L'SGML non è un «linguaggio di scrittura» da imparare e usare così com'è. Al contrario, è un linguaggio per definire il modo in cui il testo deve essere scritto: solo dopo si può iniziare a scrivere secondo le regole stabilite.

Volendo fare un abbinamento con i linguaggi di programmazione, sarebbe come se prima si dovesse definire il linguaggio stesso, per poi poter scrivere i programmi secondo quelle regole.

La descrizione che viene fatta potrebbe risultare noiosa, considerato che solo dopo molte sezioni si mostra in che modo realizzare effettivamente il proprio DTD e applicarlo a un documento. Considerata la complessità dei concetti espressi, si ritiene più conveniente una spiegazione che parte dal basso, piuttosto che usare un approccio inverso, il quale presumerebbe una conoscenza minima di partenza.

51.1 DTD: definizione del tipo di documento

Le regole che definiscono la struttura e la scomposizione del documento, assieme a quasi tutte le altre che governano la logica dell'SGML, sono contenute nel DTD.

Queste regole possono essere permissive o restrittive, in funzione degli obiettivi che ci si prefigge; ovvero, in funzione del contenuto di quel tipo di documento e delle cose che con questo ci si aspetta di fare.

La complessità del mondo reale, fa sì che non ci sia modo di realizzare un DTD unico che vada bene per tutti gli scopi. Un DTD

ipotetico, che volesse andare bene un po' per tutto, dovrebbe essere anche qualcosa di estremamente generico e permissivo, annullando tutti i benefici dell'utilizzo dell'SGML.

Esempi reali di DTD «tuttofare» sono quelli delle prime versioni dell'HTML, in cui tutto si concentra nella definizione di elementi il cui scopo prevalente è definire, anche se solo vagamente, l'aspetto finale che dovrebbe avere il risultato. Lo scopo dell'SGML non è quello di stabilire il risultato finale del documento, tuttavia, si può benissimo predisporre un DTD orientato a questo obiettivo. Ma questo, nel caso dell'HTML, giustifica poi l'estrema debolezza della sua struttura, dove è ammesso quasi tutto.

È difficile comprendere subito il significato pratico di questo approccio: la definizione del tipo di documento e poi la scrittura del testo. Lo si può comprendere solo quando si lavora assiduamente nell'ambito della produzione di documentazione, quando ci si accorge che le proprie esigenze sono diverse da quelle degli altri, per cui diventa difficile adattarsi all'uso di modelli già esistenti.

51.1.1 Elementi

Dal punto di vista di SGML, una singola unità di testo la cui dimensione varia a seconda del contesto, è un ***elemento***, a cui si impone l'attribuzione di un nome. SGML non fornisce alcun modo per attribuire un significato agli elementi del testo, tranne per il fatto di avergli dato un nome. Piuttosto, attraverso un analizzatore SGML, è possibile verificare che questi siano collocati correttamente secondo le relazioni stabilite.

I nomi degli elementi, sono definiti tecnicamente ***identificatori generici***, utilizzando la sigla GI (*Generic identifier*).

Nel sorgente SGML, gli elementi sono indicati normalmente attraverso l'uso di marcatori che hanno la forma consueta '**<**...**>**' e '**</**...**>**', dove il primo inizia l'elemento nominato tra le parentesi angolari e il secondo chiude l'elemento. Per esempio, si potrebbe definire l'elemento '**acronimo**' e utilizzarlo nel testo nel modo seguente:

```
...Il gruppo <acronimo>ILDP</acronimo> si occupa di...
```

Il significato che questo elemento può avere, non è definito dall'SGML. Il fatto di avere delimitato l'elemento '**acronimo**' potrebbe servire per estrarre dal documento tutte le sigle utilizzate, per inserire queste in un indice particolare, oppure solo per fini stilistici di evidenziamento uniforme.

La difficoltà nella scrittura di un testo in SGML si riduce a questo: utilizzare i marcatori necessari a identificare correttamente i vari elementi del testo, secondo le regole stabilite nella definizione del documento stesso (il DTD).

51.1.1.1 Abbreviazioni nell'indicazione degli elementi

Prima ancora di iniziare a vedere il contenuto del DTD, è bene chiarire che esistono altri modi per delimitare un elemento SGML. Per la precisione, si tratta di abbreviazioni di cui alcuni autori non riescono a fare a meno. La scrittura di un sorgente SGML è un po' come quella di un sorgente di un linguaggio di programmazione: si può essere concisi o prolissi. Di solito, quando si è concisi si scrive del codice difficile da leggere, mentre in generale è meglio scrivere tutto in forma chiara senza risparmiare. L'esempio visto in precedenza,

```
...Il gruppo <acronimo>ILDP</acronimo> si occupa di...
```

può essere abbreviato in

```
...Il gruppo <acronimo>ILDP</> si occupa di...
```

e anche nel modo seguente, che però porta con sé un vincolo importante: non si possono usare delle barre oblique all'interno dell'elemento abbreviato in questo modo.

```
...Il gruppo <acronimo/ILDP/ si occupa di...
```

Con questi sistemi, oltre a rendere il sorgente SGML poco leggibile, si rischia di non ottenere i risultati che si attendono se gli strumenti di elaborazione utilizzati non riconoscono tali estensioni del linguaggio.

51.1.1.2 Primo impatto con un DTD

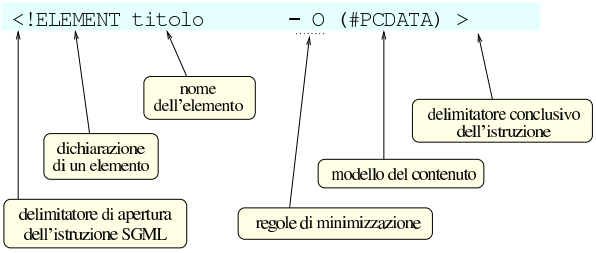
La definizione del DTD è ottenuta da una serie di istruzioni dichiarative composte secondo una sintassi molto semplice. L'esempio seguente rappresenta le istruzioni necessarie a definire gli elementi di un tipo di documento ipotetico definibile come '**relazione**'.

```
<!ELEMENT relazione     - - (titolo?, data, contenuto) >
<!ELEMENT titolo        - o (#PCDATA) >
<!ELEMENT data          - o (#PCDATA) >
<!ELEMENT contenuto     - o (paragrafo+, firma+) >
<!ELEMENT paragrafo     - o (#PCDATA) >
<!ELEMENT firma         - o (#PCDATA) >
```

Ognuna delle righe che appaiono nell'esempio rappresenta una dichiarazione di un elemento SGML. Una dichiarazione, di qualunque tipo, è delimitata da una parentesi angolare aperta (il simbolo di minore), seguita immediatamente da un punto esclamativo ('**<!**') e da una parentesi angolare chiusa ('**>**').

La dichiarazione di un elemento si compone poi della parola chiave '**ELEMENT**', seguita dal nome dell'elemento, dalle regole di minimizzazione rappresentate da due caratteri e da un modello del contenuto.

Figura 51.6. Scomposizione delle varie parti della dichiarazione di un elemento SGML.



Le varie parti che compongono qualunque tipo di dichiarazione SGML sono separate da spazi orizzontali (caratteri spazio, o tabulazioni orizzontali) oppure anche da interruzioni di riga, permettendo così di proseguire le istruzioni su più righe distinte.

51.1.2 Regole di minimizzazione

Le ***regole di minimizzazione***, rappresentate da due caratteri staccati, indicano l'obbligatorietà o meno dell'utilizzo del marcatore di apertura e di chiusura per l'elemento dichiarato. Il primo dei due simboli rappresenta l'apertura, il secondo la chiusura. Un trattino indica che il marcatore è obbligatorio, mentre la lettera '**O**' sta per «opzionale» e indica così che può essere omesso:

Regole di minimizzazione	Descrizione
- -	sono obbligatori entrambi i marcatori;
- O	è obbligatorio il marcatore iniziale, mentre quello finale è facoltativo;
O -	il marcatore iniziale è facoltativo, mentre quello finale è obbligatorio (di solito non capita questa situazione);
O O	sono facoltativi entrambi i marcatori.

Nell'esempio mostrato in precedenza, solo l'elemento '**relazione**' richiede l'utilizzo di marcatori di apertura e di chiusura, mentre tutti gli altri possono essere indicati utilizzano il solo marcatore di apertura. In pratica, il contesto permette di individuare dove finiscano tali elementi nel testo.

La possibilità o meno di rendere facoltativo l'uso dei marcatori di apertura e di chiusura non è solo un fatto di gusto, in quanto dipende anche dall'organizzazione del tipo di documento. Se le dichiarazioni diventano ambigue, non si possono più distinguere gli elementi nel testo SGML.

51.1.3 Modello del contenuto

La parte finale della dichiarazione di un elemento SGML è il ***modello del contenuto***, che si distingue perché è racchiuso tra parentesi tonde. Serve a descrivere il tipo di contenuto che può avere l'elemento e si può esprimere attraverso parole riservate che hanno un significato preciso, come '**#PCDATA**' (*Parsed character data*) che rappresenta una qualunque sequenza di caratteri valida, oppure attraverso l'indicazione di nomi di altri elementi che possono (o devono) essere contenuti in qualche modo.

Il modello del contenuto, può articolarsi in modo molto complesso, allo scopo di definire le relazioni tra gli elementi contenuti.

Per il momento, è bene osservare che un elemento, il cui modello del contenuto sia composto esclusivamente della parola riservata '**#PCDATA**', non può contenere al suo interno altri tipi di elementi. Il significato di alcune delle parole riservate più comuni, utilizzabili per definire il contenuto di un elemento, sono riportate più avanti in questo capitolo, dopo la presentazione di altri concetti essenziali, necessari per comprenderne il senso.

51.1.3.1 Indicatori di ripetizione

Il modello del contenuto utilizza un sistema abbastanza complesso per definire la possibilità di contenere più elementi dello stesso tipo e per indicare raggruppamenti di elementi. Per indicare la ripetizione, viene usato un simbolo alla fine dell'oggetto a cui si riferisce, chiamato ***indicatore di ripetizione*** (*occurrence indicator*):

Indicatore di ripetizione	Descrizione
+	il segno '**+**' usato come suffisso, rappresenta una o più ripetizioni dell'elemento;
?	il segno '**?**' usato come suffisso, rappresenta zero o al massimo un'occorrenza dell'elemento;
*	il segno '*****' usato come suffisso, rappresenta zero o più ripetizioni dell'elemento;
	se non viene usato nessun suffisso, l'elemento indicato deve essere usato esattamente una volta.

Dall'esempio mostrato in precedenza viene ripreso l'estratto seguente, nel quale si può osservare che: l'elemento '**titolo**' può apparire al massimo una volta all'interno di '**relazione**' (precisamente all'inizio di questo elemento); l'elemento '**paragrafo**' deve essere contenuto almeno una volta all'interno dell'elemento '**contenuto**' (lo stesso vale per l'elemento '**firma**'); l'elemento '**firma**' può contenere solo caratteri normali senza altri elementi.

```
<!ELEMENT relazione     - - (titolo?, data, contenuto) >
<!ELEMENT contenuto     - O (paragrafo+, firma+) >
<!ELEMENT firma         - O (#PCDATA) >
```

51.1.3.2 Connettori

Quando un elemento deve poter contenere diversi tipi di elementi, è necessario usare dei simboli, detti ***connettori***, per stabilirne la relazione:

Connettore	Descrizione
,	la virgola ('**,**') indica che l'elemento precedente e quello successivo devono apparire nell'ordine in cui sono;
&	la e-commerciale ('**&**') indica che l'elemento precedente e quello successivo devono essere presenti entrambi, ma possono apparire in qualunque ordine;
|	la barra verticale ('**|**') indica che solo uno tra i due elementi che connette può apparire.

Riprendendo il solito estratto dell'esempio già mostrato precedentemente, si può osservare l'uso della virgola in qualità di connettore:


```
<!ELEMENT relazione     - - (titolo?, data, contenuto) >
<!ELEMENT contenuto     - O (paragrafo+, firma+) >
<!ELEMENT firma         - O (#PCDATA) >
```

L'elemento '**relazione**' può contenere al massimo un titolo all'inizio, quindi deve apparire un elemento '**data**' e dopo di questo anche un elemento '**contenuto**'. L'elemento '**contenuto**' deve contenere uno o più elementi '**paragrafo**' a partire dall'inizio, mentre in coda deve avere uno o più elementi '**firma**'.

```
<!ELEMENT nominativo    - - (nome & cognome) >
<!ELEMENT voce          - - (punto | numero) >
```

Per completare gli esempi sull'uso dei connettori, si osservi quanto mostrato sopra. L'elemento '**nominativo**' deve contenere un elemento '**nome**' e un elemento '**cognome**', in qualunque ordine; l'elemento '**voce**' può contenere solo un elemento a scelta tra '**punto**' e '**numero**'.

