gli scopi della ricerca e pertanto fu snellito da alcune caratteristiche e semplificato in alcuni punti per renderlo adatto allo scopo.

Nel dicembre '97 le specifiche di XML venivano pubblicate come **Proposed Recommendation**. Tuttavia, anche se gli obiettivi iniziali della nascita di XML erano rivolti alla soluzione di un problema di standard per il Web, ben presto ci si accorse che XML non era limitato al solo contesto Web. Esso risulta essere abbastanza generale per poter essere utilizzato nei più disparati contesti: dalla definizione della struttura di documenti allo scambio di informazioni tra sistemi diversi, dalla rappresentazione di immagini alla definizione di formati di dati. Questo aspetto rappresentava una rivoluzione.

Il mondo di XML: XHTML e oltre

A questo punto, dopo aver conosciuto la storia dell'origine di XML, la domanda che generalmente viene fatta è: allora XML sostituirà l'HTML?

Anche se il dubbio risulta legittimo, possiamo dire che XML e HTML non sono in diretta concorrenza. È vero che le specifiche di HTML sono ferme alla versione 4.0 del dicembre 1997 (se si ignorano le piccole revisioni che hanno dato origine alla versione 4.01) e che non subiranno ulteriori evoluzioni. Tuttavia XML contribuisce proprio all'evoluzione dell'HTML.

Infatti, l'evoluzione del linguaggio HTML è XHTML (eXtensible HTML), una ridefinizione di HTML in termini di XML. In altre parole, XHTML è HTML definito 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
 diventa
)
- utilizzare l'attributo id al posto di name per identificare gli elementi di un documento

Possono sembrare differenze di poco conto, soprattutto perché la tolleranza dei browser ci ha abituato a commettere errori sintattici, ma sono molto importanti soprattutto perché le pagine strutturate in XHTML possono essere elaborate molto più facilmente da software diversi dai classici browser, come ad esempio gli spider dei motori di ricerca, i sintetizzatori vocali o gli innumerevoli altri dispositivi portatili e/o wireless che si stanno affacciando su Internet.

Abbiamo detto all'inizio che XML è un insieme standard di regole sintattiche per modellare la struttura di documenti e dati. Due parole di questa definizione sono di fondamentale importanza: standard e struttura.

Le regole in questione sono standard e questo garantisce l'indipendenza da una specifica piattaforma hardware e software o da uno specifico produttore. Pertanto, se definiamo un nostro linguaggio di markup tramite XML possiamo stare sicuri che esistono già strumenti per le diverse piattaforme in grado di comprendere ed elaborare correttamente il nostro linguaggio.

Le regole di XML consentono di definire la struttura di documenti e dati, ma non altre caratteristiche come il tipo o la presentazione dei dati o documenti. Questo compito non è di XML ma è delegato ad altre tecnologie, alcune delle quali sono basate sullo stesso XML, a dimostrarne la flessibilità ed universalità.

Viene fuori quindi che 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.

Questi pochi esempi ci fanno capire come XML non sia semplicemente un modo nuovo di ridefinire cose vecchie, ma c'è tutto un mondo che consente di ottenere risultati in diversi contesti ed in maniera standard.

Struttura dei Documenti XML

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.

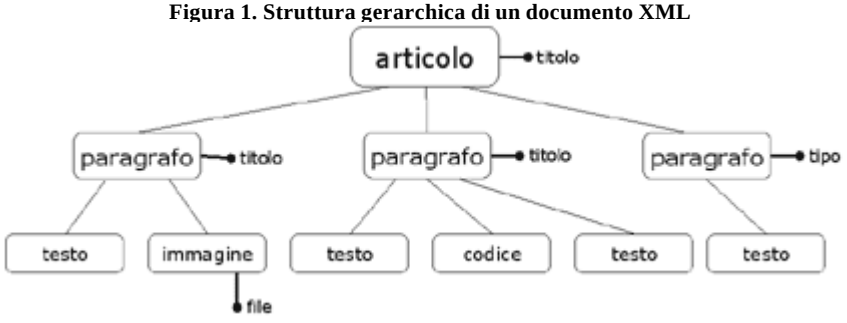
Gli elementi possono avere associate altre informazioni che ne descrivono le proprietà. Queste informazioni sono chiamate **attributi**.

L'organizzazione degli elementi segue un ordine gerarchico o arboreo che prevede un elemento principale, chiamato **root element** o semplicemente root o radice.

La radice contiene l'insieme degli altri elementi del documento. Possiamo rappresentare graficamente la struttura di un documento XML tramite un albero, generalmente noto come **document tree**.

Per fissare le idee prendiamo in considerazione la rappresentazione di un generico articolo a carattere tecnico e proviamo a rappresentarlo secondo il modello XML, come mostrato in figura.

Figura 1. Struttura gerarchica di un documento XML



Nella figura abbiamo un root element denominato articolo che contiene una lista di elementi che rappresentano i vari paragrafi dell'articolo. Ciascun paragrafo a sua volta contiene del testo, degli esempi di codice e delle immagini. La maggior parte degli elementi di questo document tree possiede degli attributi: titolo, tipo, file.

La struttura logica di un documento XML dipende dalle scelte progettuali. Siamo noi a decidere come organizzare gli elementi all'interno di un documento XML. Non esistono regole universali per l'organizzazione logica di un documento se non il buon senso e l'esperienza.

