Capitolo 3

Il linguaggio XML

3.1 Introduzione al linguaggio XML

Il linguaggio XML (eXtensible Markup Language) è stato sviluppato nel 1996 da XML Working Group, gruppo di lavoro organizzato dal World Wide Web Consortium (W3C) [3-1]. In pochi anni ha conosciuto un vasto e crescente successo per la sua semplicità e flessibilità.

Concepito come evoluzione del metalinguaggio SGML (Standard Generalized Markup Language), esso consente di creare linguaggi personalizzati di markup superandone il principale limite, costituito dalla complessità e pesantezza di utilizzo.

Il suo obiettivo originario era quello di potenziare lo scambio di informazioni nel Web e permettere un'agevole ed efficiente pubblicazione di dati arbitrariamente complessi. Le sue caratteristiche, tuttavia, fanno sì che si presti ad una larga varietà di applicazioni, tra le quali la creazione di protocolli specifici per il management di dati e lo scambio di informazioni

tra software eterogenei.
Pertanto, dopo essere stato inizialmente adottato nel mondo del Web (ad esempio per il commercio elettronico), ha incontrato consensi anche per lo sviluppo e l'integrazione di applicazioni specifiche, come dimostrano recenti proposte in cui viene utilizzato per il network management.
Nel progetto presentato in questa tesi il linguaggio XML ha trovato applicazione nella definizione di protocolli di comunicazione, come descritto in seguito nel paragrafo 3.5.

3.1.1 Sintassi del linguaggio XML

Un documento XML è composto da determinate parti costitutive, di cui alcune opzionali, che seguono semplici regole sintattiche. Di seguito viene effettuata una introduzione alla sintassi del linguaggio nelle sue varie parti; il lettore interessato ad ulteriori approfondimenti può consultare [3-2].

Prologo

È un componente opzionale, costituito dalla dichiarazione XML e/o dalla dichiarazione del tipo di documento. La prima identifica la versione delle specifiche del linguaggio XML alla quale si riferisce il documento. La seconda indica le regole al quale il documento deve essere conforme, in pertinenza agli elementi presenti, al loro contenuto, alla loro disposizione ed al loro annidamento.
Sebbene facoltativo, è opportuno inserire il prologo in documenti destinati a pubblico utilizzo o condivisione, poiché costituisce un utile riferimento per chi deve servirsi di essi.
La dichiarazione XML può contenere, oltre alla versione del linguaggio

(attributo *version*), anche la dichiarazione di codifica (*encoding*), che specifica lo schema di codifica dei caratteri, e quella di documento autonomo (*standalone*), che indica l'esistenza o meno di una dichiarazione del tipo di documento esterna al documento stesso. Un esempio di dichiarazione XML è il seguente:

```
<?xml version="1.0" encoding="UTF8" standalone="no"?>
```

La dichiarazione del tipo di documento può contenere il codice per la definizione del tipo di documento oppure indirizzare ad un file esterno dove esso sia presente, come è il caso del seguente esempio:

```
<!DOCTYPE Rubrica SYSTEM "rubr.dtd">
```

Entità

Le entità costituiscono un'unità di memorizzazione per il documento XML. Una volta dichiarata, un'entità può essere richiamata più volte all'interno del documento semplicemente riferendosi al suo nome, ed in fase di interpretazione del documento XML tale riferimento sarà sostituito dal contenuto dell'entità stessa. Per includere un riferimento si inserisce il nome dell'entità tra una "e commerciale" (&) ed un punto e virgola.

Ogni entità deve essere dichiarata con un nome univoco, ed il suo contenuto può essere di diversa natura, da una stringa di caratteri ad un file di tipo immagine di grandi dimensioni. Nel caso l'entità sia non analizzabile, ossia non interpretabile da un elaboratore di documenti XML, è necessario fornire un'ulteriore dichiarazione, detta annotazione, in cui si indica all'elaboratore quale applicazione software utilizzare per analizzarla.

Esistono poi entità cosiddette predefinite, che consistono in un modo alternativo di scrivere i caratteri utilizzati per il markup affinché possano essere interpretati come semplici dati e non come parte del linguaggio:

- `<` < (parentesi angolare di apertura)
- `>` > (parentesi angolare di chiusura)
- `&` & (e commerciale)
- `&apos;` ' (apostrofo)
- `&quot;` " (virgolette doppie)

Elementi

Si indica con elemento l'insieme di uno specifico tag di apertura, il suo tag di chiusura corrispondente ed il contenuto compreso tra i due, che può essere costituito da dati, altri elementi o entrambe le cose.

Il tag di apertura consiste nel nome dell'elemento racchiuso tra parentesi angolari; quello di chiusura contiene rispetto a quello di apertura un carattere "/" (*slash*) prima del nome dell'elemento.

XML è *case sensitive*, quindi l'elemento `TEST1` è diverso dall'elemento `Test1`. I nomi degli elementi devono essere univoci, quindi non possono essere dichiarati due elementi con lo stesso nome (anche se è possibile utilizzare lo stesso elemento più volte).

I dati di un elemento sono compresi tra i tag di apertura e di chiusura; ogni tag di apertura deve obbligatoriamente avere il corrispettivo di chiusura, ad eccezione dei tag di elemento vuoto.

I tag di elemento vuoto non necessitano di tag di chiusura e contengono un carattere "/" dopo il nome dell'elemento. L'utilità principale degli elementi vuoti è l'indicare – con maggiore concisione rispetto ad un elemento non vuoto – un'informazione booleana come la presenza/assenza di un certo dato.

Annidando opportunamente diversi elementi si possono conseguire complesse interrelazioni logiche e strutturali tra i medesimi. Al livello più

esterno deve esserci un solo elemento.