51.1.3.3 Raggruppamenti

All'interno di un modello di contenuto, è possibile indicare dei raggruppamenti che esprimono in pratica dei sottomodelli, a cui poter applicare gli indicatori di ripetizione e i connettori. Per questo si usano le parentesi tonde. Si osservi l'esempio seguente:

```
<!ELEMENT figure - - ( (eps | ph), img*, caption?) >
```

L'elemento '**figure**' deve contenere un'occorrenza del sottogruppo '**(eps | ph)**', zero o più ripetizioni dell'elemento '**img**' e al massimo un'occorrenza di '**caption**', nell'ordine descritto. Il sottogruppo '**(eps | ph)**' rappresenta una singola occorrenza di '**eps**' oppure '**ph**'.

Quando si utilizzano gli operatori di ripetizione assieme ai raggruppamenti, possono nascere degli equivoci. Ammesso che ciò possa avere senso, si osservi la variante seguente dell'esempio già presentato:

```
<!ELEMENT figure - - ( (eps | ph)+, img*, caption?) >
```

È stato aggiunto il segno '**+**' dopo il gruppo '**(eps | ph)**'. In questo modo, si intende che sia possibile l'inserimento iniziale di una serie indefinita di elementi '**eps**' o '**ph**', in qualunque ordine, purché ce ne sia almeno uno dei due. Quindi, non è necessario che si tratti solo di elementi '**eps**' o solo di '**ph**'.

51.1.3.4 Eccezione

Se nella definizione di un elemento si vogliono indicare delle eccezioni a quanto definito dal modello di contenuto, si può indicare un gruppo di elementi successivo al modello del contenuto.

Questo gruppo può essere preceduto dal segno '**+**' o dal segno '**-**' indicando rispettivamente un'eccezione di inclusione, o un'eccezione di esclusione.

```
<!ELEMENT address - O (#PCDATA) +(newline) >
```

L'esempio mostra l'elemento '**address**' contiene caratteri normali, ma che può includere eccezionalmente anche l'elemento '**newline**'.

```
<!ELEMENT acronimo - - (#PCDATA) -(acronimo) >
```

L'esempio mostra l"elemento '**acronimo**' contiene caratteri normali e che non può includere se stesso (a essere precisi, non è necessario dichiarare una cosa del genere, dal momento che il contenuto '**#PCDATA**' non ammette altri elementi al suo interno).[1]

51.1.3.5 Elementi vuoti

Alcuni tipi di elementi non sono fatti per circoscrivere una zona di testo, ma solo per rappresentare qualcosa che si trova in un certo punto. Questi elementi, non vengono dichiarati con un modello di contenuto tra parentesi, ma con l'utilizzo della parola chiave '**empty**'.

L'esempio seguente, dichiara l'elemento '**toc**' che non può contenere alcunché.

```
<!ELEMENT toc - O EMPTY>
```

Tipicamente, tali elementi, sono dichiarati in modo che il marcatore di chiusura sia solo facoltativo. Non potendo contenere alcunché, sarebbe perfettamente inutile renderlo obbligatorio.

51.1.4 Dichiarazione multipla

Eventualmente, un gruppo di elementi che abbiano le stesse caratteristiche, cioè le stesse regole di minimizzazione e lo stesso modello del contenuto, può essere dichiarato in una sola istruzione. L'esempio seguente dovrebbe essere sufficiente a comprendere il meccanismo.

```
<!ELEMENT (annotazione|avvertimento|pericolo ) - - (#PCDATA) >
```

51.1.5 Attributi

Un elemento può prevedere la presenza di uno o più attributi. Si tratta di informazioni che non compongono il contenuto dell'elemento, ma di qualcosa che, non potendo apparire nel testo, serve per qualche ragione ai programmi che elaborano successivamente il documento. Il classico esempio è costituito da quei marcatori utilizzati per i riferimenti incrociati. L'esempio seguente mostra l'uso di un elemento vuoto, denominato '**ref**', contenente l'attributo '**point**' a cui viene dato il valore '**esempio**':

```
Si veda il capitolo <ref point="esempio"> che contiene
molti esempi utili al riguardo.
```

È importante osservare che il valore assegnato a un attributo deve essere delimitato attraverso apici doppi (come mostrato nell'esempio), oppure attraverso apici singoli. Eccezionalmente, è possibile assegnare un valore senza alcuna delimitazione, quando si tratta di una sola parola composta da lettere alfabetiche, cifre numeriche, trattino normale ('**-**'), trattino basso ('_'), due punti ('**:**').

L'esempio seguente mostra la dichiarazione dell'elemento '**ref**', già presentato nell'esempio, tenendo conto che il suo scopo è quello di essere utilizzato come riferimento a una parte del documento identificata attraverso il valore assegnato all'attributo '**point**'.

```
<!ELEMENT ref - O EMPTY>
<!ATTLIST ref
        point IDREF #REQUIRED
        name CDATA "riferimento">
```

Attraverso l'istruzione '**ATTLIST**' si definiscono gli attributi di un elemento. Dopo l'indicazione del nome dell'elemento a cui si fa riferimento, segue l'elenco degli attributi, ognuno dei quali inizia con un codice di interruzione di riga seguito eventualmente da altri tipi di spazi. Ciò significa che l'istruzione '**ATTLIST**' deve essere composta proprio come indicato dall'esempio, solo i rientri sono facoltativi.

L'esempio indica che l'elemento '**ref**' contiene due attributi: '**point**' e '**name**'. Il primo è obbligatorio ('**#REQUIRED**'), mentre per il secondo è stato indicato un valore predefinito, nel caso non venga specificato espressamente ('**riferimento**').

Il tipo di contenuto di un attributo viene definito attraverso delle parole chiave, che possono essere indicate usando lettere maiuscole o minuscole indifferentemente. Di seguito ne vengono descritte alcune:

Parola chiave	Descrizione
CDATA	rappresenta una stringa di qualunque tipo di carattere, ammettendo anche simboli di punteggiatura o altro, che comunque mantiene solo il suo significato letterale (*Character data*);
NMTOKEN	rappresenta qualunque tipo di carattere alfanumerico (lettere, numeri e spazi soltanto), che dovrebbe comporre un nome (*Name token*);
NUMBER	rappresenta solo cifre numeriche, cioè un numero;
ID	rappresenta un identificatore unico per quel tipo di documento, costituito da un nome senza caratteri speciali o segni di punteggiatura, che viene utilizzato successivamente per farvi riferimento;

Parola chiave	Descrizione
IDREF	indica che l'attributo deve essere un puntatore valido a un identificatore di un attributo '**ID**', corrispondente in un altro elemento.

È importante osservare che la parola chiave '**CDATA**' viene usata anche in altre situazioni con un significato simile, ma non identico. Nel caso definisca il contenuto di un attributo, è ammesso l'uso di macro (entità) che vengono espanse.

Il tipo di contenuto di un attributo, può essere indicato in modo preciso attraverso una serie di scelte alternative. In tal caso, invece di utilizzare le parole chiave già elencate, si indicano le stringhe alternative, separate dalla barra verticale, tra parentesi tonde. Per esempio, '**(bozza | finale)**' rappresenta la possibile scelta tra le due parole '**bozza**' e '**finale**'.

L'ultimo dato da inserire per ogni attributo è il valore predefinito, oppure una parola chiave a scelta tra le seguenti:

Parola chiave	Descrizione
#REQUIRED	rappresenta l'obbligatorietà dell'inserimento del valore;
#IMPLIED	rappresenta un attributo facoltativo;
#CURRENT	in mancanza di un'indicazione esplicita, rappresenta l'utilizzo dell'ultimo valore assegnato allo stesso attributo dello stesso elemento;
#FIXED	rappresenta un valore predefinito e non modificabile, che, se usato, deve avere il valore stabilito.

Nel caso particolare dell'attributo definito con la parola chiave '**#FIXED**', a questa segue necessariamente la stringa fissata.

Tra tutti, merita attenzione la coppia '**ID**' e '**IDREF**'. Questi tipi di attributi possono essere molto utili per definire dei riferimenti incrociati all'interno del documento, quando la loro validità deve essere controllata con gli strumenti di convalida SGML. Si osservi l'esempio seguente:

```
<!ELEMENT label - O EMPTY>
<!ATTLIST label
        identity ID #REQUIRED>

<!ELEMENT ref - O EMPTY>
<!ATTLIST ref
        point IDREF #REQUIRED>
```

Nell'esempio si mostra la dichiarazione di un elemento '**label**' che non può contenere testo, in quanto serve solo per definire l'attributo '**identity**', di tipo '**ID**'. Questo permette l'utilizzo di marcatori simili a '**<label identity="miaetichetta">**', dove viene assegnato all'attributo '**identity**' un nome sempre diverso, allo scopo di identificare qualcosa. Sotto, la dichiarazione dell'elemento '**ref**' mostra un altro elemento che non può contenere testo, ma solo un attributo denominato '**point**', di tipo '**IDREF**', che può quindi contenere solo il nome di un identificatore già usato in un altro elemento con l'attributo '**ID**'.

In pratica, se nel testo SGML si dovesse utilizzare da qualche parte il marcatore '**<label identity="miaetichetta">**', in un altro punto sarebbe valido il marcatore '**<ref point="miaetichetta">**', perché l'identificatore '**miaetichetta**' esiste effettivamente.

Ricapitolando, un attributo '**ID**' di un marcatore è valido quando è unico nel documento SGML che si scrive, mentre un attributo '**IDREF**' è valido quando esiste il valore corrispondente di un attributo '**ID**'.

Spesso, per cose del genere, si preferisce usare attributi di tipo '**CDATA**', per permettere l'utilizzo di caratteri di ogni tipo, togliendo però all'SGML la possibilità di controllare la validità di tali riferimenti incrociati.

51.1.6 Entità

Con questo termine, ***entità***, si fa riferimento a due tipi di oggetti: macro per la sostituzione di stringhe (entità generali) o macro per la sostituzione di nomi all'interno di istruzioni SGML (entità parametriche).

Le macro per la sostituzione di stringhe, una volta dichiarate, si utilizzano all'interno del sorgente SGML come abbreviazioni o come un modo per identificare lettere o simboli che non possono essere usati altrimenti. Per esempio, utilizzando le entità ISO 8879:1986, la frase «Wer bekommt das größte Stück Torte?» può essere scritta nel sorgente nel modo seguente:

```
Wer bekommt das gr&ouml;&szlig;te St&uuml;ck Torte?
```

Le entità generali, quindi, sono identificate nel testo SGML perché iniziano con la e-commerciale ('**&**') e terminano con un punto e virgola. È bene osservare che il punto e virgola non è obbligatorio in ogni situazione, ma solo quando il carattere successivo sia diverso da uno spazio orizzontale o da un codice di interruzione di riga. In generale, però, sarebbe bene usare sempre il punto e virgola. La tabella 51.25 elenca alcune macro delle entità standard più importanti.

Tabella 51.25. Alcune macro delle entità standard secondo le specifiche ISO 8879:1986.

á	á	Á	Á	ö	ö	Ö	Ö
â	â	Â	Â	ß	ß		
à	à	À	À	ú	ú	Ú	Ú
å	å	Å	Å	û	û	Û	Û
ã	ã	Ã	Ã	ù	ù	Ù	Ù
ä	ä	Ä	Ä	ü	ü	Ü	Ü
æ	æ	Æ	Æ	ý	ý	Ý	Ý
ç	ç	Ç	Ç	ÿ	ÿ		
é	é	É	É	&	&	@	@
ê	ê	Ê	Ê	*	*		
è	è	È	È	ˆ	^	˜	~
ë	ë	Ë	Ë	©	©		
í	í	Í	Í	$	$	%	%
î	î	Î	Î	#	#		
ì	ì	Ì	Ì	!	!	¡	¡
ï	ï	Ï	Ï	?	?	¿	¿
ñ	ñ	Ñ	Ñ	‐	-	_	_
ó	ó	Ó	Ó	\	\		
ô	ô	Ô	Ô	"	"		
ò	ò	Ò	Ò	<	<	>	>
ø	ø	Ø	Ø	[	[	]	]
õ	õ	Õ	Õ	{	{	}	}

Le entità standard ISO 8879, sono distinte in 19 gruppi, che in parte si sovrappongono (a volte si ripetono alcune dichiarazioni nello stesso modo). Questi 19 gruppi di entità corrispondono ad altrettanti file, per i quali esiste anche un nome stabilito.

L'altro tipo di macro, riguarda invece la sostituzione all'interno delle istruzioni SGML, cioè nella dichiarazione del DTD.