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>
```

Analizziamo il contenuto del documento per trarre alcune considerazioni. La prima riga del documento lo identifica come un documento XML e ne specifica la versione (in questo caso la 1.0):

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

Il corpo vero e proprio del documento segue questa prima riga, rappresentando gli elementi tramite tag, cioè sequenze di caratteri delimitate dai segni '<' e '>' proprio come avviene per l'HTML.

A differenza dell'HTML in cui i tag sono predefiniti, XML ci lascia liberi di **definire i tag** che vogliamo. Per specificare un attributo per un elemento inseriamo il nome dell'attributo con il relativo valore all'interno del tag di apertura dell'elemento. L'organizzazione gerarchica degli elementi viene rappresentata in XML tramite il loro annidamento.

Alcuni elementi possono essere vuoti, cioè possono essere privi di contenuto testuale. A differenza di quanto avviene per l'HTML, che consente l'utilizzo di elementi senza tag di chiusura, XML prevede che vengano sempre specificati i **tag di apertura e chiusura**. È questo il caso del tag immagine.

Tuttavia, XML prevede una **sintassi abbreviata** per gli elementi vuoti che evita di dover specificare il tag di chiusura. È infatti sufficiente terminare il tag di apertura con la sequenza di caratteri "/>", come nel seguente esempio:

<immagine file="immagine1.jpg" />

Le due notazioni per gli elementi vuoti sono equivalenti.

Documenti ben formati

XML richiede un certo **rigore** sugli aspetti sintattici. Abbiamo visto nel paragrafo precedente che un tag vuoto deve necessariamente prevedere il tag di chiusura o in alternativa la versione abbreviata del tag.

Questo non è altro che un aspetto di un principio fondamentale a cui tutti i documenti XML devono sottostare: tutti i documenti XML devono essere ben formati (well formed). Questo concetto è assimilabile in qualche modo alla correttezza ortografica di una lingua ed è un principio a cui i documenti XML non possono sottrarsi.

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 del tipo

```
<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, ecc.).

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. Lo schema di codifica ed altre informazioni dirette al software incaricato di elaborare il documento XML sono indicate tramite elementi speciali detti **direttive di elaborazione** o processing instruction.

Ad esempio, la seguente direttiva di elaborazione:

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

abilita l'uso del set di caratteri noto come Latin 1 contenente le lettere accentate ed altri simboli.

Le specifiche di XML prevedono esplicitamente la possibilità di utilizzare la codifica Unicode per rappresentare anche caratteri non latini, come ad esempio i caratteri greci, cirillici, gli ideogrammi cinesi e giapponesi.

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:

- & definisce il carattere &
- < definisce il carattere <
- > definisce il carattere >
- " definisce il carattere "
- ' definisce il carattere '

Sfruttando le entità, l'esempio precedente diventa:

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

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.

C'è da sottolineare che un documento ben formato può non essere valido rispetto ad una grammatica, mentre un documento valido è necessariamente ben formato. tra l'altro, un documento valido per una grammatica può non essere valido per un'altra grammatica.

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: **Dtd** - Document Type Definition e **XML Schema**. Questi argomenti saranno affrontati in seguito.

Un documento XML può essere all'origine di diversi **tipi di elaborazione** : generazione di altri documenti, eventualmente in formati diversi, controllo delle impostazioni di programmi, rappresentazione di immagini, ecc.

Tutti i possibili impieghi di XML, però, 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.

Per la validazione di un documento XML tramite codice è possibile utilizzare diverse librerie. tra le più note segnaliamo: MSXML (componente di Internet Explorer dalla versione 4.0 in poi, quindi è presente su tutte le macchine che hanno installato questo browser), Xerces e XML4J.

Dtd: Document Type Definition

Da un punto di vista cronologico, il primo approccio per la definizione di grammatiche per documenti XML è rappresentato dai **Document Type Definition (DTD)**.

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: **<!ELEMENT>** e **<!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, documenti e considerazioni su XML

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'URI 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: elementi e struttura

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. La cosa può sembrare paradossale, ma è invece indice dell'estrema flessibilità di XML.

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.

Tipi di dato

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.

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 nella lezione n.7 "Dtd: Document Type Definition" .

Dichiarazione di 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 nel paragrafo precedente, 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.

Integrazione di grammatiche e namespace

A partire da una grammatica definita tramite uno XML Schema, è possibile sfruttare un parser XML validante per **verificare la validità** di un documento XML. Il parser avrà bisogno, quindi, sia del documento XML da validare, sia dello schema XML rispetto a cui effettuare la validazione.

Ci sono diversi modi per fornire al parser informazioni sullo **schema da utilizzare** per la validazione. Uno di questi modi consiste nell'inserire nel documento XML un riferimento allo schema da utilizzare. Questo riferimento viene associato all'elemento root come nel seguente esempio:

```
<articolo
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="articolo.xsd"
  titolo="Guida ad XML" >
```

Oltre allìattributo titolo sono stati aggiunti due attributi predefiniti.

L'attributo **xmlns:xsi** indica un URL che specifica la modalità con cui si indicherà il riferimento allo schema XML. L'attributo **xsi:noNamespaceSchemaLocation** indica il nome e l'eventuale percorso del file contenente lo schema XML di riferimento.

Una delle caratteristiche auspicabili nella creazione di un nuovo linguaggio è la possibilità di integrare elementi derivanti da grammatiche diverse. Questa caratteristica consente di **riutilizzare parti di grammatiche** già definite evitando di dover rifare parte di lavoro già fatto in altri ambiti.

In questo modo, ad esempio, sarebbe possibile integrare un linguaggio per la descrizione di articoli tecnici con un linguaggio pre-esistente per la descrizione di bibliografie, ottenendo un nuovo linguaggio composto. Tuttavia la **composizione di linguaggi** pone almeno due tipi di problemi:

- un documento che utilizza due grammatiche presenta il **problema della validazione** : a quale schema si deve fare riferimento per validare un documento XML "ibrido"?
- due linguaggi potrebbero avere tag ed attributi con lo stesso nome, anche se utilizzabili in contesti diversi: come fare a risolvere questo tipo di **ambiguità**?