I commenti utilizzano la seguente sintassi:

```
<!-- Testo del commento -->
```

Un esempio di semplice documento XML è il seguente:

```
<!-- documento di esempio -->
<!-- calendario mie partite basket -->
<?xml version="1.0"?>
<CALENDARIO_PARTITE>
        <GIORNATA>
                <DATA>18 marzo 2004</DATA>
                <LUOGO>Palestra 1</LUOGO>
                <ORA>21:30</ORA>
                <PARTECIPANTI>
                        <SQUADRA_1>
                                <NOME>Giocatore 1</NOME>
                                <NOME>Giocatore 2</NOME>
                                <NOME>Giocatore 3</NOME>
                                <NOME>Giocatore 4</NOME>
                                <NOME>Giocatore 5</NOME>
                        </SQUADRA_1>
                        <SQUADRA_2>
                                <NOME>Giocatore 1</NOME>
                                <NOME>Giocatore 2</NOME>
                                <NOME>Giocatore 3</NOME>
                                <NOME>Giocatore 4</NOME>
                                <NOME>Giocatore 5</NOME>
                        </SQUADRA_2>
                </PARTECIPANTI>
                <RISULTATO>101 – 89</RISULTATO>
                <!-- il seguente elemento vuoto è presente in
                caso di -->
                <!-- vittoria della mia squadra, assente in caso
                di sconfitta -->
                <VITTORIA/>
        </GIORNATA>
        <!-- qui potrebbero seguire altri elementi GIORNATA...
        -->
</CALENDARIO_PARTITE>
```

Attributi

Gli attributi sono uno strumento per introdurre informazioni aggiuntive all'interno di un elemento. Essi non influiscono sul contenuto dell'elemento stesso, poiché si trovano all'interno dei tag di apertura o in un tag di elemento vuoto. Il valore che assume l'attributo è racchiuso tra virgolette.
L'utilizzo degli attributi è solitamente sconsigliato in favore di un maggiore utilizzo degli elementi (più flessibili e significativi), per ragioni sia di chiarezza e leggibilità del documento, ma soprattutto per la maggiore semplicità nel recuperare, con gli strumenti attualmente disponibili, il contenuto di un elemento rispetto al valore di un attributo. A tale proposito si veda il paragrafo 3.5.1.
Nell'esempio seguente si aggiunge l'attributo `numero` all'elemento `GIORNATA` dell'esempio precedente.

```
<!-- documento di esempio -->
<?xml version="1.0"?>
<CALENDARIO_PARTITE>
        <GIORNATA numero="1">
        ...
        </GIORNATA>
</CALENDARIO_PARTITE>
```

3.1.2 Vantaggi del linguaggio XML

Come è possibile vedere nel precedente paragrafo, la sintassi del linguaggio XML è semplice ed intuitiva; tuttavia, questo non è il suo unico pregio. I più importanti ed ampiamente riconosciuti vantaggi di XML sono qui elencati:

- **XML è agevole da apprendere e da utilizzare**: non occorrono particolari competenze per familiarizzare con il linguaggio, e la progettazione con XML risulta facile, rapida e concisa
- **Un documento XML consiste in semplice testo**: è auto-descrittivo e può essere generato o visualizzato con strumenti di larga diffusione e facile utilizzo come editor di testo. Nel caso di un protocollo di comunicazione XML client-server, si possono effettuare test di funzionalità del server semplicemente immettendo del testo nella connessione tramite telnet o simili client testuali
- **XML è universale**: come il suo predecessore SGML, XML è non proprietario ed indipendente dalla piattaforma
- **XML ha una vasta disponibilità di librerie per tutti i linguaggi di programmazione**: la fase di sviluppo di un progetto che utilizza XML risulta facilitata dalla ampia e diffusa documentazione e dalle molteplici API (Application Program Interface) accessibili per il parsing, la formattazione, l'esplorazione e la convalida di documenti, indipendentemente dal linguaggio di programmazione utilizzato
- **XML consente di convalidare un documento secondo un modello precostituito**: esistono due approcci per la definizione del modello di documento; a tal proposito si veda il paragrafo 3.2
- **XML è corredato da una serie di strumenti che ne espandono ulteriormente le potenzialità**: nel paragrafo 3.3 si effettua una rassegna dei più utili e diffusi standard nati intorno al linguaggio
- **XML garantisce la possibilità di estendere, aggiornare o modificare un protocollo senza perdere compatibilità**: è possibile incrementare le funzionalità di un protocollo senza dover aggiornare le applicazioni che già lo utilizzano, introducendo se necessario nuovi elementi per le nuove strutture dati richieste. Le applicazioni costruite sul vecchio

modello continueranno ad operare correttamente, poiché ignoreranno questi nuovi elementi. Introducendo un attributo o un elemento che indica la versione del protocollo, è inoltre possibile per un'applicazione capire se un documento è stato redatto secondo una nuova versione e prendere provvedimenti.

3.2 Definizione del modello di documento

Un aspetto di fondamentale importanza nel linguaggio XML è la possibilità di definire una tipologia di documento personalizzata e conseguentemente controllare se un dato documento è conforme o meno a tale tipologia. Qualora XML sia utilizzato in un protocollo per lo scambio di informazioni, questa sua caratteristica consente di effettuare una semplice ed immediata convalida di un messaggio ricevuto.
Esistono due metodi, ciascuno con i propri vantaggi, per la definizione del tipo di documento: Document Type Definition e XML Schema. Sono degli standard creati con il patrocinio del W3C come XML, ed in questo paragrafo saranno esaminati entrambi.

3.2.1 Document Type Definition (DTD)

Dei due approcci, quello della DTD è sicuramente il più semplice e diffuso (sono infatti molte le applicazioni che si possono trovare a supporto di questo strumento) ed è certamente appropriato per la maggior parte dei possibili utilizzi di XML, anche se presenta dei limiti nelle potenzialità.