L'esempio seguente mostra la dichiarazione dell'elemento '**p**' che può contenere l'elemento o gli elementi indicati all'interno della macro '**%inline;**'.

```
<!ELEMENT p - O (%inline;) >
```

La dichiarazione di un'entità avviene utilizzando l'istruzione '**ENTITY**'. L'esempio seguente mostra la dichiarazione di un'entità da utilizzare nel sorgente SGML.

```
<!ENTITY agrave "\'a">
```

In questo caso, si vuole che la macro '**à**' venga sostituita con la stringa '**\'a**'. Evidentemente, questa trasformazione non ha niente a che vedere con SGML. È semplicemente una scelta motivata dal tipo di programma utilizzato successivamente per rielaborare il risultato generato dall'analizzatore SGML.

L'esempio seguente mostra la dichiarazione di due entità da utilizzare all'interno delle istruzioni SGML.

```
<!ENTITY % emph " em | concept | cparam ">
<!ENTITY % inline "(#PCDATA | %emph;)*">
```

La dichiarazione di questo tipo di entità si distingue perché viene utilizzato il simbolo di percentuale subito dopo la parola '**ENTITY**', staccandolo da questa e anche dal nome dell'entità successivo. Anche in questo caso si utilizza solo come pura sostituzione di stringhe, per cui la dichiarazione di '**%inline;**', facendo a sua volta riferimento a '**%emph;**', è equivalente a quella seguente:

```
<!ENTITY % inline "(#PCDATA | em | concept | cparam )*">
```

Naturalmente, una macro può contenere anche il riferimento a un'altra macro. Per esempio, la dichiarazione dell'ipotetico elemento '**p**', fatta nel modo seguente,

```
<!ELEMENT p - O (%inline;) >
```

è equivalente alla dichiarazione:

```
<!ELEMENT p - O ((#PCDATA | em | concept | cparam )*) >
```

51.1.6.1 Acquisizione dall'esterno

Le entità di qualunque tipo, possono essere dichiarate abbinando una stringa a una macro, come è stato mostrato in precedenza. In alternativa, a una macro si può abbinare un file esterno (file inteso nel senso più ampio possibile). In tal caso, si utilizza la parola chiave '**SYSTEM**' come nell'esempio seguente:

```
<!ENTITY capitolo2 SYSTEM "capitolo2.sgml">
```

In tal modo, quando nel documento SGML si utilizza la macro '**&capitolo2;**' e poi lo si elabora attraverso un analizzatore SGML, si ottiene l'inserimento del file '`capitolo2.sgml`'. Più o meno ciò che si fa normalmente con le direttive di un precompilatore di un linguaggio di programmazione.

Nello stesso modo si può fare per dichiarare un'entità parametrica, come nell'esempio seguente:

```
<!ENTITY % isoent SYSTEM "isoent.txt">
```

L'esempio mostra la dichiarazione della macro '**%isoent;**', riferita al file '`isoent.txt`'. Per utilizzare questa macro, bisogna sapere a cosa si riferisce; trattandosi di un file, è logico pensare che si tratti di un testo articolato su più righe, quindi inadatto all'inserzione all'interno delle istruzioni. Generalmente, una macro del genere serve a incorporare un pezzo di DTD dall'esterno.

```
%isoent;
```

Come si vede dall'esempio, è normale vedere la chiamata di una macro di questo tipo, da sola, all'esterno di qualunque istruzione del DTD. L'esempio mostrato è comunque significativo: rappresenta l'inclusione di un file che presumibilmente, dal nome, serve a incorporare le entità ISO, cioè quelle standard riferite alle lettere accentate e ai simboli speciali.

A questo proposito, potrebbero esistere diversi file, del tipo: ‘`isoent.latex.txt`’, ‘`isoent.html.txt`’,... che prima di avviare l’analizzatore SGML vengono sostituiti al file ‘`isoent.txt`’, in modo da ottenere la sostituzione corretta in base all’elaborazione successiva che si vuole ottenere (LaTeX, HTML, ecc.).

Se non fosse ancora chiaro, ecco come potrebbe essere composto l’ipotetico file ‘`isoent.txt`’ quando si vogliono le sostituzioni corrette per LaTeX.

```
<!ENTITY agrave "\‘a">
<!ENTITY Agrave "\‘A">
<!ENTITY egrave "\‘e">
<!ENTITY Egrave "\‘E">
<!ENTITY eacute "\’e">
<!ENTITY Eacute "\’E">
…
```

L’acquisizione di una macro da un file esterno può essere dichiarata senza specificare esplicitamente il file, lasciando che l’analizzatore trovi il file corretto in base a un catalogo SGML. L’argomento viene ripreso in seguito, comunque, in questo tipo di dichiarazione, manca il nome del file.

```
<!ENTITY capitolo2 SYSTEM>
```

```
<!ENTITY % isoent SYSTEM>
```

Solitamente, si preferisce includere in questo modo solo le macro parametriche, cosa che può essere compresa intuitivamente in seguito.

51.1.6.2 Codici macro speciali

È bene ribadire che l’uso delle entità standard (ISO), permette di rendere il testo SGML indipendente dalla piattaforma utilizzata. Tuttavia, la dichiarazione della sostituzione dipende dalla piattaforma e, come si è mostrato, si tendono a predisporre diversi schemi di sostituzione per le diverse piattaforme a cui si vuole fare riferimento.

In situazioni eccezionali, può essere conveniente indicare i caratteri per numero, decimale o esadecimale, attraverso una notazione simile a quella delle entità normali. Per esempio, se si usa la codifica ISO 8859-1 (Latin 1), la macro ‘**`&#232;`**’, oppure la macro ‘**`&#xe8;`**’ corrisponde alla lettera ‘**è**’ (la «e» accentata normale).

Questa possibilità è fondamentale proprio quando si definiscono le stringhe di sostituzione per una piattaforma determinata (hardware-software), in cui si debbano indicare caratteri speciali identificati dal numero corrispondente.

```
<!ENTITY egrave "&#232;">
```

Potrebbe sembrare che un testo SGML non possa utilizzare una codifica particolare, quale ISO 8859-1 o altro. Non è così. L’SGML mette a disposizione le entità standard, ma ciò non toglie che si possa decidere di usare comunque una codifica (ASCII) estesa come Latin 1 o altro. Ovviamente questo rende il testo dipendente dalla piattaforma, precisamente dalla codifica. Volendo essere precisi, la codifica utilizzabile dipende dalla dichiarazione SGML, cosa che viene descritta nella sezione 51.3.

51.1.7 Sezioni marcate

Le sezioni marcate sono una specialità di SGML, poco usata e poco conosciuta. Si tratta di istruzioni che vengono inserite nel testo SGML (non nel DTD) e servono a vario titolo per delimitare del testo per qualche scopo.

Una sezione marcata si compone di un sorta di marcatore di apertura e di una sorta di marcatore di chiusura. Il marcatore di apertura contiene una parola chiave che ne identifica il comportamento. Si osservi l’esempio seguente:

```
<![INCLUDE[
Questa parte del testo è inclusa nell’elaborazione SGML.
]]>
```


Come si può intuire, la sezione marcata dell’esempio è introdotta da ‘**`<![INCLUDE[`**’ ed è terminata da ‘**`]]>`**’. In questo caso, la parola chiave ‘**`INCLUDE`**’ indica che il testo contenuto nella sezione marcata deve essere incluso nell’elaborazione (anche se ciò, per ora, può sembrare perfettamente senza significato).

Le parole chiave utilizzabili per definire la sezione marcata sono diverse; di seguito ne appare l’elenco.

Parola chiave	Descrizione
`INCLUDE`	Il contenuto della sezione marcata deve essere incluso nel documento SGML e deve essere elaborato normalmente.
`IGNORE`	Il contenuto della sezione marcata deve essere escluso dal documento SGML. Se l’analizzatore SGML genera un qualche tipo di output, questo non contiene tale sezione.
`CDATA`	Il contenuto della sezione marcata deve essere incluso e trattato come testo letterale, in modo da ignorare ciò che altrimenti potrebbe essere interpretato come un marcatore o un’entità. Ciò vale per tutto, tranne il simbolo di chiusura della sezione marcata (‘**`]]>`**’), che quindi è l’unica cosa che non può essere rappresentata all’interno di questa.
`RCDATA`	Il contenuto della sezione marcata deve essere incluso e trattato come testo letterale, in modo da ignorare ciò che altrimenti potrebbe essere interpretato come un marcatore, ma continuando a espandere le entità.
`TEMP`	Il contenuto della sezione marcata deve essere inteso come «temporaneo». Ciò serve solo come riferimento umano, per localizzare facilmente una parte del documento che richiede una revisione o che deve essere rimossa.

L’utilizzo di sezioni marcate di tipo ‘**`INCLUDE`**’ e ‘**`IGNORE`**’ è utile solo in abbinamento a entità parametriche. Prima di proseguire, è bene chiarire che quella specie di marcatore che apre una sezione marcata è come un’istruzione SGML, di quelle che appaiono nel DTD, anche se viene usata al di fuori di questo, nel documento. In questo senso, al suo interno si possono usare le entità parametriche; quindi, una di queste macro può servire per definire in modo dinamico la parola chiave ‘**`INCLUDE`**’ oppure ‘**`IGNORE`**’, per decidere di includere o escludere quel blocco (e probabilmente anche altri) con la modifica di una sola macro.

Per esempio, nel DTD del documento potrebbe apparire la dichiarazione di un’entità parametrica denominata ‘**`commentato`**’.

```
<!ENTITY % commentato "INCLUDE">
```

Nel documento SGML potrebbero esserci una serie di sezioni marcate la cui inclusione deve dipendere da questa macro.

```
…
1 + 2 = 3
<![%commentato;[
La matematica non è un’opinione.
]]>
…
```

Quando il testo viene analizzato, la macro viene espansa e trovando che corrisponde a ‘**`INCLUDE`**’, il testo delle sezioni marcate che l’hanno usata, vengono incluse. Al contrario, basta modificare la macro, assegnandole il valore ‘**`IGNORE`**’, per fare in modo che tutte quelle sezioni marcate vengano ignorate.

Questo tipo di approccio potrebbe sembrare ugualmente scomodo per l’utilizzatore che non vuole toccare il DTD. Però, come è possibile vedere in seguito, si possono inserire delle eccezioni al DTD nel preambolo di un documento SGML. Oppure, si può benissimo progettare un DTD con una componente esterna, destinata a questo tipo di ritocchi.

51.1.8 Dettagli importanti

Prima di passare alla descrizione dell'abbinamento di un DTD a un testo SGML, è bene chiarire alcuni dettagli che sono stati trascurati fino a questo punto.

51.1.8.1 Commenti

All'interno del documento sorgente SGML, come nel DTD, possono essere indicate delle righe di commento da non considerare parte del documento o della codifica. Queste si ottengono con i delimitatori '**<!--**' e '**-->**'.

Volendo approfondire meglio il problema, la sequenza '**<!>**' rappresenta l'istruzione SGML nulla e può essere usata indifferentemente nel DTD o nel sorgente SGML. In qualità di istruzione nulla viene ignorata semplicemente.

All'interno delle istruzioni SGML è possibile inserire dei commenti, attraverso una sequenza di due trattini ('**--**'), per aprire e chiudere il commento. L'esempio seguente dichiara l'elemento '**itemize**' con un commento incorporato:

```
<!ELEMENT itemize - - (item+) -- elenchi puntati -- >
```

Ciò dovrebbe chiarire il senso del commento composto da '**<!--**' e '**-->**': si tratta di un'istruzione (nulla) che contiene un commento.

Questa particolarità di SGML ha delle conseguente: nel testo che compone il commento, non possono apparire sequenze di due o più trattini.

51.1.8.2 Maiuscole e minuscole

Per convenzione, i nomi di entità sono sensibili alla differenza tra lettere maiuscole e minuscole, per cui '**À**' e '**à**' rappresentano rispettivamente la lettera «A» maiuscola con accento grave e la «a» minuscola con accento grave.

Per convenzione, i nomi degli elementi, i simboli delle regole di minimizzazione, i nomi degli attributi e le parole chiave, non sono sensibili alla differenza tra lettere maiuscole e minuscole. Quindi, nell'ambito delle dichiarazioni del DTD i due esempi seguenti sono identici:

```
<!element ref - o empty>
<!attlist ref
        id cdata #required
        name cdata "riferimento">
```

```
<!ELEMENT REF - O EMPTY>
<!ATTLIST REF
        ID CDATA #REQUIRED
        NAME CDATA "riferimento">
```

Nello stesso modo, nell'ambito del testo SGML sono identici i due esempi seguenti:

```
... <ref id="capitolo-introduttivo" name="Intro"> ...
```

```
... <REF id="capitolo-introduttivo" name="Intro"> ...
```

indifferentemente dal modo (maiuscolo o minuscolo) in cui l'elemento '**ref**' è stato dichiarato nel DTD.

Evidentemente, in generale, il contenuto delle stringhe delimitate è sensibile alla differenza tra maiuscole e minuscole, dal momento che riguarda i programmi che fanno uso del documento dopo l'analisi SGML; in pratica, dipende da questi programmi successivi il senso che hanno tali informazioni.