La soluzione a questi problemi deriva dai **namespace**. Un namespace è un insieme di nomi di elementi e nomi di attributi identificati univocamente da un identificatore.

L'identificatore univoco individua l'insieme dei nomi distinguendoli da eventuali omonimie in altri namespace. Per fare un esempio, se nell'ambito di una grammatica per descrivere dei dati anagrafici è stato definito un elemento indirizzo, questo nome potrebbe essere confuso con l'elemento indirizzo definito nell'ambito di una grammatica che descrive messaggi di posta elettronica. L'identificatore del relativo namespace consente di distinguere i due elementi omonimi.

Il concetto non è nuovo nell'informatica: ad **esempio**, quando definiamo i nomi dei campi in una tabella di un database abbiamo definito un namespace. Non possiamo avere campi con lo stesso nome all'interno di una tabella, ma possiamo avere gli stessi nomi in tabelle diverse. Possiamo risolvere l'ambiguità tra due campi omonimi facendoli precedere dal nome della tabella (il namespace).

Nell'ambito delle tecnologie XML, un **XML Schema definisce implicitamente un namespace** degli elementi e degli attributi che possono essere usati in un documento XML.

Se in un documento XML si utilizzano **elementi definiti in schemi diversi** abbiamo bisogno di un meccanismo che permetta di identificare ciascun namespace e il relativo XML Schema che lo definisce.

Sintassi dei namespace

In un documento XML si fa riferimento ad un namespace utilizzando un attributo speciale (**xmlns**) associato al root element, come nel seguente esempio:

<articolo **xmlns**="http://www.dominio.it/xml/articolo">

Questo indica che l'elemento articolo ed i suoi sottoelementi utilizzano i nomi definiti nel namespace identificato dall'identificatore http://www.dominio.it/xml/articolo.

L'identificatore di un namespace può essere rappresentato da una qualsiasi stringa, purché sia univoca. Proprio per garantirne l'univocità, è prassi ormai consolidata **utilizzare un URI** (Uniform Resource Identifier) come identificatore.

È bene evidenziare che non è necessario che l'indirizzo specificato come identificatore di namespace corrisponda ad un file pubblicato sul Web. Esso è utilizzato semplicemente come identificatore ed il parser non accederà al Web per verificare l'esistenza dell'URL.

Per mettere in relazione un namespace con il relativo XML Schema occorre dichiararlo nel root element come nel seguente esempio:

```
<articolo
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.dominio.it/xml/articolo"
  xmlns="http://www.dominio.it/xml/bibliografia"
  xsi:schemaLocation="http://www.dominio.it/xml/articolo
   articolo.xsd"
  xsi:schemaLocation="http://www.dominio.it/xml/bibliografia
   bibliografia.xsd"
>
```

L'attributo **xmlns:xsi** specifica la modalità con cui viene indicato il riferimento allo schema, mentre l'attributo **xsi:schemaLocation** indica il namespace ed il file in cui è definito il relativo XML Schema separati da uno spazio.

Eì possibile **combinare più namespace** facendo in modo che ciascun elemento utilizzato faccia riferimento al proprio namespace.

Occorre tener presente che quando si fa **riferimento ad un namespace**, questo riferimento vale per l'elemento corrente e per tutti gli elementi contenuti, a meno che non venga specificato un diverso namespace.

Il seguente esempio utilizza elementi tratti da due diversi namespace: uno relativo alla grammatica della struttura di un articolo e l'altro relativo alla grammatica di bibliografie:

```
<articolo xmlns="http://www.dominio.it/xml/articolo" titolo="Guida ad XML">
 <paragrafo titolo="Introduzione">
  <testo>
   bla bla bla
  </testo>
 </paragrafo>
```

```
 <paragrafo titolo="Bibliografia">
  <bibliografia
   xmlns="http://www.dominio.it/xml/bibliografia">
   <autore>
    Tizio
   </autore>
   <titolo>
    Opera citata
   </titolo>
   <anno>
    1999
   </anno>
  </bibliografia>
 </paragrafo>
</articolo>
```

Riportare il riferimento ad un namespace per ogni elemento è di solito scomodo e rende di difficile lettura il documento XML. È possibile creare delle abbreviazioni per fare riferimento ai namespace.

Queste **abbreviazioni** sono costituite da caratteri alfanumerici seguiti da due punti (:) dichiarati nel root element ed utilizzati come prefissi dei nomi degli elementi. Il seguente esempio riporta il codice XML precedente facendo uso di questi prefissi:

```
<art:articolo titolo="Guida ad XML"
 xmlns:art="http://www.dominio.it/xml/articolo"
 xmlns:bibl="http://www.dominio.it/xml/bibliografia" >
  <art:paragrafo titolo="Introduzione">
   <art:testo>
    bla bla bla
   </art:testo>
  </art:paragrafo>
  <art:paragrafo titolo="Bibliografia">
   <bibl:bibliografia>
    <bibl:autore>
     Tizio
    </bibl:autore>
    <bibl:titolo>
     Opera citata
    </bibl:titolo>
    <bibl:anno>
     1999
    </bibl:anno>
   </bibl:bibliografia>
 </art:paragrafo>
</art:articolo>
```

Le dichiarazioni xmlns:art e xmlns:bibl assegnano i prefissi art: e bibl: ai relativi namespace e questi prefissi vengono utilizzati per ciascun elemento del documento XML.

Presentazione di XML 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.