Tramite la DTD è possibile specificare un modello di documento XML ed

effettuare la convalida dei documenti rispetto al modello definito. Un documento XML *valido* ottempera senza eccezioni a tutte le regole in tal modo sancite (oltre naturalmente a quelle sintattiche del linguaggio XML). La creazione della DTD rappresenta dunque una fase delicata ed importante della progettazione di un eventuale protocollo che utilizzi XML per lo scambio di informazioni.

Nella DTD si determina la struttura del documento: quali e quanti elementi vi debbano comparire, in quale ordine, il loro contenuto e le relazioni gerarchiche tra di essi, la presenza o meno di entità nel documento o attributi all'interno degli elementi.

Il limite fondamentale della Document Type Definition è quello di trattare il contenuto degli elementi esclusivamente come stringhe di caratteri, analizzabili dall'elaboratore XML oppure no (da notare che se un elemento deve poter contenere altri elementi, il suo contenuto deve essere definito analizzabile). Non è quindi possibile dichiarare il contenuto di un elemento come un tipo di dato diverso da una stringa, ad esempio un intero o un vettore. Tuttavia, come già detto, questo vincolo non comporta grossi impedimenti o restrizioni nella maggioranza delle applicazioni del linguaggio XML.

La DTD può essere interna al documento XML oppure esterna (ad esempio può risiedere in un file a parte). Nel caso sia esterna, il prologo del documento deve contenere il riferimento al file, nella dichiarazione `DOCTYPE` e dopo la parola chiave `SYSTEM`; nel caso sia interna, essa deve trovarsi per intero nel prologo del documento, nella dichiarazione `DOCTYPE` e tra parentesi quadre. È possibile la coesistenza di una DTD interna ed una esterna nel prologo di un documento XML; in tale caso sarà assegnata la precedenza a quella interna.

Caso di DTD esterna:

```
<!-- documento di esempio -->
<?xml version="1.0"?>
<!-- nella dichiarazione doctype c'è il nome -->
<!-- dell'elemento di root, poi il riferimento  -->
<!-- al file contenente la DTD  -->
<!DOCTYPE CALENDARIO_PARTITE SYSTEM "calend.dtd">
<CALENDARIO_PARTITE>
        <GIORNATA >
        ...
        </GIORNATA>
</CALENDARIO_PARTITE>
```

Caso di DTD interna:

```
<!-- documento di esempio -->
<?xml version="1.0"?>
<!DOCTYPE CALENDARIO_PARTITE [
        <!-- la DTD è contenuta interamente tra queste parentesi
        quadre -->
]>
<CALENDARIO_PARTITE>
        <GIORNATA >
        ...
        </GIORNATA>
</CALENDARIO_PARTITE>
```

La sintassi delle dichiarazioni delle varie parti di un documento XML all'interno di una DTD è molto semplice, ed è oggetto del resto del paragrafo. La trattazione è sufficiente a spiegare come redigere una propria DTD, ma per uno studio esaustivo si rimanda alla specifica del linguaggio [3-3].

Dichiarazione delle entità

Nel caso l'utente voglia definire delle entità proprie (non predefinite) per l'utilizzo all'interno di un documento XML, tali entità devono essere dichiarate. La dichiarazione di un'entità comincia con la parola chiave `ENTITY`, e segue lo schema:

```
<!ENTITY Nome_entità Definizione>
```

Le entità possono essere analizzabili o non analizzabili. Le prime contengono semplice testo; le seconde sono riferimenti ad un file binario esterno (entità binarie), oppure possono essere costituite da testo non analizzabile.

Un esempio di dichiarazione di entità interna è il seguente:

```
<!ENTITY FIRMA "Andrea Esposito">
```

L'entità appena esemplificata può essere richiamata nel documento XML con il codice `&FIRMA;` all'interno di un elemento.

Un'entità esterna invece è dichiarata con la parola chiave `SYSTEM`.

```
<!ENTITY FOTO1 SYSTEM "Immagine1.jpg" NDATA JPG>
```

La parola chiave `NDATA` indica una cosiddetta annotazione, ossia la specifica dell'applicazione da usarsi per elaborare l'entità non analizzabile. La precedente dichiarazione deve quindi essere accompagnata dall'oppor-tuna annotazione che indica come elaborare la classe di entità `JPG`, ad esempio:

```
<!NOTATION JPG SYSTEM "jpgview.exe">
```

Dichiarazione degli elementi

Ogni dichiarazione di elemento comincia con la parola chiave `ELEMENT`, contiene il nome dell'elemento e, tra parentesi tonde, il tipo di dati che l'elemento deve contenere. Risulta dunque di questo tipo:

```
<!ELEMENT Nome_elemento (Contenuto)>
```

Per quanto riguarda il contenuto, ci sono diverse possibilità:

- L'elemento è vuoto; in tal caso si usa la parola chiave `EMPTY` all'interno delle parentesi.
- L'elemento contiene dati da analizzare (si indica all'elaboratore che deve esaminarli); in tal caso si usa la parola chiave `#PCDATA` (che sta per Parsed Character DATA) all'interno delle parentesi. È il caso di contenuto misto dati-elementi.
- L'elemento contiene dati da non analizzare (l'elaboratore li ignorerà); in tal caso si usa la parola chiave `#CDATA` all'interno delle parentesi.
- L'elemento può contenere qualsiasi cosa definita nella DTD senza ulteriori specifiche; in tal caso si usa la parola chiave `ANY` all'interno delle parentesi.
- L'elemento contiene soltanto altri elementi; in tal caso si definisce la struttura di tali elementi all'interno delle parentesi (se non si desidera una struttura precisa, si usa `ANY`)

Nella dichiarazione della struttura degli elementi (ossia all'interno delle parentesi tonde nell'ultimo caso del precedente elenco), alcuni caratteri speciali indicano le regole per la comparsa degli elementi stessi. Questa sintassi è molto importante se si vogliono stabilire relazioni logiche tra gli elementi:

- Le parentesi tonde "( )" racchiudono una serie di elementi.
- La virgola "," separa gli elementi imponendo il loro ordine di presentazione.
- Il simbolo *pipe* "|" separa una serie di elementi imponendo un'alternativa tra di loro.