51.1.8.3 Delimitatori di stringa

In varie situazioni, all'interno del DTD e all'interno dei marcatori utilizzati nel testo SGML, può essere necessaria l'indicazione di stringhe. I simboli utilizzati per delimitare le stringhe possono essere gli apici doppi ('**"**…**"**'), oppure gli apici singoli ('**'**…**'**'). La scelta tra i due tipi di delimitatori dovrebbe essere indifferente, a parte la possibile necessità di inserire nelle stringhe proprio questi caratteri. Si osservi l'esempio seguente, in cui vengono dichiarate le entità riferite ad alcune lettere accentate da usare con LaTeX.


```
<!ENTITY uuml   '\"u'>
<!ENTITY Uuml   '\"U'>
<!ENTITY yacute "\'y">
<!ENTITY Yacute "\'Y">
```

In situazioni più complesse, potrebbe essere necessario indicare i caratteri con l'aiuto delle macro '**&#*nnn*;**', che permettono di identificare l'oggetto attraverso il numero corrispondente riferito al tipo di codifica utilizzato (purché il contesto preveda la successiva ulteriore espansione di tali macro).

51.1.8.4 Tipo di contenuto di un'entità generale

In precedenza, quando è stato mostrato in che modo possa essere definita un'entità, si è trascurato il fatto che si deve definire in che modo la stringa di sostituzione vada interpretata. Per questo, si aggiunge una parola chiave prima della stringa.

Se non si usa alcuna parola chiave, si intende che la stringa vada interpretata come appare, espandendo eventuali entità contenute al suo interno. Si osservi l'esempio.

```
<!ENTITY attenzione "&lt;ATTENZIONE&gt;">
```

Quando dovesse essere utilizzata la macro '**&attenzione;**', si otterrebbe la stringa '**<ATTENZIONE>**', perché le entità '**<**' e '**>**' vengono espanse ulteriormente.

Se si indica la parola chiave '**CDATA**', si intende che la stringa di sostituzione deve essere utilizzata in modo letterale, senza espandere alcuna sequenza che potrebbe sembrare un'entità.

```
<!ENTITY attenzione CDATA "&lt;ATTENZIONE&gt;">
```

L'esempio, modificato con l'introduzione della parola chiave '**CDATA**', fa sì che la macro '**&attenzione;**' si traduca in pratica in '**<ATTENZIONE>**', perché le entità '**<**' e '**>**' non vengono riconosciute come tali e quindi non vengono espanse.

Se si utilizza la parola chiave '**SDATA**' (*Special data*), si intende che la stringa di sostituzione deve essere utilizzata in modo letterale, senza espandere alcuna sequenza che potrebbe sembrare un'entità. Però, a differenza di '**CDATA**', l'informazione viene filtrata in modo particolare quando l'analizzatore SGML genera un risultato transitorio da riutilizzare con un altro programma di composizione.

51.1.8.5 Contenuto elementare degli elementi

In precedenza è già stata spiegata la dichiarazione degli elementi e la dichiarazione del contenuto. In particolare si è visto che attraverso la parola chiave '**#PCDATA**' si fa riferimento a testo normale che viene elaborato normalmente (*parsed*). Ciò significa che questo tipo di testo è soggetto alla sostituzione delle entità, come fino a questo punto si è dato per scontato.

Tuttavia esistono altre parole chiave per definire tipi di testo differenti. Segue l'elenco di quelle più comuni.

Parola chiave	Descrizione
`#PCDATA`	*Parsed character data*. Si riferisce a testo normale soggetto alla sostituzione delle entità. Questo testo non può contenere altri elementi.
`CDATA`	Il contenuto dell'elemento deve essere trattato come testo letterale, in modo da ignorare ciò che altrimenti potrebbe essere interpretato come un marcatore o un'entità. In pratica, la definizione di elementi con contenuto '**CDATA**' è decisamente sconsigliabile. Se esiste la necessità di delimitare una zona di testo da trattare in modo letterale, solitamente, si preferisce utilizzare una sezione marcata del tipo '**<![CDATA[…]]>**'.
`RCDATA`	Il contenuto dell'elemento deve essere trattato come testo letterale, in modo da ignorare ciò che altrimenti potrebbe essere interpretato come un marcatore, ma continuando a espandere le entità.

51.1.9 Abbinare il DTD al documento SGML

L'abbinamento di un DTD a un documento SGML avviene generalmente in modo formale. In presenza di situazioni eccezionali, questo abbinamento può essere implicito, come nel caso dell'HTML, ma è bene utilizzare ugualmente l'approccio generale anche in questi casi estremi.

Un sorgente SGML inizia normalmente con la dichiarazione del tipo di DTD utilizzato. Può trattarsi di un file esterno o di dichiarazioni incorporate nel documento stesso. Per esempio, la dichiarazione seguente indica all'analizzatore SGML di utilizzare un DTD esterno, denominato '**`linuxdoc`**' e contenuto nel file '`linuxdoc.dtd`'.

```
<!DOCTYPE linuxdoc SYSTEM "linuxdoc.dtd">
```

L'esempio seguente mostra invece una dichiarazione iniziale che contiene le istruzioni che compongono il DTD, racchiuse tra parentesi quadre.

```
<!DOCTYPE personale [
...
-- istruzioni SGML --
...
...
]>
```

Una terza possibilità permette di definire un file esterno e di aggiungere altre istruzioni particolari riferite al documento, come nell'esempio seguente, sempre utilizzando le parentesi quadre.

```
<!DOCTYPE linuxdoc SYSTEM "linuxdoc.dtd" [
...
-- istruzioni SGML --
...
...
]>
```

Inoltre, come è stato visto nel caso delle entità, l'acquisizione dall'esterno di un file contenente un DTD, può avvenire anche senza stabilire espressamente il nome di un file, lasciando che questo venga determinato da un catalogo. Così, l'esempio già visto del DTD '**`linuxdoc`**' si potrebbe trasformare nel modo seguente:

```
<!DOCTYPE linuxdoc SYSTEM>
```

Esiste anche un'altra alternativa: quella di indicare un identificatore pubblico, anch'esso riferito a un catalogo. Quello che segue è il preambolo di un file SGML scritto secondo il DTD HTML 3.2. Si osservi, a questo proposito, l'uso della parola chiave '**`PUBLIC`**'.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
```

51.1.9.1 Strategie

La scelta di incorporare il DTD nel documento, o di lasciarlo all'esterno, dipende da delle preferenze organizzative. Di sicuro, può essere sensato l'inclusione del DTD nel documento SGML quando si tratta di un DTD specifico che non viene usato altrove.

Nella realtà, si utilizza quasi sempre un DTD esterno, probabilmente predisposto da altri, abbinato a una serie di strumenti che permettono di produrre dei documenti in formato finale a partire dai sorgenti SGML scritti seguendo quel DTD particolare.

L'estensibilità del DTD resta sempre una possibilità utile per poter aggiungere delle entità generali (interne o esterne) o delle entità parametriche allo scopo di gestire opportunamente le sezioni marcate.

51.1.9.2 Entità generali

Come si è visto nella sezione precedente, la dichiarazione del DTD può includere delle istruzioni del DTD, generalmente estendendolo. Questo meccanismo permette, tra le altre cose, di inserire delle entità generali, interne o esterne. Si osservi l'esempio:


```
<!DOCTYPE linuxdoc SYSTEM
[
<!ENTITY pericolo "!!">
<!ENTITY posix    "POSIX">
<!ENTITY unix     "Unix">
<!ENTITY xwin     "X Window System">
<!ENTITY edizione "1999.12.31">
]>
```

L'esempio appena mostrato permette di utilizzare le macro '**`&pericolo;`**', '**`&posix;`**', '**`&unix;`**', '**`&xwin;`**' e '**`&edizione;`**', all'interno del sorgente SGML, ottenendo la loro sostituzione automatica. Per intenderne l'utilità, basta pensare al caso della macro '**`&xwin;`**' dell'esempio precedente: non occorre più ricordare come si deve scrivere («X», «X Window» o «X Window System»); se si decidesse di cambiare, basterebbe modificare la dichiarazione dell'entità. Il concetto è analogo a quello delle macro del precompilatore nei linguaggi di programmazione.

L'esempio fa riferimento in particolare a entità interne, perché tutto quello che serve è contenuto nella dichiarazione, mentre sarebbero entità esterne se si fa riferimento a file esterni per l'espansione del loro contenuto.

Segue un esempio con l'uso di un'entità esterna:

```
<!DOCTYPE linuxdoc SYSTEM
[
<!ENTITY COPY         SYSTEM "formalita/copy.sgml">
]>
```

La definizione di entità generali è consentita anche quando queste dovessero essere già state dichiarate nel DTD. Le entità dichiarate nelle istruzioni aggiuntive, dovrebbero prendere la precedenza e sostituirsi a quelle eventualmente già dichiarate nel DTD.

51.1.9.3 Entità parametriche

Come nel caso delle entità generali, nelle estensioni del DTD può essere conveniente aggiungere la dichiarazione di entità parametriche da utilizzare per controllare l'inclusione o l'esclusione di sezioni marcate. Si osservi l'esempio, già mostrato in precedenza, in cui la macro '**`commentato`**' serve per controllare alcune sezioni marcate, stabilendone il tipo.

```
<!DOCTYPE linuxdoc SYSTEM
[
<!ENTITY pericolo "!!">
<!ENTITY posix    "POSIX">
<!ENTITY unix     "Unix">
<!ENTITY xwin     "X Window System">
<!ENTITY edizione "1999.12.31">
...
<!ENTITY % commentato "INCLUDE">
]>
...
...
1 + 2 = 3
<![%commentato;[
La matematica non è un'opinione.
]]>
...
...
```

51.1.9.4 Ultime annotazioni sulla dichiarazione del DTD

Si è visto in che modo inizia un sorgente SGML. Quello che non è ancora stato chiarito è che il tipo di documento deve essere stato dichiarato nel DTD, anche se ciò può sembrare ridondante. In effetti è necessario dire di cosa è composto il documento. Nel DTD potrebbe apparire un'istruzione come quella seguente:

```
<!ELEMENT linuxdoc O O (article|report|book|letter) >
```

In questo esempio, si comprende che non è necessario usare marcatori del tipo '**`<linuxdoc>`**' '**`</linuxdoc>`**' per delimitare il sorgente SGML. Infatti, la coppia di '**`O`**' afferma che queste sono opzionali.

Invece, il tipo di documento '**linuxdoc**' deve contenere esattamente un elemento del tipo '**article**', oppure '**report**', oppure '**book**', o ancora '**letter**'.

Il sorgente SGML che fa riferimento al tipo di documento '**linuxdoc**' e che utilizza il formato definito dall'elemento '**article**', è composto schematicamente come segue:

```
<!DOCTYPE linuxdoc SYSTEM>
<article>
...
...
...
</article>
```

Un tipo di documento potrebbe essere definito in maniera diversa, per esempio nel modo seguente:

```
<!element miodoc - - ( sezione+ ) >
```

In questo caso, il documento può contenere solo elementi '**sezione**', ed è obbligatorio l'utilizzo dei marcatori per indicare l'inizio e la fine del tipo di documento.

```
<!doctype miodoc system>
<miodoc>
        <sezione>
        ...
        ...
        ...
</miodoc>
```

51.1.10 Mappe di sostituzione (shortref)

Fino a questo punto, si è vista la filosofia dell'SGML applicata alla struttura del documento e all'indipendenza rispetto alla piattaforma. L'analizzatore SGML standard, oltre che convalidare il documento in base al DTD, si occupa di rielaborare il sorgente SGML per generare un risultato intermedio, più facile da gestire per altri programmi di composizione.

In un certo qual modo, questo risultato intermedio può essere controllato, all'interno del DTD, attraverso la definizione di mappe di sostituzione, o *shortref*.

Con questo meccanismo, si punta normalmente ad attribuire significati speciali a simboli determinati, oltre che a controllare la spaziatura orizzontale e verticale del testo.

51.1.10.1 Dichiarazione e abbinamento delle mappe di sostituzione

La mappa di sostituzione definisce un abbinamento tra un simbolo e un'entità che ne deve prendere il posto. L'esempio seguente è solo un pezzo ipotetico della dichiarazione di una mappa del genere.

```
<!SHORTREF miamappa
...
        "[" lsqb
        "]" rsqb
        "~" nbsp
        "_" lowbar
        "#" num
        "%" percnt
        "^" circ
        "{" lcub
        "}" rcub
        "|" verbar >
```

Dall'esempio si può osservare che alcuni simboli vengono sostituiti con le entità relative, indicate solo per nome, senza bisogno della e-commerciale e del punto e virgola finale. Questo fatto, di per sé, potrebbe sembrare assolutamente inutile dal punto di vista di SGML: se si può scrivere una parentesi quadra aperta, perché sostituirla automaticamente con la sua entità corrispondente. Il fatto è che il software utilizzato per la composizione, potrebbe attribuire un significato speciale a una parentesi quadra, mentre quello che si vuole nel testo SGML è che questa valga solo per quello che appare. In tal modo, chi scrive dovrebbe utilizzare necessariamente la macro

'**[**' per non creare problemi al programma di composizione o di elaborazione successiva.

Nello stesso modo, attraverso la mappa di sostituzione, si può attribuire un significato completamente diverso alla parentesi quadra aperta: per assurdo, potrebbe diventare una parentesi graffa...