Considerando il seguente esempio

<articolo titolo="Guida ad XML">

non abbiamo la possibilità di visualizzare il titolo dell'articolo che è stato espresso come attributo dell'elemento <articolo>. Questo costringerebbe a dover strutturare un documento XML in funzione della sua formattazione con CSS, in evidente **contraddizione** con lo stesso concetto di foglio di stile che tende a separare la definizione dei dati dalla sua presentazione.

Inoltre, non sempre un documento XML descrive qualcosa di visualizzabile con un browser. Ad esempio, è possibile definire un linguaggio XML per la descrizione di immagini o di ambienti virtuali. In questo caso i CSS non sono di alcun aiuto nella presentazione del documento.

Per risolvere questi problemi il W3C ha definito un'insieme di specifiche volte a gestire in maniera altamente flessibile la presentazione e la trasformazione di documenti XML: l'**eXtensible Stylesheet Language (XSL)**.

XSL: eXtensible Stylesheet Language

L'eXtensible Stylesheet Language (XSL) è un **insieme di tre linguaggi** che forniscono gli strumenti per l'elaborazione e la presentazione di documenti XML in maniera molto flessibile.

La definizione di questa tecnologia si basa sull'osservazione del processo di **presentazione di dati** di qualsiasi natura. Infatti, in un tale processo possiamo individuare i seguenti meccanismi di base:

- un meccanismo per l'**individuazione dei dati** da presentare
- un meccanismo per il **controllo dell'elaborazione** dei dati e di come la presentazione deve essere effettuata
- un meccanismo per la definizione della **formattazione da applicare** ai dati per la presentazione vera e propria

A ciascuno di questi tre meccanismi, XSL associa uno specifico linguaggio:

- **XPath** consente di individuare gli elementi e gli attributi di un documento XML sui quali verranno applicate le operazioni necessarie per la presentazione dei dati
- **XSLT** (XSL transformation) consente di controllare le operazioni che rendono i dati presentabili
- **XSL-FO** (XSL Formatting Objects) definisce un insieme di tag di formattazione

Questa **suddivisione dei compiti** nel processo di presentazione è il punto di forza di XSL e ne garantisce la flessibilità. Infatti, questi tre linguaggi non sono strettamente dipendenti l'uno dall'altro.

Se, ad esempio, in una particolare applicazione ci rendiamo conto che XPath non soddisfa le nostre esigenze di ricerca di elementi in un documento XML, potremmo utilizzare linguaggio analogo (XQL, per citarne uno) ma senza modificare la presentazione dei caratteri.

Oppure potremmo decidere di non utilizzare affatto XSL-FO per formattare i dati di un documento XML e produrre direttamente codice HTML, o meglio XHTML, cioè la versione di HTML basata su XML.

La **presentazione dei dati** racchiusi in un documento XML è basata su due elementi:

- un documento che descrive come i dati devono essere elaborati per la presentazione, chiamato **foglio di stile XSLT**
- un componente software, chiamato **processore XSLT**, in grado di prendere in input un documento XML e un foglio di stile XSLT e di produrre in output i dati secondo il formato di presentazione prescelto (XSL-FO, XHTML, testo, ecc.)

La definizione di un foglio di stile XSLT è quindi il punto cruciale della presentazione dei dati XML. Le **regole di trasformazione** presenti in un foglio XSLT consentono di selezionare gli elementi di un documento XML e di gestirne la modalità di presentazione. Come abbiamo giù detto, nella definizione standard di XSLT la selezione degli elementi da presentare viene fatta tramite il linguaggio XPath.

XPath: espressioni e funzioni

Individuare gli elementi di un documento XML rappresenta il primo passo di un'**elaborazione** per la presentazione dei dati. Nei CSS questo ruolo è svolto dal selettore, cioè l'elemento sintattico di una regola CSS che individua gli elementi da formattare.

In XSL questo passo viene descritto tramite il **linguaggio XPath**. A differenza dei selettori CSS, però, XPath è molto più potente e flessibile.

Questo linguaggio consente di creare espressioni dichiarative, chiamate espressioni XPath o **pattern**, che individuano i vari nodi dell'albero di rappresentazione di un documento XML.

La sua sintassi è molto compatta e, per certi versi, ricorda un po' le espressioni per **individuare il percorso** di un file o una cartella su un file system.

Il simbolo '/' rappresenta il root element di un documento XML. facendo riferimento al seguente documento XML, l'espressione '/' rappresenta l'elemento articolo:

```
<?xml version="1.0" ?>
<articolo titolo="">
 <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>
</articolo>
```

Per individuare l'elemento <testo> all'interno del nostro esempio di documento XML dobbiamo specificare un percorso di questo tipo:

/articolo/paragrafo/testo

Se siamo interessati all'attributo titolo dell'elemento <paragrafo> possiamo specificarlo nel modo seguente:

/articolo/paragrafo/@titolo

Queste espressioni, però, individuano il primo elemento che corrisponde al percorso. Per **selezionare uno specifico elemento**, ad esempio il paragrafo con titolo Titolo del secondo paragrafo, possiamo farlo con l'espressione seguente:

/articolo/paragrafo/[@titolo=ìTitolo del secondo paragrafoì]

In pratica, all'interno delle parentesi quadre specifichiamo la **condizione** che deve essere soddisfatta dall'elemento.

Per selezionare un elemento specifico è possibile utilizzare anche alcune **funzioni** predefinite, come **position()**, che specifica la posizione di un elemento, e **last()**, che specifica l'ultima posizione di una sequenza di elementi. Ad esempio, le due espressioni seguenti specificano rispettivamente il secondo paragrafo e l'ultimo paragrafo dell'articolo:

/articolo/paragrafo[position()=2]
/articolo/paragrafo[position()=last()]

Gli esempi di pattern che abbiamo visto rappresentano percorsi assoluti, cioè individuano un elemento partendo dall'elemento root. Un importante concetto è quello di **nodo corrente**, cioè il nodo che si è appena individuato. Il concetto è analogo a quello di cartella corrente per i file system.