- Il nome di un elemento da solo impone che quell'elemento compaia una volta.
- Il nome di un elemento seguito dal punto interrogativo "?" indica che l'elemento è opzionale e, se compare, può comparire una sola volta.
- Il nome di un elemento seguito dal simbolo più "+" indica che l'elemento è obbligatorio e può comparire più volte, ossia deve comparire **almeno** una volta.
- Il nome di un elemento seguito dal simbolo asterisco "*" indica che l'elemento è opzionale e, se compare, può figurare anche più volte.

Dichiarazione degli attributi

Gli attributi vengono dichiarati nella DTD usando la seguente sintassi:
`<!ATTLIST Nome_elemento Nome_attributo Tipo_Attributo Default>`
La parola chiave `ATTLIST` identifica la dichiarazione di un attributo, ed è seguita dal nome dell'elemento in cui l'attributo deve comparire, dal nome dell'attributo, dal tipo di attributo e dalle impostazioni predefinite per l'attributo.

Tipi di attributo:

- `CDATA`: l'attributo può contenere soltanto dati in formato carattere.
- `ENTITY`: il valore che assume l'attributo deve far riferimento ad un'entità binaria dichiarata.
- `ENTITIES`: è analogo a `ENTITY`, ma permette di specificare più valori separandoli con degli spazi.
- `ID`: il valore dell'attributo deve essere un identificatore univoco. Questo tipo di attributo può essere utile per catalogare gli elementi.

- `IDREF`: il valore dell'attributo deve essere un riferimento ad un `ID` dichiarato altrove nel documento.
- `IDREFS`: è simile all'attributo `IDREF`, ma permette di specificare più valori separati da spazi.
- `NOTATION`: il valore dell'attributo deve essere un riferimento ad una tra un elenco di annotazioni dichiarate altrove nella DTD.
- *Enumerated*: consente di specificare con esattezza tutti e soli i possibili valori dell'attributo. In questo caso si utilizzano le alternative tra parentesi e non una parola chiave. Ad esempio:
  ```
  <!ATTLIST Nome_tag RISPOSTA (sì|no) Default>
  ```

Le impostazioni predefinite per il valore dell'attributo sono invece:

- `#REQUIRED`: ogni elemento contenente questo attributo deve obbligatoriamente specificarne un valore.
- `#IMPLIED`: indica che l'attributo è opzionale e può non essere specificato.
- `#FIXED` *fixedvalue*: impone che l'attributo abbia il valore *fixedvalue*. Se l'attributo non è specificato nell'elemento, viene comunque stabilito il valore *fixedvalue.*
- *Default*: Indica il valore predefinito per l'attributo nel caso non venga specificato nell'elemento.

Alla luce di quanto detto per le dichiarazioni della varie parti di un documento XML, si ripropone l'esempio di prima completo della DTD.

```
<!-- documento di esempio -->
<?xml version="1.0"?>
<!DOCTYPE CALENDARIO_PARTITE [
       <!-- dichiarazione elementi -->
```

```
        <!ELEMENT CALENDARIO_PARTITE (GIORNATA+)>
        <!ELEMENT GIORNATA (DATA, LUOGO, ORA, PARTECIPANTI,
        RISULTATO, VITTORIA?)>
        <!ELEMENT DATA (#PCDATA)>
        <!ELEMENT LUOGO (#PCDATA)>
        <!ELEMENT ORA (#PCDATA)>
        <!ELEMENT PARTECIPANTI (SQUADRA_1, SQUADRA_2)>
        <!ELEMENT SQUADRA_1 (#PCDATA)>
        <!ELEMENT SQUADRA_2 (#PCDATA)>
        <!ELEMENT RISULTATO (#PCDATA)>
        <!ELEMENT VITTORIA (EMPTY)>
        <!-- dichiarazione attributi -->
        <!ATTLIST GIORNATA numero CDATA #REQUIRED>
]]>
<!-- fine prologo -->
<CALENDARIO_PARTITE>
        <GIORNATA numero="1">
        ...
        </GIORNATA>
        ...
</CALENDARIO_PARTITE>
```

3.2.2 XML Schema

XML Schema è un approccio alla definizione del tipo di documento che offre tutte le funzionalità della DTD ed altre aggiuntive, ad esempio consente di creare relazioni più complesse tra gli elementi e caratterizzarli maggiormente. È infatti possibile definire tipi di dati per il loro contenuto, superando così il principale limite della DTD che operava soltanto con stringhe di caratteri.

XML Schema è un linguaggio più potente e flessibile, ma anche più complesso rispetto alla DTD. Una sua trattazione approfondita esula dagli scopi di questo lavoro, pertanto si rimanda il lettore interessato a [3-4]; in questa sede ci si limita a descrivere le sue caratteristiche fondamentali e fornire un semplice esempio.

XML Schema è un linguaggio di markup che utilizza la stessa sintassi di XML, traendone gli ovvi vantaggi in quanto a semplicità di gestione, compatibilità e possibilità di espansione. Un documento XML Schema deve essere un documento XML "ben formato". Per documento ben formato[1] si intende un documento in cui ogni tag di apertura abbia il corrispettivo di chiusura (eccezion fatta per gli elementi vuoti), gli elementi siano annidati in modo appropriato, i valori degli attributi siano racchiusi tra virgolette e tutte le entità non predefinite siano dichiarate.
Un'altra funzionalità supplementare che il linguaggio offre rispetto alla DTD è il supporto per il *namespace*. Uno "schema" consiste di definizioni di tipi e dichiarazioni varie (elementi, attributi…) i cui identificativi appartengono ad uno "spazio dei nomi". L'autore di un documento XML potrebbe desiderare che il documento sia conforme a più schemi, oppure che sue diverse parti siano conformi a diversi schemi. Il supporto per lo spazio dei nomi concede questo tipo di flessibilità che la rigida – ed univocamente associata ad un documento – DTD non consente.