```
<!SHORTREF miamappa
...
        "[" lcub
        "]" rcub
...
        "{" lcub
        "}" rcub
        "|" verbar >
```

Volendo fare delle acrobazie, si può associare un simbolo a un'entità che poi si traduce in un marcatore. Si osservi l'esempio.

```
<!ENTITY formula1 '<formula>'>
<!ENTITY formula0 '</formula>'>
<!SHORTREF miamappa
...
        "[" formula1
        "]" formula0
...
        "{" lcub
        "}" rcub
        "|" verbar >
```

In questo modo, quando nel testo si utilizzano le parentesi quadre, ciò che si ottiene è l'apertura e la chiusura dell'elemento '**formula**'.

Anche se questa tecnica è stata usata nel noto DTD LinuxDoc, come in Qwertz, proprio per delimitare agevolmente le formule matematiche, si tratta di una cosa decisamente sconsigliabile dal punto di vista dell'SGML.

Gli elementi SGML vanno abbinati alle mappe che si ritiene siano più adatte per i loro scopi. Tuttavia, un elemento può non essere stato abbinato esplicitamente ad alcuna mappa; in tal caso eredita quella dell'elemento che lo contiene effettivamente, di volta in volta, nel documento. Di conseguenza, diventa importante abbinare esplicitamente una mappa almeno all'elemento più esterno, ovvero a quello che corrisponde al nome del tipo stesso di documento.

Dagli esempi mostrati, si può notare che la mappa ha un nome, indicato subito dopo la parola chiave '**SHORTREF**' che apre il comando. Se si vuole abbinare la mappa '**miamappa**' all'elemento '**acronimo**', si procede come nell'esempio seguente:

```
<!USEMAP miamappa acronimo>
```

51.1.10.2 Spaziature e interruzioni di riga

In linea di principio, il risultato dell'elaborazione dell'analizzatore SGML contiene tutti gli spazi orizzontali e verticali esistenti nel sorgente di partenza. Però, per quanto possibile, si cerca normalmente di evitare che il sorgente SGML sia vincolato dalla spaziatura utilizzata, che in realtà potrebbe servire solo per facilitarne la lettura umana con rientri, allineamenti, spazi verticali come si farebbe con un linguaggio di programmazione.

Per questo ci deve essere un modo per poter identificare le spaziature orizzontali, le righe vuote e quelle bianche, in modo da poterle sopprimere nel risultato dell'elaborazione SGML. Naturalmente, bisogna poter distinguere, perché ci sono situazioni in cui gli spazi e le righe vuote hanno un significato e vanno mantenuti.

Per queste cose si utilizzano delle macro speciali, ma prima di descriverle, occorre definire alcuni concetti. Dal punto di vista dell'SGML, una riga è una sequenza di caratteri, con un inizio e una fine, ignorando completamente la codifica che si utilizza in pratica per separare una riga dall'altra.

Nei sistemi Unix, il codice di interruzione di riga è composto dal carattere *<LF>* mentre in altri sistemi si utilizza la sequenza *<CR><LF>*. Per l'SGML è come se questi codici non esistessero:

le righe finiscono **prima** del codice di interruzione di riga e iniziano **dopo** tale codice. Si osservi l'esempio seguente:

```
<paragrafo>Ciao,
come stai?
Io bene; e tu?</paragrafo>
```

L'idea che ha l'SGML di ciò che è stato scritto, può essere rappresentata dallo schema seguente, dove è stato utilizzato il simbolo '**^**' per segnalare l'inizio della riga, il simbolo '**$**' per segnalarne la fine, i simboli '**>**' e '**<**' per indicare l'inizio e la fine dell'elemento.

```
>Ciao,$
^come stai?$
^Io bene; e tu?<
```

Può sembrare strano, ma all'inizio e alla fine del testo mancano questi margini: esiste solo l'inizio e la fine dell'elemento. Se si dovesse sopprimere una riga, si eliminerebbe implicitamente anche il suo inizio e la sua fine.

Da qualche parte si potrebbe leggere che il codice di inizio riga equivale al codice *<LF>*, mentre quello di fine riga corrisponde a *<CR>*. Evidentemente questo ragionamento può valere solo per le piattaforme che utilizzano file di testo con un'interruzione di riga *<CR><LF>*, ma si tratta di una semplificazione che non corrisponde alla logica di SGML e può essere solo forviante.

La tabella 51.70 mostra le macro più importanti che possono essere usate per il controllo delle spaziature superflue.

Tabella 51.70. Simboli di definizione di spaziature e delimitazione delle righe.

Simbolo	Significato
`&#RS;`	Inizio di una riga (*Record start*).
`&#RE;`	Fine di una riga (*Record end*).
`&#RS;B`	Spaziatura iniziale.
`B&#RE;`	Spaziatura finale.
`&#RS;&#RE;`	Una riga vuota.
`&#RS;B&#RE;`	Una riga contenente solo spazi orizzontali (bianca).
`&#SPACE;`	Uno spazio singolo, [*SP*].
`&#TAB;`	Tabulazione, [*HT*].
`BB`	Spazio orizzontale all'interno e agli estremi dell'elemento.

L'esempio seguente mostra una mappa di sostituzione tipica, in cui si vogliono ignorare (e di conseguenza, eliminare) gli spazi orizzontali superflui, le righe vuote, quelle bianche, infine si vuole che tutto il testo si traduca in una riga sola.

```
<!shortref miamappa
        "BB"            space
        "&#RS;B"        null
        "B&#RE;"        space
        "&#RS;B&#RE;"   null
        "&#RS;&#RE;"    null
        "&#RS;"         null
        "&#RE;"         space
        "[" lsqb
        "]" rsqb
        "~" nbsp
        "_" lowbar
        "#" num
        "%" percnt
        "^" circ
        "{" lcub
        "}" rcub
        "|" verbar >
```

Le macro '**&space;**' e '**&null;**' si riferiscono rispettivamente a un solo carattere spazio e alla stringa nulla. Generalmente devono essere dichiarate nel DTD nel modo seguente:


```
<!ENTITY space " ">
<!ENTITY null "">
```

Per comprendere meglio l'effetto della mappa di sostituzione proposta, conviene partire da un esempio e analizzare gli effetti di ogni dichiarazione, una alla volta. In particolare, gli utenti dei sistemi Unix devono dimenticare per un po' il comportamento del codice di interruzione di riga (*new-line*), perché SGML considera solo la stringa nulla all'inizio e alla fine della riga: solo quando la fine di una riga e l'inizio della successiva sono stati rimossi, allora queste due vengono unite assieme.

Supponiamo di cominciare da una variante dell'esempio già descritto, dove sono stati aggiunti tanti spazi orizzontali e verticali superflui.

```
>     Ciao,      $
^$
^         $
^come stai?$
^$
^     Io         bene;         e      tu?     <
```

Applicando la trasformazione '**"BB" space**', vengono sostituiti gli spazi orizzontali all'inizio dell'elemento, alla fine e all'interno delle frasi con uno spazio singolo normale.

```
> Ciao,      $
^$
^         $
^come stai?$
^$
^     Io bene; e tu? <
```

Si può osservare che le frasi si sono ricompattate; inoltre, all'inizio e alla fine dell'elemento è rimasto un solo spazio superfluo (che non può essere rimosso). Si continua applicando '**"&#RS;B" null**'; si ottiene l'eliminazione dell'inizio delle righe (quelle che contengono effettivamente qualcosa) fino al primo carattere diverso da uno spazio orizzontale.

```
> Ciao,      $
^$
^         $
^come stai?$
^$
 Io bene; e tu? <
```

Quando si applica anche '**"B&#RE;" space**'; si ottiene la sostituzione degli spazi orizzontali nella parte finale, fino alla fine delle righe (quelle che contengono effettivamente qualcosa), con uno spazio singolo. Nell'esempio, dal momento che nella prima riga è scomparso il simbolo che segnalava la fine della riga, appare un trattino basso, ma solo per aiutare il lettore.

```
> Ciao,_
^$
^         $
^come stai?$
^$
 Io bene; e tu? <
```

La sostituzione '**"&#RS;B&#RE;" null**' elimina le righe bianche, ma non vuote.

```
> Ciao,_
^$
^come stai?$
^$
 Io bene; e tu? <
```

La sostituzione '**"&#RS;&#RE;" null**' elimina le righe vuote.

```
> Ciao,_
^come stai?$
 Io bene; e tu? <
```

Si può osservare che la riga contenente la frase «come stai?», è rimasta intatta. Infatti, non contenendo spazi aggiuntivi all'inizio o alla

fine, non è mai stata interessata dalle trasformazioni applicate fino a questo momento.

Finalmente entrano in gioco '**"&#RS; null**' e '**"&#RE; space**', per eliminare l'inizio e la fine delle righe rimaste. Per la precisione, la fine delle righe deve essere sostituito con uno spazio singolo, altrimenti si rischia di attaccare assieme delle parole. La trasformazione viene mostrata in due passaggi.

```
> Ciao,_
 come stai?_
 Io bene; e tu? <

> Ciao, come stai? Io bene; e tu? <
```

Nonostante la descrizione fatta con tanta cura, è probabile che la trasformazione '**"&#RS; null**' venga semplicemente ignorata, perché l'analizzatore SGML si limita a tenere in considerazione solo la fine delle righe (*record end*).

51.1.10.3 Limitazioni ed esagerazioni

Da quanto visto nella sezione precedente si potrebbe supporre che il meccanismo delle mappe di sostituzione permetta di sostituire quello che si vuole. Non è così, solo alcuni simboli sono considerati dei possibili *shortref*. In ogni caso, ci si accorge subito quando si usa qualcosa di sbagliato: l'analizzatore SGML avvisa immediatamente.

Attraverso le mappe di sostituzione si possono realizzare anche delle acrobazie che spesso sono poco giustificabili e che sarebbe meglio evitare. A parere di chi scrive, la cosa meno utile che si possa richiedere a un sistema SGML è quella di fare in modo che le righe vuote e quelle bianche nel sorgente siano trasformate in separazioni tra i paragrafi. Infatti, l'SGML non ha questo scopo, eppure molti sistemi si impegnano in questo senso. LinuxDoc raggiunge questo risultato intervenendo proprio nelle mappe di sostituzione, facendo in modo che le righe bianche, identificate dal simbolo '**&#RS;B&#RE;**', e quelle vuote, identificate dal simbolo '**&#RS;&#RE;**', siano sostituite da '** **', ovvero dai marcatori che servono a chiudere e a riaprire un paragrafo.

```
...
<!ENTITY psplit '</p><p>' >
...
<!SHORTREF pmap
        "&#RS;B" null
        "&#RS;B&#RE;" psplit
        "&#RS;&#RE;" psplit
...
        "{" lcub
        "}" rcub
        "|" verbar >
...
```

Quello che si vede sopra è proprio un estratto dal DTD di LinuxDoc, dove si vede che la macro '**&psplit;**' viene poi rimpiazzata dai marcatori già descritti.

Naturalmente, questo non esclude la possibilità di generare una grande quantità di elementi '**p**' vuoti, in presenza di più righe vuote o bianche. È chiaro che, successivamente, il sistema di composizione utilizzato deve prendersi carico della loro eliminazione.

51.1.10.4 Soluzione normale

Dopo aver visto in quanti modi si possono usare le mappe di sostituzione, vale la pena di mostrare una soluzione «normale», in cui il problema che si vuole risolvere è l'eliminazione degli spazi superflui all'inizio e alla fine delle righe, oltre che l'eliminazione delle righe bianche e quelle vuote:


```
<!ENTITY space " ">
<!ENTITY null "">
<!ENTITY recordstart "&#RS;">
<!ENTITY recordend "&#RE;">

<!SHORTREF standard
        "&#RS;B"        recordstart
        "B&#RE;"        recordend
        "&#RS;B&#RE;"   null
        "&#RS;&#RE;"    null
>
```

In questo modo, come si vede, è stato necessario dichiarare due entità nuove, '**recordstart**' e '**recordend**', per poter sopprimere gli spazi iniziali e finali superflui, pur mantenendo la separazione in righe distinte.

51.1.11 Elementi di testo riportato letteralmente

La predisposizione di un elemento SGML che consenta la scrittura di testo da riportare in modo letterale costituisce un problema. Si possono scegliere soluzioni diverse, ma nessuna perfetta secondo tutti i punti di vista.

Questo tipo di problema è particolarmente sentito nella scrittura di documenti tecnici, in cui ci può essere la necessità di mostrare porzioni di codice scritto in un qualche linguaggio di programmazione. Per evitare che simboli determinati vengano interpretati dall'analizzatore SGML, occorrerebbe utilizzare continuamente delle macro alternative.

Si possono seguire due direzioni per cercare di risolvere il problema: l'uso di elementi predisposti per un tipo di contenuto più o meno letterale, oppure l'uso di elementi normali con l'aggiunta di una sezione marcata di tipo '**CDATA**'.

51.1.11.1 Tipo di contenuto letterale

Nella definizione di un elemento occorre stabilite il tipo di contenuto. A livello elementare, quando l'elemento non può contenere altri elementi, si utilizza normalmente la parola chiave '**#PCDATA**', con cui si fa riferimento a testo che viene analizzato alla ricerca di entità generali da espandere, senza ammette altri elementi al suo interno.

Per ottenere un elemento adatto al contenuto letterale, si usa solitamente il tipo di contenuto definito dalla parola chiave '**RCDATA**', che non è perfettamente letterale, ma vi si avvicina molto. Per la precisione, la forma del testo viene mantenuta, con tutte le sue spaziature e le interruzioni di riga, ma le entità generali vengono espanse, mentre vengono ignorati eventuali marcatori di apertura. Ciò significa che, la e-commerciale ('**&**') non può essere usata in modo letterale, a meno di usare una macro adatta al suo posto. Lo stesso ragionamento riguarda la sequenza di minore e barra obliqua ('**</**'), che è ammessa solo nel marcatore di chiusura di questo elemento.