Il nodo corrente viene indicato con il punto '**.**', mentre il nodo genitore del nodo corrente viene indicato con i due punti '**..**' . La doppia barra '//' consente di individuare tutti i discendenti di un particolare elemento. Ad esempio, l'espressione

/articolo/paragrafo//immagine

individua tutti gli elementi <immagine> contenuti nell'elemento <paragrafo>, a qualsiasi livello.

Per individuare tutti i sottoelementi di un certo elemento possiamo utilizzare l'**asterisco (*)**. Ad esempio, per selezionare tutti gli elementi del secondo paragrafo possiamo scrivere la seguente espressione XPath:

/articolo/paragrafo[position()=2]/

Esistono numerose altre possibilità per **costruire espressioni** di ricerca, soprattutto sfruttando le varie funzioni predefinite che XPath mette a disposizione.

Abbiamo visto negli esempi precedenti le funzioni position() e last() che restituiscono rispettivamente la posizione dell'elemento corrente e la posizione dell'ultimo elemento. Un'altra funzione che lavora sugli elementi è **count()**, che restituisce il numero di elementi relativi all'espressione passata come argomento.

Ad esempio, count(//paragrafo) restituisce il numero di elementi paragrafo nel documento corrente.

Altre funzioni risultano molto utili in situazioni diverse. Ad esempio, sono previste funzioni di manipolazione di stringhe, come **concat()** che concatena le stringhe passate come parametro. Per esempio, concat("par", "agra", "fo") restituisce la stringa "paragrafo".

La funzione **substring()** restituisce una sottostringa in base ai parametri specificati. Ad esempio, substring("paragrafo", 5, 3) restituisce la sottostringa "gra", poiché estrae tre caratteri iniziando dal quinto carattere della stringa "paragrafo".

La funzione **string-length()** restituisce il numero di caratteri che compongono la stringa specificata come parametro.

Un'altra utile funzione è **starts-with()** che restituisce true se il primo parametro di tipo stringa inizia con la stringa passata come secondo parametro. Ad esempio, starts-with("paragrafo", "par") restituisce true. Analogamente esiste la funzione **ends-with()** che valuta la parte finale di una stringa.

Altre funzioni consentono di manipolare **espressioni numeriche**. Ad esempio, la funzione **number()** converte il valore passato come parametro in un numero, mentre **round()** arrotonda un numero all'intero più vicino, **floor()** restituisce l'intero uguale o precedente al parametro passato e **ceiling()** restituisce l'intero uguale o successivo.

Queste ed altre funzioni definite dalle specifiche di XPath contribuiscono ad ottenere espressioni molto potenti.

Fogli di stile XSLT

Nell'ambito dello standard XSL il compito di **trasformare un documento XML** in un altro documento è affidato al linguaggio XSLT.

Nella terminologia di XSLT, il documento da trasformare è chiamato **documento origine** (source document), mentre il documento generato dal processo di trasformazione è chiamato **documento risultante** (result document).

Il documento risultante di una trasformazione XSLT può essere un documento XML o un documento di altro tipo. Ad esempio, a partire da un documento XML possiamo generare un documento XHTML oppure un documento WML o anche RTF o altri formati testuali.

La trasformazione avviene in base alle informazioni contenute in un particolare tipo di documento e interpretate da un processore XSLT. Questo documento di trasformazione, chiamato **foglio di stile XSLT**, non è altro che un documento XML che fa uso di tag appartenenti alla grammatica di XSLT in grado di controllare il processo di trasformazione.

In generale, un foglio di stile XSLT ha la seguente **struttura**:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/transorm">
... template di trasformazione ...
</xsl:stylesheet>
```

All'interno del root element **<xsl:stylesheet>** vengono definite le istruzioni di trasformazione da applicare al documento XML origine. XSLT consente di trasformare ciascun elemento del documento XML di origine in un altro elemento del formato del documento risultante.

È possibile anche aggiungere al documento risultante elementi completamente nuovi o non prendere in considerazione determinati elementi del documento origine, riordinare gli elementi, fare elaborazioni in base al risultato di determinate condizioni, ecc.

Il **processo di trasformazione** controllato da XSLT si basa sull'uso del linguaggio XPath per individuare gli elementi del documento origine, sui quali viene applicato un template (o modello) di trasformazione.

Da un punto di vista sintattico, un **template** è un elemento del linguaggio XSLT che ha la seguente forma di base:

```
<xsl:template match="espressione XPath">
... Definizione dellìoutput...
</xsl:template>
```

Un template XSLT individua un elemento o un insieme di elementi di un documento XML sfruttando unìespressione XPath e vi applica una serie di elaborazioni per ottenere un output. Per fare una analogia con i CSS, un template corrisponde approssimativamente ad una regola CSS.

Per fare un esempio, facendo riferimento al documento XML che descrive un articolo, il seguente template restituisce in output il titolo del secondo paragrafo:

```
<xsl:template match="/articolo/paragrafo[position()=2]">
 <xsl:value-of select="@titolo"/>
</xsl:template>
```

L'elemento <xsl:value-of> contenuto all'interno dell'elemento <xsl:template> consente di catturare il valore di un elemento o di un attributo e di produrlo in output.

Un **foglio di stile XSLT** non è altro che un insieme di template da applicare ai vari elementi di un documento XML. Comprendere come avviene l'applicazione dei template di un foglio di stile XSLT da parte del relativo processore è un elemento essenziale per realizzare fogli di stile corretti.

Il modello di elaborazione dei **processori XSLT** segue la struttura gerarchica del documento XML di partenza. In pratica, il processore XSLT va alla ricerca dei template da applicare partendo dal root element e seguendo l'albero di rappresentazione logica del documento stesso.

In linea di principio, con l'applicazione del relativo template al root element termina il compito del processore XSLT. Se il foglio di stile contiene diversi template che devono essere applicati ai vari elementi del documento XML di partenza è necessario specificarlo esplicitamente.

Questo può essere fatto specificando all'interno del template corrente l'elemento **<xsl:apply-templates/>**. Con questo elemento del linguaggio XSLT indichiamo al processore di cercare eventuali altri template da applicare alla trasformazione corrente.

Non è importante l'ordine con cui sono stati specificati i template all'interno del foglio di stile; sarà il processore ad individuarli in base all'espressione XPath associata.

Ad esempio, un possibile template del root element che avvia la trasformazione di un articolo può essere analogo al seguente:

```
<xsl:template match="/">
 <xsl:value-of select="articolo/@titolo"/>
 <xsl:apply-templates/>
</xsl:template>
```

Questo template estrae il titolo dell'articolo ed indica al processore XSLT di cercare altri template da applicare al documento XML.

Elaborazioni con XSLT

XSLT è un linguaggio potente e flessibile con costrutti analoghi a quelli dei linguaggi di programmazione. Infatti permette di **controllare il flusso** dell'elaborazione per produrre un output sulla base del verificarsi di determinate condizioni e di effettuare delle iterazioni.

facendo riferimento al documento XML che rappresenta un articolo, potremmo voler produrre una formattazione del titolo diversa in base al tipo di paragrafo. Ad **esempio**, in corrispondenza del paragrafo di tipo bibliografia potremmo voler evidenziare in output la scritta Bibliografia.

Per ottenere questo risultato possiamo utilizzare l'elemento **<xsl:if>** di XSLT il cui comportamento è analogo all'istruzione if dei linguaggi di programmazione tradizionali. Possiamo quindi sfruttare questo elemento come nel seguente codice:

```
<xsl:template match="paragrafo">
  <h3>
  <xsl:if test="@tipo=bibliografia"'>
    Bibliografia
  </xsl:if>
  <xsl:value-of select="@titolo"/>
  </h3>
  <xsl:apply-templates/>
</xsl:template>
```

L'attributo test dell'elemento <xsl:if> specifica la **condizione da valutare** e può essere una condizione booleana o un'espressione XPath che individua un insieme di nodi. In quest'ultimo caso la condizione viene considerata vera se l'insieme dei nodi individuati non è vuoto.

L'elemento <xsl:if> prevede **una sola condizione**, anche se composta tramite operatori booleani, e non prevede alternative nel caso in cui la condizione non è vera.

Per ottenere diversi output in base a diverse condizioni possiamo sfruttare l'**<xsl:choose>**, la cui semantica corrisponde alle istruzioni di tipo select case o **switch** dei linguaggi di programmazione.

Per estendere l'esempio visto prima a proposito dell'elemento <xsl:if> possiamo scrivere il seguente template:

```
<xsl:template match="paragrafo">
  <h3>
  <xsl:choose>
    <xsl:when test="@tipo='abstract">
      Abstract dell&apos;articolo
    </xsl:when>
    <xsl:when test="@tipo='bibliografia">
      Bibliografia
    </xsl:when>
    <xsl:when test="@tipo='note">
      Note finali
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="@titolo"/>
```

```
    </xsl:otherwise>
  </xsl:choose>
  </h3>
  <xsl:apply-templates/>
</xsl:template>
```

All'interno dell'elemento <xsl:choose>, le condizioni da valutare vengono specificate tramite gli elementi **<xsl:when>**. L'elemento **<xsl:otherwise>** indica l'output da generare nel caso in cui nessuna delle condizioni precedenti sia vera.

Nel caso specifico, in presenza di ciascun tipo di paragrafo viene generato un titolo predefinito, mentre se l'elemento individuato dal template non ha associato alcun tipo viene generato il titolo corrispondente all'omonimo attributo.

Abbiamo visto come l'**applicazione dei template** ai vari nodi di un documento XML è automaticamente ricorsiva, cioè il processore XSLT va automaticamente alla ricerca di tutti i nodi del documento origine che corrispondono ad un template.

Se vogliamo in qualche modo analizzare il contenuto dei diversi nodi all'interno di un template possiamo fare ricorso all'elemento **<xsl:for-each>**, il cui funzionamento è analogo alle istruzioni for dei linguaggi di programmazione tradizionali.

Vediamo come sfruttare l'elemento <xsl:for-each> nel nostro caso. Immaginiamo di voler produrre in output nel documento XHTML risultante un sommario iniziale in cui vengono elencati i titoli dei paragrafi dell'articolo. Possiamo scrivere il template di elaborazione dell'elemento <articolo> nel seguente modo:

```
<xsl:template match="/">
  <h1><xsl:value-of select="articolo/@titolo"/></h1>
  <h2>Sommario</h2>
  <xsl:for-each select="paragrafo">
    <h3><xsl:value-of select="@titolo"/></h3>
  </xsl:for-each>
  <xsl:apply-templates/>
</xsl:template>
```

Dopo aver prodotto in output il titolo dell'articolo, tramite l'elemento <xsl:for-each> individuiamo tutti gli elementi <paragrafo> contenuti nell'elemento corrente (<articolo>) e ne estraiamo il titolo.

Ci sono situazioni in cui l'applicazione di un foglio di stile XSLT deve **produrre un documento XML** ed in particolare tag relativi ad elementi XML.

Supponiamo, ad esempio, di dover generare del codice XHTML a partire dall nostro articolo in XML, ed in particolare dobbiamo porre l'elemento <img> in corrispondenza dell'elemento <immagine>.

La cosa a prima vista non sembrerebbe così complicata. La prima cosa che potrebbe venire in mente potrebbe essere l'utilizzo di un template analogo al seguente:

```
<xsl:template match="immagine">
  <img src="<xsl:value-of select='@file'/>" />
</xsl:template>
```

Tuttavia, dal momento che si tenta di inserire un elemento all'interno del tag di un altro elemento, l'espressione non risulta ben formata ed il processore XSLT darà un messaggio d'errore.

Per fortuna XSLT prevede la possibilità di generare elementi e relativi attributi da inserire nel documento risultante. Questa possibilità si basa sull'uso degli elementi <xsl:element> e <xsl:attribute>. Nel nostro caso li utilizziamo nel seguente modo:

```
<xsl:template match="immagine">
  <xsl:element name="img">
    <xsl:attribute name="src">
     <xsl:value-of select="@file"/>
    </xsl:attribute>
  </xsl:element>
</xsl:template>
```

Il risultato dell'applicazione di questo template sarà un elemento <img> il cui attributo src avrà il valore del corrispondente attributo file del nostro elemento <immagine>.

L'applicazione di un foglio di stile XSLT può essere fatta esternamente, cioè avviando un processore XSLT e indicando il documento origine e il foglio da applicare, oppure internamente, cioè specificando il foglio da applicare all'interno del documento XML stesso.

In quest'ultimo caso il processore XSLT individuerà automaticamente il foglio di stile da applicare. Per assegnare un foglio di stile XSLT ad un documento XML occorre specificare la seguente dichiarazione prima dell'elemento radice:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="stile.xslt" ?>
```

La maggior parte dei processori XSLT è in grado di interpretare la direttiva ed applicare il relativo foglio di stile XSLT al documento XML corrente.

Anche la maggior parte dei browser recenti sono in grado di applicare un foglio di stile XSLT ad un documento XML. In particolare, i browser Microsoft Internet Explorer 6.0, Netscape 6.0, Mozilla 1.4, Firefox 0.9.

Conclusioni

Questa guida ha voluto fornire una panoramica su XML e sulle tecnologie collegate a questo meta-linguaggio. Ovviamente gli argomenti trattati meritano maggiori approfondimenti, anche in base alle esigenze di ciascun lettore e di ciascun progetto che intende coinvolgere XML. La prima appendice di questa guida riporta alcuni riferimenti per approfondire i diversi argomenti trattati.

Dalla nascita di XML ad oggi si sono fatti notevoli progressi, non soltanto sul fronte della definizione di linguaggi e sulla loro standardizzazione, ma anche sul fronte delle implementazioni.

Numerosi sono ormai gli strumenti che consentono di lavorare con XML, implicitamente ed esplicitamente. Una delle appendici elenca un piccolo sottoinsieme di software per XML.

In campo industriale e commerciale, inoltre, XML è stato bene accettato, tanto che sono stati definiti diversi linguaggi basati su questa tecnologia e focalizzati negli ambiti più disparati. Alcuni di questi linguaggi sono sinteticamente descritti in una delle appendici che seguono.

Il proliferare di questi linguaggi pone però il problema della loro **duplicazione**, cioè il rischio di ridefinire in un ambito specifico linguaggi che qualcun altro ha già definito.

A questo scopo sono sorti diversi consorzi e **comunità per la standardizzazione** e la diffusione di linguaggi basati su XML ed utilizzabili in ambito industriale e commerciale. tra i consorzi più noti citiamo OASIS (www.oasis.org), che fornisce un database di XML Schema e Dtd di linguaggi negli ambiti più diversi.

In conclusione di questo viaggio nel mondo XML, riassumiamo brevemente i vantaggi offerti da questa tecnologia nei diversi ambiti di applicazione:

standardizzazione
XML e le tecnologie correlate sono state definite come standard pubblici di riferimento; questo ne favorisce la diffusione e ne incoraggia l'adozione

apertura
XML non è legato ad un produttore o ad un ambito ristretto; esso è abbastanza generale ed è utilizzabile in diversi ambiti senza alcun legame con una specifica applicazione o uno specifico produttore

indipendenza dalla piattaforma
Come conseguenza della standardizzazione e della apertura, XML è indipendente da una specifica piattaforma hardware e software; questo consente il riutilizzo della tecnologia in diversi ambiti tecnologici e una semplificazione nello scambio di dati strutturati

esistenza di strumenti standard
Altra conseguenza della standardizzazione e dell'apertura di XML è costituita dalla larga diffusione di strumenti standard per le elaborazioni di base: validazione rispetto ad una grammatica e trasformazione dei documenti XML

Alla luce di quanto detto finora possiamo dire che XML è ormai una tecnologia matura che inizia a dare i suoi frutti. Fino a qualche anno fa si diceva che XML avrebbe influenzato il futuro dell'informatica nei suoi diversi ambiti. Oggi possiamo dire che questo futuro è arrivato.

Appendice A

MathML
MathML è un'applicazione XML per la descrizione di formule matematiche. Una formula MathML può essere visualizzata all'interno di una pagina Web o in altri tipi di documento, ma può anche essere valutata da opportuni software per effettuare dei calcoli. Le specifiche del linguaggio sono state definite dal W3C e sono consultabili a partire dall'indirizzo http://www.w3.org/Math.

Open Financial Exchange (OFX)
Open Financial Exchange (OFX) è un'applicazione XML utilizzata per lo scambio di dati a carattere finanziario tra software specifici per banche ed istituti finanziari. Le specifiche sono mantenute da un consorzio di società private e sono consultabili a partire dall'indirizzo http://www.ofx.net/ofx/de_spec.asp.