Nel linguaggio XML Schema le dichiarazioni delle parti costitutive di un documento XML sono viste nell'ottica della definizione di un tipo di dato. Si opera una fondamentale distinzione tra tipi di dati semplici e complessi.
I tipi di dati semplici sono quelli predefiniti per il linguaggio XML Schema (circa 50); tra di essi figurano bytes, stringhe, numeri decimali, esadecimali, tipi interi, booleani, unsigned, double, rappresentazioni di date e lassi temporali, e molti altri. Si possono creare nuovi tipi semplici applicando determinate restrizioni a quelli predefiniti (ad esempio una stringa di un numero fissato o massimo di caratteri, o un intero che può variare in un

[1] Da non confondersi con documento *valido*, che è un documento conforme ad una DTD.

range prefissato di valori). È possibile anche derivare tipi complessi dai tipi semplici (ad esempio un vettore di interi) mediante opportune dichiarazioni. I tipi di dati complessi possono essere sia una derivazione dei tipi semplici che una composizione di elementi, attributi e tipi semplici.

Un semplice esempio di schema è il seguente:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <xsd:element name="rubrica" type="recapito" minOccurs="0"
   maxOccurs="unbounded"/>

   <xsd:complexType nome="recapito">
      <xsd:sequence>
         <xsd:element name="nome" type="xsd:string"/>
         <xsd:element name="via" type="xsd:string"/>
         <xsd:element name="n_civico" type="xsd:decimal"/>
         <xsd:element name="comune" type="xsd:string"/>
         <xsd:element ref="sigla_provincia" type="sigla"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:simpleType name="sigla">
      <xsd:restriction base="xsd:string">
         <xsd:pattern value="[A-Z]{2}"/>
      </xsd:restriction>
   </xsd:sympleType>

</xsd:schema>
```

L'elemento di root è `schema`; il prefisso `xsd:` che si trova in ogni dichiarazione è l'indicazione del *namespace*, dichiarato come attributo nell'elemento di root.
L'elemento (vuoto) per la dichiarazione degli elementi è `element`; il nome dell'elemento dichiarato si trova nell'attributo `name`, mentre la presenza

dell'attributo `ref` in luogo di `name` significa che la dichiarazione fa riferimento ad un elemento già dichiarato.

L'elemento `rubrica` viene dichiarato come elemento opzionale (l'attributo `minOccurs` specifica il minimo numero di volte che l'elemento deve comparire; se non viene indicato, il valore di default è 1) che può comparire un numero arbitrario di volte (l'attributo `maxOccurs` specifica il massimo numero di volte per la comparsa, il valore `unbounded` significa che non ci sono vincoli in tal senso; se non viene indicato, il valore di default è 1). L'elemento `rubrica` è di tipo `recapito` (definito sotto).

L'elemento `recapito` viene definito come tipo complesso (contiene infatti altri elementi).

L'elemento `sigla_provincia` viene dichiarato di tipo "`sigla`"; il tipo `sigla` viene definito come un pattern di due lettere maiuscole all'interno dell'elemento `simpleType`.

Un esempio di documento XML conforme a questo schema è il seguente (i campi vanno riempiti opportunamente, sono tutti stringhe eccetto `n_civico` che deve essere un numero decimale e `provincia` che deve essere una sequenza di due lettere maiuscole):