```
<!ELEMENT formattato - - RCDATA>
```

L'esempio mostra la dichiarazione dell'elemento '**formattato**', di tipo '**RCDATA**'. Generalmente, per poter utilizzare questo elemento nel modo corretto, si devono dichiarare anche due entità generali specifiche.

```
<!ENTITY ero   CDATA "&">
<!ENTITY etago '</' >
```

In tal modo, al posto del simbolo '**&**' si deve utilizzare la macro '**&ero;**', mentre al posto della sequenza '**</**', si deve usare la macro '**&etago;**'. È il caso di osservare che l'entità generale '**ero**' è volutamente diversa da un'entità analoga, necessaria a indicare una e-commerciale in un testo normale. Infatti, in questo caso, si vuole generare un testo letterale, che si presume possa essere interpretato nello stesso modo letterale anche da altro software di composizione successivo.

In alternativa, si potrebbe usare anche un tipo di contenuto definito dalla parola chiave '**CDATA**', che dovrebbe essere in grado di ignorare sia i simboli dei marcatori, che le macro delle entità generali.

Di fatto però, questo tipo di elemento non dà normalmente i risultati sperati.

51.1.11.2 Sezioni marcate

Nel sorgente SGML, all'interno di un elemento che non sia stato predisposto per un contenuto letterale, è possibile inserire una sezione marcata di tipo '**CDATA**', come nell'esempio seguente:

```
<![CDATA[
Testo letterale: &amp;, &etago;, <ciao>, </ciao>,
ecc., vengono trattati in modo letterale.
]]>
```

In tal modo, vengono preservati anche gli spazi, orizzontali e verticali, e ogni eventuale mappa di sostituzione (*shortref*) viene ignorata temporaneamente. L'unica cosa che non può contenere questo ambiente, è la sequenza '**]]>**', che serve a concludere la sezione marcata.

Bisogna tenere presente che la sequenza '**]]>**' può essere rappresentata anche con l'inserzione di spazi; per esempio come '**]]>**', '**]] >**' o '**]] >**', che rappresentano sempre la stessa cosa.

Questa tecnica ha il vantaggio di potersi applicare anche a un DTD che non sia stato predisposto con elementi atti all'inserimento di testo letterale. Purtroppo, non tutti gli strumenti SGML sono in grado di riconoscere le sezioni marcate; si pensi ai navigatori, che pur sapendo interpretare l'HTML, non sono sempre in grado di riconoscere tali particolarità.

51.1.12 Cataloghi

Nelle sezioni precedenti si è visto che il DTD può essere composto da diversi file fisici nel sistema. Lo stesso preambolo di un sorgente SGML prevede la dichiarazione e l'inclusione di un DTD. È stato mostrato come includere un blocco di DTD esterno, attraverso la dichiarazione e il successivo utilizzo di un'entità parametrica che fa riferimento a un file esterno.

Quando si vogliono utilizzare componenti esterni senza fare riferimento a un file preciso, si possono predisporre dei cataloghi, con i quali si esplicitano questi dettagli riferiti al sistema di cui si dispone effettivamente.

Questo tipo di approccio viene usato tipicamente per due motivi: evitare di dover fare riferimento a un file preciso per il DTD nella dichiarazione del tipo di documento all'inizio del sorgente SGML; includere in modo dinamico le entità standard riferite alle lettere accentate e ai simboli speciali. Per quanto riguarda il secondo problema, si deve tenere presente che l'SGML si astrae dalla piattaforma, quindi, il modo in cui le entità di questo tipo vanno rappresentate dipende da quello che si vuole fare dopo.

51.1.12.1 Riferimenti esterni

Generalmente, quando si vogliono usare i cataloghi, si possono fare due tipi di riferimenti a componenti esterne: l'identificatore pubblico e l'identificatore di sistema. Seguono quattro esempi significativi a questo proposito: nei primi due si tratta della dichiarazione del tipo di documento '**HTML**' e di un'entità parametrica, attraverso un identificatore pubblico (una stringa piuttosto lunga); negli ultimi due si tratta delle stesse dichiarazioni, ma fatte attraverso un identificatore di sistema.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
```

```
<!ENTITY % ISOlat1 PUBLIC "ISO 8879:1986//ENTITIES Added Latin 1//EN">
```

```
<!DOCTYPE HTML SYSTEM>
```

```
<!ENTITY % ISOlat1 SYSTEM>
```

Si tratta, evidentemente, di due approcci equivalenti, ma che hanno delle conseguenze differenti nell'applicazione pratica. Dalle parole

chiave utilizzate, '**PUBLIC**' e '**SYSTEM**', si può intuire che l'identificatore di sistema è legato alla situazione del sistema, anche se non è obbligatoria l'indicazione immediata del file corrispondente.

L'uso degli identificatori pubblici è quindi una scelta più conveniente, essendo meno vincolata alla piattaforma. Infatti, questi vengono utilizzati prevalentemente per tutto ciò che è già stato standardizzato: i DTD standard e le entità esterne standard.

51.1.12.2 Il catalogo in pratica

Quando si usano strumenti di analisi ed elaborazione SGML comuni, il catalogo è un file. A seconda degli strumenti utilizzati, potrebbe essere necessario configurare una variabile di ambiente, o usare un'opzione opportuna nella riga di comando, per comunicare a questi la sua posizione.

Il catalogo serve a esplicitare tutte le componenti esterne che non sono state indicate in modo preciso (il nome del file). Si osservi l'esempio seguente:

```
-- Entità standard richiamate attraverso un identificatore di sistema --
-- Sarebbe meglio non usare questo metodo --
ENTITY %ISOlat1            "ISOlat1"
ENTITY %ISOnum             "ISOnum"
ENTITY %ISOdia             "ISOdia"

-- Entità standard richiamate attraverso un identificatore pubblico --
-- Questo tipo di indicazione è preferibile in generale --
PUBLIC "ISO 8879:1986//ENTITIES Added Latin 1//EN"                "ISOlat1"
PUBLIC "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN"  "ISOnum"
PUBLIC "ISO 8879:1986//ENTITIES Diacritical Marks//EN"            "ISOdia"

-- DTD predefinito per il tipo HTML --
DOCTYPE "HTML"                                  "html32.dtd"

-- Identificatori pubblici per le varie forme dell'HTML 3.2 --
PUBLIC "-//W3C//DTD HTML 3.2//EN"               "html32.dtd"
PUBLIC "-//W3C//DTD HTML 3.2 Draft//EN"         "html32.dtd"
PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"         "html32.dtd"
```

Ogni direttiva dell'esempio occupa una riga e si compone di tre parti, dove l'ultima informazione rappresenta il file da utilizzare per quel particolare tipo di entità, documento o identificatore pubblico.

Per la precisione, invece che di file, occorrerebbe parlare di identificatore di sistema effettivo, dove questo concetto viene poi definito dallo standard ISO 8879. In generale si tratta di file e questo dovrebbe bastare come primo approccio all'SGML.

Si noti che i commenti sono delimitati da coppie di trattini, '**--**', come si fa all'interno delle istruzioni SGML.

Direttiva	Descrizione
`PUBLIC` ***identificatore_pubblico*** ↩ ↪***identificatore_di_sistema***	Stabilisce l'identificatore di sistema effettivo (il file) corrispondente all'identificatore pubblico indicato. Quando possibile, è preferibile utilizzare gli identificatori pubblici per definire gli oggetti.
`DOCTYPE` ***nome identificatore_di_sistema***	Stabilisce l'identificatore di sistema effettivo (il file) corrispondente al nome del tipo di documento indicato. Dal momento che questo nome può fare riferimento a uno tra diversi DTD alternativi (si pensi al caso dell'HTML con le sue versioni), questa dichiarazione serve prevalentemente per stabilire un DTD predefinito nel caso in cui non sia stato specificato un identificatore pubblico nel documento che si elabora.

Direttiva	Descrizione
`ENTITY` ***nome identificatore_di_sistema***	Stabilisce l'identificatore di sistema effettivo (il file) corrispondente all'entità generale indicata.
`ENTITY %`***nome identificatore_di_sistema***	Stabilisce l'identificatore di sistema effettivo (il file) corrispondente all'entità parametrica indicata. Si osservi il fatto che il simbolo di percentuale è attaccato al nome dell'entità.

Negli esempi seguenti, viene mostrata prima l'istruzione utilizzata nel DTD, o nel preambolo del sorgente SGML, quindi si presenta la direttiva corrispondente, necessaria nel catalogo.

- `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">`
  Si tratta della dichiarazione, all'inizio di un sorgente SGML, dell'utilizzo del DTD '**`HTML`**', definito in base all'identificatore pubblico '**`-//W3C//DTD HTML 3.2 Final//EN`**'. Perché da questo si possa arrivare a identificare un file particolare, occorre che nel catalogo appaia una direttiva come quella seguente:
  ```
  PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"          "html32.dtd"
  ```
  In tal caso, all'identificatore pubblico '**`-//W3C//DTD HTML 3.2 Fin`** viene abbinato il file '`html32.dtd`'.
- `<!DOCTYPE HTML SYSTEM">`
  Si tratta della dichiarazione, all'inizio di un sorgente SGML, dell'utilizzo del DTD '**`HTML`**', utilizzando un identificatore di sistema che non viene precisato in modo effettivo. Perché da questo si possa arrivare a identificare un file particolare, occorre che nel catalogo appaia una direttiva come quella seguente:
  ```
  DOCTYPE HTML            "html32.dtd"
  ```
  In tal caso, all'identificatore di sistema '**`HTML`**', viene abbinato il file '`html32.dtd`' (l'identificatore di sistema effettivo).
- `<!ENTITY % ISOlat1 PUBLIC "ISO 8879:1986//ENTITIES Added Latin 1//EN">`
  All'interno del DTD, dichiara l'entità parametrica '**`ISOlat1`**' definita secondo l'identificatore pubblico '**`ISO 8879:1986//ENTITIES Added Latin 1//EN`**'. Perché da questo si possa arrivare a identificare un file particolare, occorre che nel catalogo appaia una direttiva simile a una delle due mostrate di seguito:
  ```
  PUBLIC "ISO 8879:1986//ENTITIES Added Latin 1//EN" "ISOlat1.latex"
  ```
  ```
  ENTITY %ISOlat1 "ISOlat1.latex"
  ```
  Nel primo caso, all'identificatore pubblico '**`ISO 8879:1986//ENTITIES Added Latin 1//EN`**', viene abbinato il file '`ISOlat1.latex`'; nel secondo, si specifica direttamente che l'entità parametrica '**`ISOlat1`**' corrisponde al contenuto del file '`ISOlat1.latex`'.
- `<!ENTITY % ISOlat1 SYSTEM>`
  Nel DTD, viene dichiarata l'entità parametrica '**`ISOlat1`**', utilizzando un identificatore di sistema che non viene precisato in modo effettivo. Perché da questo si possa arrivare a identificare un file particolare, occorre necessariamente che nel catalogo appaia una direttiva come quella seguente, dal momento che non è possibile fare riferimento a un identificatore pubblico:
  ```
  ENTITY %ISOlat1              "ISOlat1.latex"
  ```
  In questo modo si definisce l'identificatore di sistema effettivo dell'entità parametrica '**`ISOlat1`**', facendola corrispondere al file '`ISOlat1.latex`' (l'identificatore di sistema effettivo).