Resource Description Framework (RDF)
RDF (Resource Description Framework) è un linguaggio per la rappresentazione di informazioni su risorse di qualsiasi tipo (siti Web, articoli, libri, persone, ecc.). In senso stretto, questo linguaggio sarebbe un linguaggio astratto, cioè un insieme di elementi e regole per la descrizione di risorse. XML costituisce una possibile sintassi utilizzabile per descrivere risorse secondo il framework RDF. In alternativa potrebbero essere utilizzate altre sintassi, come ad esempio dei grafi. Le specifiche di RDF sono mantenute dal W3C a partire dall'indirizzo http://www.w3.org/RDF.

RSS
Con la sigla RSS si intendono in realtà due linguaggi basati su XML che hanno l'obiettivo di fornire un supporto per la distribuzione di contenuti sul Web, come ad esempio notizie, link, prodotti, ecc. tramite questi due linguaggi è possibile pubblicare notizie su diversi siti oppure gestire e consultare i contenuti tramite appositi programmi detti news aggregator. I due linguaggi hanno origine diversa, pur avendo lo stesso scopo: RDF Site Summary è stato definito dal W3C nell'ambito del progetto RDF, mentre Really Simple Syndication è stato inizialmente definito da Netscape e poi sviluppato Userland.

Scalable Vector Graphics (SVG)
SVG (Scalable Vector Graphics) è un linguaggio per la descrizione di immagini vettoriali bidimensionali pensato principalmente per il Web. Le specifiche sono state definite dal W3C e sono accessibili a partire dall'indirizzo http://www.w3.org/Graphics/SVG.

Simple Object Access Protocol (SOAP)
SOAP (Simple Object Access Protocol) è un protocollo per l'invocazione remota di procedure. La funzione di SOAP è analoga a quella di XML-RPC, ma a differenza di quest'ultimo ha diverse caratteristiche extra, come ad esempio la possibilità di effettuare chiamate sincrone o asincrone, un'ampia possibilità di definire tipi di dato, una maggiore flessibilità nella gestione del trasporto delle richieste, ecc. Le specifiche sono pubblicate dal W3C a partire dall'indirizzo http://www.w3.org/2000/xp/Group.

Synchronized Multimedia Integration Language (SMIL)
SMIL (Synchronized Multimedia Integration Language) è un linguaggio basato su XML che consente di descrivere le modalità di integrazione di materiale multimediale (video, audio, immagini, testo, ecc.) per la realizzazione di presentazioni multimediali. Il linguaggio consente di indicare effetti di transizione, di specificare le modalità di sincronizzazione tra audio e video, di gestire l'avanzamento del tempo, ecc. Il linguaggio è stato definito dal W3C e le specifiche si possono trovare a partire dall'indirizzo http://www.w3.org/AudioVideo.

VoiceXML
VoiceXML è un linguaggio per la descrizione vocale di testo, cioè testo destinato a sintetizzatori vocali ed utilizzabili da segreterie telefoniche, risponditori automatici, servizi di telefonia automatica, ecc. Le specifiche sono mantenute dal W3C e possono essere consultate a partire dall'indirizzo http://www.w3.org/Voice.

Wireless Markup Language (WML)
WML (Wireless Markup Language) è un linguaggio per la descrizione di pagine Web da visualizzare su dispositivi wireless, quindi in genere a bassa banda di trasmissione, con schermi di dimensioni ridotte, con ridotte possibilità interattive. WML è utilizzato nellìambito del Wireless Application Protocol (WAP), lo standard per la navigazione su Internet tramite dispositivi wireless. Le specifiche sono mantenute dal Wap Forum (http://www.wapforum.org).

XML Remote Procedure Call (XML-RPC)
XML-RPC (XML Remote Procedure Call) è un protocollo per l'invocazione remota di procedure (ad esempio, tramite Internet) indipendentemente dall'ambiente operativo ed il linguaggio di programmazione utilizzato. XML-RPC usa HTTP come protocollo di trasporto ed un linguaggio XML-based per la descrizione delle richieste e dei risultati. Le specifiche sono reperibili all'indirizzo http://www.xmlrpc.com.

XML Web Services
Gli XML Web services, o più semplicemente Web services, rappresentano un insieme di tecnologie che consentono l'interoperabilità tra applicazioni su una rete indipendentemente dal linguaggio di programmazione e dalla piattaforma. tramite queste tecnologie un'applicazione può esporre un'API (Application Programming Interface) richiamabile tramite un protocollo come HTTP. I Web service prevedono la possibilità di descrivere l'API messa a disposizione tramite un linguaggio XML-based chiamato WSDL (Web Service Description Language) ed utilizzano SOAP o XML-RPC per le chiamate remote. Le specifiche generali sono mantenute dal W3C (http://www.w3.org/2002/ws/).

Appendice B

Editor

- Authentic (www.altova.com/cust_authentic_overview.html)
- Cooktop (www.xmlcooktop.com)
- EditML Pro (www.netbryx.com)
- oXygen (www.oxygenxml.com)
- XML Pro (www.vervet.com)
- XMLWriter (www.xmlwriter.net)

Parser

- MSXML (www.microsoft.com/downloads)
- Xerces (xml.apache.org/#xerces)
- XML4J (www.alphaworks.ibm.com/tech/xml4j)

Processori ed editor XSLT

- Saxon (saxon.sourceforge.net)
- Visual XSLT (www.activestate.com/Products/Visual_XSLT)
- Xalan (xml.apache.org/xalan-j)
- Xselerator (www.marrowsoft.com/Products.htm)

Ambienti integrati

- Stylus Studio (www.stylusstudio.com)
- XMLSpy (www.xmlspy.com)