```
<rubrica>
        <recapito>
                <nome>…</nome>
                <via>…</via>
                <n_civico>…</n_civico>
                <comune>…</comune>
                <provincia>…</provincia>
        </recapito>
</rubrica>
```

3.3 Strumenti per il linguaggio XML

Come detto in precedenza, uno dei punti di forza del linguaggio XML è quello di essere contornato da una serie di altri standard e linguaggi correlati che ne incrementano le potenzialità, riassunti schematicamente in figura 3.1.

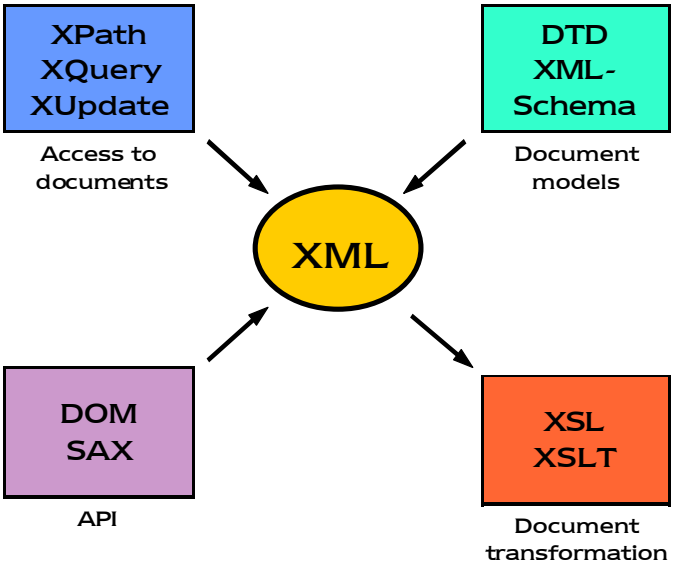

Figura 3.1 – Strumenti per il linguaggio XML

In questo paragrafo si compie un necessariamente superficiale excursus sui più utili tra questi strumenti; per approfondimenti in merito si rimanda ai riferimenti presenti in ogni paragrafo.

3.3.1 Extensible Stylesheet Language (XSL) e XSL Transformation (XSLT)

Il linguaggio XSL (eXtensible Stylesheet Language) [3-5], standard del W3C, è un linguaggio di markup progettato per definire metodi di visualizzazione e pubblicazione dei documenti XML, specialmente in ambito Web. In pratica, mentre il documento XML descrive la struttura ed il contenuto dei dati, il "foglio di stile" XSL specifica la maniera in cui questi debbano apparire. XSL segue la medesima sintassi di XML, ed un foglio di stile XSL è un documento XML ben formato.

Il linguaggio XSL consiste di due fondamentali parti:

- un vocabolario XML che fornisce la semantica per specificare la formattazione;
- un linguaggio per trasformare i documenti XML, chiamato XSLT.

Il foglio di stile XSL caratterizza la presentazione di una classe di documenti XML, descrivendo come un documento della classe sia trasformato in un altro documento XML che utilizza anche la semantica per la formattazione.

Un foglio di stile XSL è organizzato in *modelli*. Ogni modello determina una struttura di output, e contiene dei *pattern* utilizzati per specificare le parti del documento XML alle quali si riferisce il modello. Gli elementi e gli attributi del foglio di stile funzionano da comandi per l'elaboratore XSL, indicando come gestire i dati.

XSLT [3-6] è un sottoinsieme del linguaggio XSL, e definisce modalità di conversione di un tipo di documento XML in vari formati, ad esempio HTML oppure un altro tipo di documento XML. Molto spesso il linguaggio

XSLT viene trattato separatamente dal linguaggio XSL, poiché da solo esso costituisce un agevole strumento per la presentazione di un documento XML, utilizzabile senza dover imparare tutti i formalismi di XSL. Tuttavia è necessario specificare che XSLT non è stato progettato per essere completamente indipendente, essendo principalmente calibrato sulle classi di trasformazioni richieste da XSL. Un foglio di stile XSLT è un documento XML ben formato.
Le più comuni trasformazioni consentite, oltre a quella già accennata (da XML a HTML), sono quelle che convertono in report testuali, riordinano, modificano nei livelli gerarchici un documenti XML, oppure quelle che permettono l'aggiunta di ulteriori informazioni (come *hyperlinks*).

3.3.2 XPath e XQuery

XPath [3-7] e XQuery [3-8] sono due standard W3C strettamente correlati tra di loro, al punto che vengono spesso presentati assieme. Il primo è antecedente al secondo; lo scopo della loro creazione è la definizione di un linguaggio per un accesso comodo e flessibile a parti di un documenti XML.

XPath è un linguaggio largamente utilizzato e supportato che serve per ricercare, identificare e selezionare determinate sezioni di un documento XML. A supporto di questo scopo principale, fornisce anche elementari funzionalità per trattare stringhe, numeri e dati booleani.
Il suo funzionamento si basa sulla creazione di un albero a partire dal documento, e la sintassi assai succinta permette di indirizzare una specifica parte attraverso i nodi dell'albero con una semplice espressione di tipo *path*.

Il termine XPath deriva infatti proprio dal tipo di notazione usata per individuare un percorso all'interno di un documento XML, analoga a quella di un URL (Uniform Resource Locator). L'utilizzo di XPath può tornare particolarmente utile quando si debbano manipolare soltanto piccole porzioni di grandi documenti, sia per trasferirle, sia per visualizzarle, magari tramite XSL[2].

XPath opera sulla struttura logica del documento, creando nodi di diverso tipo, tra cui quelli per elementi, attributi e testo semplice. La struttura sintattica fondamentale del linguaggio è detta *espressione*; l'elaborazione di un'espressione può portare ad un risultato tra quattro tipi diversi: un set di nodi, un valore booleano, un numero o una stringa.
Tra le espressioni rivestono particolare importanza quelle di ricerca, che restituiscono un set contenente i nodi che soddisfano i parametri impostati nel *path* di ricerca. Seguono due esempi di espressioni di ricerca:

```
/name1/name2/test
```

(ricerca tutti gli elementi `test` inclusi in un elemento `name2` incluso a sua volta in un elemento `name1`)