51.2 Elaborazione SGML

L'elaborazione SGML si compone fondamentalmente di un programma in grado di verificare la correttezza formale di un sorgente SGML in base al suo DTD. Questo tipo di programma è l'analizzatore SGML (*SGML parser*) e il suo compito si estende frequentemente alla generazione di un risultato intermedio, pronto per una rielaborazione successiva, normalmente attraverso un sistema di composizione tipografica.

L'elaborazione successiva richiede strumenti specifici, ma per le situazioni più semplici, dove basta rimpiazzare un marcatore con una codifica equivalente adatta a un programma di composizione tipografica particolare, si è utilizzato in passato il cosiddetto ASP: *Amsterdam SGML parser*.

L'utilizzo di un analizzatore SGML, precisamente il pacchetto SP con il programma '**`nsgmls`**', è una cosa consueta e attuale, mentre l'utilizzo di un analizzatore ASP può considerarsi una tecnica superata. Tuttavia, l'abbinamento di '**`nsgmls`**' e '**`sgmlsasp`**' (il secondo è un analizzatore ASP) è un metodo semplice e pratico per costruire i propri strumenti SGML, quando non si vuole utilizzare quello che è già a disposizione.

In sostituzione di '**`sgmlsasp`**' si può utilizzare anche il pacchetto SGMLSpm, il quale si compone di una serie di moduli Perl e in particolare fornisce il programma '**`sgmlspl`**', che svolge un compito simile a quello di un analizzatore ASP.

51.2.1 SP

SP è il pacchetto di analisi SGML di James Clark. Si tratta dello strumento fondamentale, ed è disponibile anche su piattaforme differenti dai sistemi Unix. In passato, al posto di SP, era disponibile il pacchetto Sgmls che comunque non è compatibile con molte caratteristiche particolari dell'SGML.

Il pacchetto SP contiene il programma '**`nsgmls`**', assieme a una serie di DTD di esempio. Il programma '**`nsgmls`**' è tutto quello che serve per convalidare un file SGML con il suo DTD e per generare un risultato intermedio analizzabile automaticamente attraverso '**`sgmlsasp`**', un accessorio del vecchio pacchetto Sgmls, o in alternativa attraverso '**`sgmlspl`**', del pacchetto SGMLSpm.

51.2.1.1 Avvio di nsgmls

Il programma '**`nsgmls`**' utilizza lo standard input, oppure i file indicati in coda alla riga di comando (gli identificatori di sistema), per analizzarne il contenuto secondo l'SGML ed eventualmente per generare un output pre-elaborato.

```
nsgmls [opzioni] [identificatore_di_sistema]...
```

Gli errori vengono segnalati attraverso lo standard error, mentre il risultato dell'elaborazione viene emesso attraverso lo standard output.

Opzione	Descrizione
`-c` ***identificatore_di_sistema***	Permette di specificare l'utilizzo di un catalogo, rappresentato dal file indicato come argomento dell'opzione. Questa opzione può essere specificata più volte, per richiedere l'utilizzo di più cataloghi. Se nella stessa directory del file del documento analizzato esiste un file denominato '`catalog`', questo viene aggiunto in coda ai cataloghi letti attraverso questa opzione. Inoltre, se esiste la variabile di ambiente ***SGML_CATALOG_FILES***, l'elenco dei cataloghi in essa contenuti viene aggiunto in coda a tutti gli altri.

Opzione	Descrizione
`-D` ***directory***	Permette di definire una directory da utilizzare per la ricerca di file specificati negli identificatori di sistema. Sono ammissibili più opzioni '**`-D`**'. Se esiste la variabile di ambiente ***SGML_SEARCH_PATH***, l'elenco di directory che questa contiene viene aggiunto in coda a quello definito attraverso l'opzione '**`-D`**'.
`-E` ***n_massimo_errori***	Permette di stabilire il numero massimo di errori, dopo il quale '**`nsgmls`**' termina l'analisi. Il valore predefinito è 200.
`-i`***nome***	Permette di definire un'entità parametrica, con il nome indicato, contenente la stringa '**`INCLUDE`**'. In pratica ciò che nel DTD dovrebbe essere definito con l'istruzione '**`<!ENTITY %` *nome* `"INCLUDE">`**'. Questa dichiarazione prende la precedenza su un'altra dichiarazione della stessa entità fatta in qualunque altra posizione; il suo scopo è quello di facilitare la gestione delle sezioni marcate da includere in modo condizionato. In pratica, si definiscono nel DTD solo entità parametriche di questo tipo con il valore '**`IGNORE`**', con le quali si delimitano parti di testo attraverso l'uso di sezioni marcate. Quindi, quando si vogliono includere quelle porzioni di testo, si può utilizzare questa opzione, anche più volte, per fare sì che le entità parametriche desiderate contengano invece la parola chiave '**`INCLUDE`**'.
`-s`	Sopprime l'emissione dell'output intermedio. In questo modo si limita a emettere le segnalazioni di errori attraverso lo standard error.
`-p`	Analizza solo il prologo, in pratica il DTD, ignorando il documento. Ciò implica, di fatto, l'uso dell'opzione '**`-s`**'.

Segue la descrizione di alcuni esempi.

- `$` **`nsgmls -s -c ~/catalogo`** [*Invio*]

  Si limita a convalidare il contenuto del documento proveniente dallo standard input, avvalendosi del catalogo contenuto del file '`~/catalogo`'.

- `$` **`nsgmls -c ~/catalogo`** [*Invio*]

  Convalida il contenuto del documento proveniente dallo standard input, avvalendosi del catalogo contenuto del file '`~/catalogo`', generando anche il documento rielaborato opportunamente.

- `$` **`nsgmls -i annotazioni -c ~/catalogo`** [*Invio*]

  Come nell'esempio precedente, ma in più dichiara l'entità parametrica '**`annotazioni`**' contenente la parola chiave '**`INCLUDE`**'.

51.2.1.2 Variabili di ambiente

Ci sono due variabili di ambiente a cui è sensibile '**`nsgmls`**': ***SGML_SEARCH_PATH*** e ***SGML_CATALOG_FILES***. Entrambe servono a contenere l'indicazione di un elenco di percorsi, separati attraverso i soliti due punti ('**`:`**').

La variabile ***SGML_SEARCH_PATH*** serve ad aggiungere altre directory a quelle che possono essere definite attraverso l'opzione '**`-D`**', per la ricerca di file corrispondenti agli identificatori di sistema.

La variabile ***SGML_CATALOG_FILES*** serve ad aggiungere altri cataloghi (indicati con il loro percorso assoluto) a quelli che possono essere definiti attraverso l'opzione '**`-c`**'.

Queste due variabili possono essere molto importanti quando si devono fornire queste indicazioni, senza avere il controllo diretto sul comando di avvio dell'eseguibile '**`nsgmls`**'. In pratica, quando si installano strumenti SGML che si avvalgono di SP e c'è la necessità di indicare dove si trova il file del catalogo, oppure dove si trovano gli altri file, la modifica di queste variabili può essere l'unica soluzione.

51.2.1.3 Formato dell'output

Il risultato dell'output dell'elaborazione di un file SGML attraverso '**`nsgmls`**' è composto da una serie di righe di testo, di lunghezza variabile, precedute da un carattere nella prima colonna che ne definisce il significato.

In pratica, ogni riga inizia necessariamente con un codice composto da un solo carattere di «comando», e subito dopo, senza spazi aggiuntivi, inizia il contenuto di uno o più argomenti, a seconda del comando, separati da un solo carattere spazio. L'ultimo argomento (che potrebbe anche essere l'unico) può contenere spazi.

Gli «argomenti» di questi comandi possono contenere delle sequenze di escape:

Sequenza	Descrizione
`\\`	rappresenta una singola barra obliqua inversa ('\');
`\n`	rappresenta la fine di una riga (*record end*), secondo la logica di SGML;
`\\|`	viene usato per delimitare (all'inizio e alla fine) le entità di tipo '**`SDATA`**', dopo che queste sono state espanse regolarmente;
`\`***nnn***	rappresenta un carattere particolare attraverso il suo codice ottale.

'**`nsgmls`**' prevede un numero molto grande di caratteri di comando per distinguere il contenuto delle righe del risultato dell'elaborazione. Qui ne vengono mostrati solo alcuni, i più comuni. Gli altri sono descritti dettagliatamente nella pagina di manuale *nsgmls(1)*.

Comando	Descrizione
`(`***identificatore_generico***	Una parentesi aperta rappresenta l'inizio di un elemento, nominato subito dopo (l'identificatore generico). Se questo elemento dovesse avere attributi, verrebbero rappresentati prima, attraverso i comandi '**`A`**'.
`)`***identificatore_generico***	Una parentesi chiusa rappresenta la fine di un elemento, nominato subito dopo (l'identificatore generico).
`A`***nome_attributo valore***	Specifica un attributo per il prossimo elemento. Se l'elemento possiede più attributi, si utilizzano altrettanti record di tipo '**`A`**'. Il valore assegnato all'attributo si può articolare in più componenti, che qui non vengono descritte.
`C`	Questa lettera, che appare da sola alla fine dell'output di '**`nsgmls`**', rappresenta che il contenuto del file sorgente è corretto.

Di seguito vengono descritti alcuni esempi, rappresentati da pezzi dell'output di '**`nsgmls`**'.

- ```
  (HTML
  (HEAD
  (TITLE
  -Introduzione all'SGML
  )TITLE
  )HEAD
  (BODY
  ...
  )BODY
  )HTML
  ```

  Quello che si vede sopra, rappresenta lo schema fondamentale di ciò che si può ottenere analizzando un file HTML. Si può osservare l'apertura e la chiusura dei vari elementi ('**`HTML`**', '**`HEAD`**', '**`TITLE`**', '**`HEAD`**', '**`BODY`**'). I puntini di sospensione rappresentano solo l'interruzione e la ripresa della visualizzazione dell'output.
  ```

```
(P
-Ciao,\ncome stai?\nIo bene; e tu?
)P
```

Rappresenta un elemento '**P**' contenente una frase, divisa in vari punti dal codice '**\n**', che rappresenta la fine della riga (*record end*) secondo SGML.

```
ANAME IMPLIED
AHREF CDATA indice.html
AREL IMPLIED
AREV IMPLIED
ATITLE IMPLIED
(A
-Indice generale
)A
```

Rappresenta un elemento '**A**', contenente la frase «Indice generale», e una serie di attributi: '**NAME**', '**HREF**', '**REL**', '**REV**' e '**TITLE**'.

```
C
```

Alla fine dell'output, il carattere di comando '**C**' rappresenta il buon fine dell'elaborazione.

51.2.2 Sgmls

Il pacchetto Sgmls è stato il predecessore di SP. Questo forniva il programma '**sgmls**', il cui funzionamento è analogo a '**nsgmls**' anche se meno completo, e '**sgmlsasp**', un analizzatore ASP utile ancora adesso in quanto abbinabile all'output di '**nsgmls**'.

'**sgmlsasp**' elabora lo standard input, in base al contenuto di uno o più file specificati come argomenti. Lo standard input deve essere compatibile con il formato standard di '**sgmls**' e di '**nsgmls**', mentre i file di rimpiazzo devono rispettare il formato ASP (*Amsterdam SGML parser*). Il risultato viene emesso attraverso lo standard output.

```
sgmlsasp file_di_rimpiazzo...
```

'**sgmlsasp**' è in grado di elaborare solo alcuni dei comandi contenuti nei record dell'output di '**nsgmls**', cosa che limita in parte le funzionalità utilizzabili con l'SGML.

Il file di rimpiazzo, secondo lo standard ASP, permette di sostituire i marcatori riferiti alle entità con delle stringhe che si presume siano utili per l'elaborazione successiva del testo. Questo file può contenere dei commenti, preceduti dal simbolo di percentuale e terminati dalla fine della riga del file. Le righe bianche e quelle vuote vengono ignorate.

Le direttive si compongono di due soli elementi: il marcatore di apertura o di chiusura e la stringa da utilizzare per il rimpiazzo. Si osservi l'esempio seguente:

```
<titolo>        +       "\n\\section{"
</titolo>               "}"                     +
```

In questo modo, si dichiara di voler sostituire il marcatore '**<titolo>**' con la stringa '**\n\\section{**', mentre il marcatore '**</titolo>**' va sostituito con la stringa '**}**'. Come può intuire chi conosce LaTeX, si vuole sostituire all'elemento '**titolo**' l'ambiente '**\section{}**' di LaTeX.

La stringa usata per il rimpiazzo può contenere delle sequenze di escape. Per la precisione può trattarsi di:

Sequenza	Descrizione
`\\`	rappresenta una singola barra obliqua inversa ('\');
`\"`	rappresenta un apice doppio con valore letterale;
`\[`	rappresenta una parentesi quadra aperta con valore letterale;
`\]`	rappresenta una parentesi quadra chiusa con valore letterale;
`\n`	rappresenta un'interruzione di riga (*newline*).

Pertanto, la stringa di rimpiazzo vista nell'esempio, va letta come: '*new-line***\section{**'.

All'inizio e alla fine della stringa di rimpiazzo può apparire il segno '**+**'. Se è presente, significa che in quel punto si richiede espressamente l'aggiunta di un'interruzione di riga. Se una stringa di rimpiazzo termina con un '**+**' e subito dopo si deve inserire un'altra stringa di rimpiazzo che è preceduta da un altro '**+**', si ottiene comunque una sola interruzione di riga, perché il secondo '**+**' si limita a confermarla.

Una stringa di rimpiazzo può apparire su più righe, come nell'esempio seguente:

```
<relazione>     +       "\\documentstyle{article}\n"
                        "\\begin{document}"     +

</relazione>    +       "\\end{document}"       +
```

Quando un elemento prevede degli attributi, il contenuto di questi può essere inserito nella stringa di rimpiazzo utilizzando la notazione '**[***nome_attributo***]**', dove le parentesi quadre servono a delimitare questo nome, che in particolare va indicato con caratteri maiuscoli.