```
/name1[@attr1="value1"]/test
```

(ricerca tutti gli elementi `test` inclusi in un elemento `name1` con attributo `attr1` che vale `value1`)

XQuery è un linguaggio di query concepito appositamente per essere applicabile a qualsiasi sorta di documento XML, e si basa sull'utilizzo di XPath per la specificazione di percorsi all'interno dei documenti.

[2] In effetti, XPath è nato come strumento di supporto di XSLT

XQuery ha funzionalità che consentono di poter attingere da fonti di dati multiple per la ricerca, per filtrare i documenti o riunire i contenuti di interesse.
Come XPath, anche XQuery è un linguaggio che si basa su espressioni, ed è possibile comporre arbitrariamente tali espressioni (ad esempio calcolandone una con il risultato di altre in ingresso), purché i loro tipi siano compatibili. Una funzionalità molto importante di XQuery è la possibilità di importare definizioni di tipi da uno schema di XML Schema.

3.3.3 Document Object Model (DOM) e Simple API for XML (SAX)

Una API (Application Program Interface) è uno strumento che fornisce una sintassi di accesso ad un certo protocollo, per un dato sistema operativo o per un dato linguaggio di programmazione.
Esistono molte API – progettate per la grande maggioranza dei linguaggi di programmazione – attorno allo standard XML. Esse consentono la creazione, il parsing, la convalida, la manipolazione di un documento XML.
Le due più diffuse e conosciute, Document Object Model e Simple API for XML, sono concepite con diverse modalità di accesso ai documenti, e rappresentano in effetti ciascuna il principale esponente dei due preminenti metodi di elaborazione per XML: le API rispettivamente *tree-based* ed *event-based.*

Lo standard Document Object Model (DOM) [3-9] è una raccomandazione W3C che fa uso dei molti altri standard correlati ad XML, nata per fornire una soluzione neutrale e sulla quale tutti possano convergere per la gestione

di documenti XML, HTML, fogli di stile. Questa API è indipendente dalla piattaforma e dal linguaggio di programmazione e consente a programmi e *script* di accedere dinamicamente ai documenti, modificarne il contenuto o la struttura, lo stile, effettuarne convalida e parsing.
Come accennato in precedenza, DOM è un'interfaccia tree-based, ossia effettua il parsing di un documento creando un albero nel quale i nodi sono gli elementi costitutivi del documento stesso. Per quanto riguarda i documenti XML, esempi di tipi di nodo sono i nodi *document*, *element*, *attribute*, *character data.* L'albero resta in memoria per tutto il tempo dell'elaborazione, ed è consentito ad un'applicazione di accedervi per compiere le operazioni elencate in precedenza. Questo è forse il limite principale dell'approccio tree-based: se l'elaborazione riguarda un documento di grandi dimensioni, la procedura della creazione in memoria dell'albero è oltremodo esigente per le risorse del sistema.

L'interfaccia Simple API for XML (SAX) [3-10], nata originariamente per il linguaggio Java, è divenuta uno standard *de facto*; questa API, facente parte del progetto SourceForge [3-11], ha infatti conosciuto successo e larga diffusione, ed adesso ne esistono versioni per svariati linguaggi di programmazione.
SAX è un'API event-based, ossia informa direttamente l'applicazione degli eventi di parsing (come l'apertura e la chiusura degli elementi in un documento XML), e non costruisce alcun albero in memoria. L'applicazione reagirà con un opportuno *handler* alla segnalazione di un evento.
I vantaggi di questo tipo di accesso rispetto al modello tree-based sono una drastica riduzione della memoria richiesta per il processing dei documenti ed un incremento di efficienza nell'elaborazione. Quest'ultimo è dovuto al fatto che alcune applicazioni necessitano di ridefinire l'albero secondo pro-

prie strutture dati, con il risultato di una doppia elaborazione. Tali vantaggi sono pagati con una leggermente maggiore complessità di programmazione.

3.4 Network management con XML

Una rete informatica è un sistema complesso in ambito distribuito, per la cui gestione si deve ricorrere a risorse interne alla rete stessa. La complessità risiede sia nel numero dei nodi di cui è costituita, sia nella loro distanza geografica, sia nella loro eterogeneità, in quanto essa accoglie tecnologie e protocolli differenti.
La gestione efficiente di una rete richiede di avere sotto controllo una serie di rapporti dai suoi nodi. Occorre dunque un protocollo che consenta di leggere e scrivere in modo remoto informazioni di stato nei nodi della rete. Recenti proposte (delle quali si indicano i riferimenti nel paragrafo 3.4.2) hanno additato XML come linguaggio ottimale per l'implementazione di un simile protocollo.

3.4.1 Attuali sistemi di network management e loro limiti

Un sistema di network management è costituito da due parti fondamentali. La prima è un *manager*, che rappresenta l'interfaccia esterna del sistema e deve consentire all'amministratore della rete di raccogliere informazioni ed impostare lo stato dei nodi della rete. La seconda è un *agent*, che rappresenta la parte operativa, ossia un processo attivo in ogni nodo che lo rende accessibile per le operazioni appena menzionate.
Gli attuali standard di network management prevedono, con poche eccezioni, l'utilizzo del protocollo SNMP (Simple Network Management

Protocol) per la comunicazione tra i due blocchi del sistema.
SNMP è largamente utilizzato e supportato, e rappresenta senz'altro una valida tecnologia che fin dai primi anni '90 risponde in modo consistente alla maggior parte delle esigenze. È stato oggetto di lavoro di perfezionamento ed alcuni suoi difetti sono stati superati, ed ha rappresentato una solida base di investimento e sviluppo per i gestori di reti.
SNMP consiste in pratica in una sorta di protocollo *request-reply* specializzato, che utilizza UDP per il trasporto e si basa su due tipi fondamentali di messaggi: GET (per ottenere informazioni da un nodo) e SET (per scrivere informazioni su un nodo). Solitamente SNMP è usato tramite un client con interfaccia grafica per agevolare l'operatore.

SNMP presenta dei fondamentali limiti nell'efficienza operativa, nella scalabilità e nella difficoltà di aggiornamento. Per quanto riguarda l'efficienza e la scalabilità, i messaggi SNMP non consentono di trasferire complesse o cospicue informazioni con una sola chiamata (è necessaria a tale scopo una successione di richieste elementari).
Il problema della difficoltà di aggiornamento dipende dal fatto che SNMP, da solo, non è sufficiente a svolgere il lavoro. Dal momento che le reti sono eterogenee, e le informazioni potrebbero essere di tipo diverso da un nodo all'altro, è necessaria la contestuale definizione di un protocollo aggiuntivo chiamato MIB (Management Information Base). Per quanti sforzi ci siano stati nel cercare di realizzare protocolli di management il più possibile universali, la diversità degli hardware da un costruttore all'altro, e il passaggio ad un hardware più recente richiedono sempre una nuova definizione di MIB propri per i gestori di reti, prospettiva assai dispendiosa in termini di tempo e risorse economiche poiché richiede personale altamente specializzato. Inoltre, con ogni nuovo protocollo di gestione si

perde con buona probabilità la compatibilità con il vecchio hardware, altra prospettiva non certo entusiasmante.

3.4.2 Applicazione di XML al network management

In questo capitolo sono state presentate le caratteristiche del linguaggio XML e nel paragrafo 3.1.2 sono stati evidenziati i suoi principali vantaggi. Alla luce di quanto detto, risulta logico capire come esso possa costituire il rimedio ai problemi che SNMP sta incontrando.

XML è diffuso e molto utilizzato. Gode del supporto di utili strumenti di corredo nonché di numerose API, le quali consentono alle applicazioni di accedere agevolmente alle informazioni di management. Il linguaggio XML Schema può essere utilizzato per definire la struttura dei documenti ed il tipo di informazioni con il dettaglio desiderato, ed effettuare la convalida dei messaggi. XPath ed XQuery consentono di selezionare parte delle informazioni, che poi possono essere elaborate tramite XSL ed XSLT per essere presentate in una vasta scelta di formati. L'ausilio di tutti questi strumenti rende agevole la creazione sia di *manager* che di *agent* basati su XML.

Un protocollo basato su XML è semplice da progettare e da gestire a livello operativo, è efficiente nel trasporto dei dati, e soprattutto si presta ad essere esteso e rivisto con semplicità e senza perdita di compatibilità con le versioni precedenti. Costruire un protocollo per la comunicazione tra *manager* ed *agent* basato su XML rappresenta senz'altro un modo flessibile ed efficiente per scambiare dati di management. Modelli creati con XML Schema possono agevolmente sostituire le definizioni MIB, consentendo di trattare dati arbitrariamente complessi. Per quanto riguarda il trasporto dei

messaggi, XML può appoggiarsi ad un qualsiasi protocollo di trasporto affidabile, traendo vantaggio dalla possibilità di sceglierne uno largamente implementato, come ad esempio TCP.

Le proposte che sono effettuate in questo contesto sono molte; si va dall'utilizzo di XML per la codifica ed il trasferimento dei dati di management con svariati protocolli – ad esempio di tipo RPC (Remote Procedure Call), all'integrazione di XML con l'architettura *web-server* per il network management, alla gestione basata su XML di Internet e delle reti IP in generale, all'integrazione di XML e SNMP, alla creazione di interi sistemi di management (*manager*, *agent* e protocolli) basati su XML. Si rimanda il lettore interessato ad approfondimenti sul network management con XML ai seguenti riferimenti: [3-12] [3-13] [3-14] [3-15] [3-16] [3-17].

3.5 Utilizzo del linguaggio XML nel progetto

Nel secondo capitolo (paragrafo 2.5) si è visto come il Client Interface Manager (CIM) debba gestire dei moduli Client Interface (CI) connessi ciascuno ad un Metercontroller. Tramite l'accesso al CIM è possibile pilotare in modo remoto i Metercontroller per farsi inviare i dati sul traffico nei MA-LER DS nei quali è attivo il sistema di misura. Una fase importante del progetto è stata quella di creare dei protocolli ad hoc per lo scambio di queste informazioni di gestione.
Si noti che lo stesso CIM necessita di protocolli di accesso per la gestione delle operazioni di controllo e delle operazioni di stima.
Scartata una prima idea di approccio TLV (*type*, *length*, *value*), si è scelto di basare tutti i protocolli di controllo del progetto sul linguaggio XML, in

virtù della sua flessibilità, semplicità di utilizzo e della possibilità di rivedere ed espandere facilmente i protocolli stessi.
Le definizioni dei protocolli sono state create mediante l'utilizzo della DTD. La convalida secondo il modello di documento ha rappresentato un utile strumento: consente infatti un'estrema semplificazione per il controllo di validità di un messaggio ricevuto dal CIM o dal Metercontroller. Non si è ritenuto necessario far ricorso a XML Schema poiché i dati scambiati nel protocollo sono piuttosto semplici (stringhe, double o flag binari), ed inoltre le librerie standard del C forniscono mezzi comodi per la conversione di stringhe. I limiti della DTD non hanno perciò posto alcuna restrizione.

3.5.1 La libreria Libxml2

Nello sviluppo del presente progetto si è fatto utilizzo della libreria Libxml2 [3-18] versione 2.9.10, un kit di strumenti per il parsing e l'elaborazione dei documenti XML realizzato nell'ambito del progetto Gnome [3-19]. Libxml2 è *free software* disponibile secondo la MIT License [3-20].
La libreria è scritta in C, ma contiene associazioni di linguaggi che la rendono disponibile in altri ambienti di programmazione; è progettata per funzionare in una grande varietà di sistemi operativi. Libxml2 implementa molti degli standard correlati ad XML, tra cui DTD, XML Schema, XPath.
La rappresentazione interna di un documento XML è modellata secondo i canoni DOM (di tipo tree-based), ma è possibile anche gestire i documenti con un'interfaccia di tipo SAX (event-based).

In particolare è stata utilizzata con profitto l'API Libxml2 XmlTextReader, interfaccia tree-based per il linguaggio C nella quale, una volta costruito in

memoria l'albero a partire dal documento XML, è sufficiente chiamare ripetutamente la funzione `XmlTextReaderRead()` per scorrere i nodi in sequenza. Ad ogni chiamata è possibile interrogare direttamente il nodo in questione e recuperare ogni sorta di informazione. I nodi corrispondenti agli attributi non sono attraversati in questo modo e necessitano di una leggera complicazione del codice, per cui si è scelto di utilizzare esclusivamente elementi nei protocolli sviluppati, cosa che non presenta alcuno svantaggio né limitazioni. I messaggi scambiati sono raramente superiori ai 500 bytes nelle dimensioni, per cui l'API tree-based non desta i problemi visti per l'utilizzo delle risorse.

L'interfaccia XmlTextReader consente anche la convalida secondo DTD del documento XML. Le funzioni per il parsing della messaggistica XML del presente progetto utilizzano tutte questa API, che è risultata potente e di facile apprendimento ed utilizzo, grazie anche alla esaustiva documentazione disponibile.