```
<etichetta>             "\\label{[ID]}"
</etichetta>
```

L'esempio mostra la sostituzione del marcatore '**<etichetta id=...>**' con la stringa '**\label{...}**', dove i puntini di sospensione rappresentano il valore dell'attributo '**ID**'.

51.2.3 SGMLSpm

SGMLSpm è un pacchetto che si compone di moduli e programmi Perl, per la gestione dell'output generato da '**nsgmls**' (SP). Il modo più semplice per sfruttare le funzionalità di questo pacchetto è quello di utilizzare direttamente il programma '**sgmlspl**', scritto ovviamente in Perl, con cui è sufficiente predisporre un file simile a quello utilizzato per la sostituzione ASP.

Qui viene mostrato soltanto il funzionamento di '**sgmlspl**', ma il lettore tenga presente che il pacchetto SGMLSpm offre molte possibilità in più, se si vuole programmare in Perl allo scopo di elaborare l'SGML.

```
sgmlspl script [opzione_script]... < file_sp > file_elaborato
```

'**sgmlspl**' elabora quanto riceve dallo standard input generando un risultato che emette attraverso lo standard output, utilizzando le specifiche indicate nel file che deve essere indicato come primo e unico argomento, che in pratica è uno script di '**sgmlspl**' stesso.

In questo senso, eventuali argomenti successivi vengono passati direttamente allo script.

In pratica, lo standard input deve corrispondere al risultato emesso dall'analizzatore SP ('**nsgmls**') e il file delle specifiche è un pezzo di programma Perl, scritto sfruttando le caratteristiche di SGMLSpm. È il file delle specifiche che stabilisce il modo in cui i marcatori degli elementi SGML vengono trasformati nel risultato finale.

51.2.3.1 File con le specifiche di sostituzione

Rispetto al meccanismo di rimpiazzo utilizzato da ASP, in questo caso si devono scrivere delle righe di codice Perl abbinate agli eventi che interessano, riferiti all'analisi del file generato da SP. Volendo, oltre a distinguere i marcatori di apertura e di chiusura degli elementi, si possono individuare anche le stringhe SDATA e altri componenti di utilizzo meno frequente. Tenendo conto che il pacchetto SGMLSpm è accompagnato da una buona documentazione, qui viene mostrato semplicemente come gestire la sostituzione dei marcatori che delimitano gli elementi SGML.

Come accennato, il file per la sostituzione (ovvero il file delle specifiche) è scritto in Perl e, in particolare, tutto è visto in forma di reazione al verificarsi di un evento:

```
sgml( evento, funzione_da_eseguire );
```

Quello appena mostrato è lo schema generale delle istruzioni da utilizzare per descrivere ciò che deve fare '**sgmlspl**' quando si verifica l'evento specificato nel primo argomento. In pratica, quando si verifica, viene eseguita la funzione del secondo argomento.

L'evento viene specificato in forma di stringa, dove in particolare la forma '**<*x*>**' rappresenta l'incontro del marcatore di apertura dell'elemento ***x*** e, conseguentemente, '**</*x*>**' rappresenta il marcatore di chiusura. Naturalmente, '**sgmlspl**' è in grado di intercettare molti altri tipi di eventi, che comunque non vengono mostrati qui.

È importante tenere presente che gli eventi che identificano i marcatori di apertura e di chiusura degli elementi SGML, devono essere indicati nella loro forma «normalizzata» secondo l'SGML. In pratica, ciò significa che in generale devono essere annotati utilizzando esclusivamente lettere maiuscole.

La funzione indicata come secondo argomento può essere semplicemente una stringa, intendendo che questa rappresenti ciò che si vuole emettere al posto dell'evento che si è manifestato, oppure una funzione (eventualmente un puntatore a una funzione dichiarata altrove), che probabilmente si occupa di generare un qualche tipo di output.

Generalmente, all'interno delle funzioni da abbinare agli eventi si utilizza la subroutine '**output**' per emettere dell'output, secondo quanto prescritto dalla documentazione di SGMLSpm.

Il passaggio degli attributi contenuti eventualmente nei marcatori di apertura degli elementi SGML, non è così intuitivo come avviene nella sintassi ASP. In questo caso occorre considerare che la funzione indicata come secondo argomento riceve degli argomenti in forma di oggetti, da cui possono essere estratte le informazioni sugli attributi SGML.

Si passa alla dimostrazione di alcuni esempi che dovrebbero essere sufficienti per mostrare l'utilizzo essenziale del file delle specifiche di sostituzione per '**sgmlspl**'.

- ```
  sgml( '<RELAZIONE>', "\n\\documentstyle{article}\n\\begin{document}\n" );
  sgml( '</RELAZIONE>', "\n\\end{document}\n");
  ```

  Questa è la situazione più semplice, in cui ci si limita a sostituire i marcatori con una stringa conveniente (in questo caso si tratta di istruzioni LaTeX). Si osservi il fatto che le istruzioni terminano con il punto e virgola; inoltre si utilizza la sequenza '**\n**' per indicare l'inserimento di un codice di interruzione di riga.

- ```
  sgml( '<RELAZIONE>', sub {
      output "\n\\documentstyle{article}";
      output "\n\\begin{document}\n";
  });
  sgml( '</RELAZIONE>', sub {
      output "\n\\end{document}\n";
  });
  ```

  In questo caso, si vuole ottenere lo stesso risultato dell'esempio precedente, con la differenza che nel secondo argomento si indica effettivamente una funzione (senza nome), il cui scopo è semplicemente quello di emettere le stesse stringhe già viste precedentemente, attraverso la subroutine '**output**'.

- ```
  sub relazione_apertura {
      output "\n\\documentstyle{article}";
      output "\n\\begin{document}\n";
  });

  sgml( '<RELAZIONE>', \&relazione_apertura );
  ```

  Questa rappresenta un'altra variante dell'esempio iniziale, in cui, per il marcatore di apertura, si fa riferimento a una subroutine esterna, indicata attraverso un puntatore alla stessa.

- ```
  sgml( '<ETICHETTA>', sub{
      my ($elemento,$evento) = @_;
      my $id = $elemento->attribute('ID')->value;
      output "\\label{$id}";
  });
  sgml( '</ETICHETTA>', '' );
  ```

  Questo esempio mostra il caso di un elemento SGML che prevede l'attributo '**ID**' nel marcatore di apertura. Per estrarre il valore di questo attributo occorre agire come si vede: si distinguono gli argomenti della funzione dichiarando due variabili private corrispondenti, '**my ($elemento,$evento) = @_;**', quindi si ottiene l'attributo richiesto dall'oggetto a cui fa riferimento la variabile '**$elemento**': '**$elemento->attribute('ID')->value**'. Quello che si ottiene viene conservato nella variabile '**$id**', che poi viene inserita nella stringa emessa attraverso la subroutine '**output**'.

  In questo caso, il marcatore di chiusura dell'elemento viene rimpiazzato semplicemente con una stringa nulla.

- ```
  sgml( '<IMMAGINE>', sub{
      my ($elemento,$evento) = @_;
      my $file = $elemento->attribute('FILE')->value;
      my $altezza = $elemento->attribute('ALTEZZA')->value;
      output "\n\\begin{center}\n";
      output "\\epsfig{file=$file,height=$altezza,angle=0}\n";
      output "\\end{center}\n";
  });
  sgml( '</IMMAGINE>', '' );
  ```

  Quello che si vede è un esempio simile a quello precedente, con la differenza che gli attributi da estrarre sono due.

- ```
  sgml('<LIST>', sub {
    my ($element,$event) = @_;
    my $type = $element->attribute('TYPE')->value;

    if ($type eq 'ORDERED') {
      output "\\begin{enumerate}\n";
    } elsif ($type eq 'UNORDERED') {
      output "\\begin{itemize}\n";
    } else {
      die "Bad TYPE '$type' for element LIST at line " .
        $event->line . " in " . $event->file . "\n";
    }
  });
  ```

  Questo esempio proviene dalla documentazione di SGMLSpm e mostra in che modo modificare il risultato della trasformazione in base al contenuto degli attributi di un elemento SGML.

51.2.3.2 Scheletro pronto con «skel.pl»

Da quanto è stato mostrato, si intende che la realizzazione di uno script con le specifiche di sostituzione per l'uso con '**sgmlspl**' non rappresenta un problema. Tuttavia, assieme alla documentazione di SGMLSpm si trova uno script speciale per '**sgmlspl**' che aiuta nella sua realizzazione iniziale. Si tratta di '**skel.pl**':

```
sgmlspl skel.pl < file_sp > scheletro_script_sgmlspl
```

Come si vede, si deve partire da un risultato generato da SP; da questo si ottiene uno scheletro per la realizzazione del proprio script di '**sgmlspl**'. In generale si tratta di qualcosa simile all'esempio seguente:

```
##############################################################################
# SGMLSPL script produced automatically by the script sgmlspl.pl
#
# Document Type: SGMLTEXI
# Edited by:
##############################################################################

use SGMLS;                      # Use the SGMLS package.
use SGMLS::Output;              # Use stack-based output.

#
# Document Handlers.
#
sgml('start', sub {});
sgml('end', sub {});
```

```
#
# Element Handlers.
#

# Element: SGMLTEXI
sgml('<SGMLTEXI>', "");
sgml('</SGMLTEXI>', "");

# Element: HEAD
sgml('<HEAD>', "");
sgml('</HEAD>', "");

# Element: ADMIN
sgml('<ADMIN>', "");
sgml('</ADMIN>', "");

…

#
# Default handlers (uncomment these if needed).  Right now, these are set
# up to gag on any unrecognised elements, sdata, processing-instructions,
# or entities.
#
# sgml('start_element',sub { die "Unknown element: " . $_[0]->name; });
# sgml('end_element','');
# sgml('cdata',sub { output $_[0]; });
# sgml('sdata',sub { die "Unknown SDATA: " . $_[0]; });
# sgml('re',"\n");
# sgml('pi',sub { die "Unknown processing instruction: " . $_[0]; });
# sgml('entity',sub { die "Unknown external entity: " . $_[0]->name; });
# sgml('start_subdoc',sub { die "Unknown subdoc entity: " . $_[0]->name; });
# sgml('end_subdoc','');
# sgml('conforming','');

1;
```

51.2.3.3 Ridirezione dell'output e altre sofisticazioni

Uno script per '**sgmlspl**' è in realtà uno script Perl. In questo senso si possono dichiarare variabili globali e funzioni aggiuntive. Questo consente di accumulare dei dati e di emetterli solo quando tutte le informazioni necessarie sono state ricevute, in un ordine differente rispetto alla struttura del sorgente SGML.

Per migliorare questa possibilità, è consentita la ridirezione del flusso generato attraverso la funzione ***output()***, in modo da poterlo ripescare al momento del bisogno. Prima di vedere come funziona questa cosa, si pensi a un problema tipico: si vuole accumulare in qualche modo l'informazione contenuta nell'elemento '**titolo**', in modo da poterla emettere nel momento appropriato.

In generale, si riesce a intercettare il marcatore di apertura e quello di chiusura dell'elemento, senza poter «afferrare» il testo contenuto. Più o meno nel modo seguente:

```
sgml('<TITOLO>', sub{
    output "\n\\section{";
});
sgml('</TITOLO>', sub{
    output "}";
});
```

Ma quel titolo potrebbe servire per qualche motivo. Ecco come si risolve il problema:

```
sgml('<TITOLO>', sub{
    output "\n\\section{";
    push_output('string');
});
sgml('</TITOLO>', sub{
    $titolo = pop_output;
    output "$titolo}";
});
```

Attraverso l'istruzione '**push_output('string')**' viene ridiretto temporaneamente tutto il flusso verso una stringa indefinita (viene chiarito meglio tra poco); successivamente viene prelevato il testo accumulato con l'istruzione '**pop_output**', inserendolo nella variabile '**$titolo**'. Infine, il testo accumulato viene anche emesso nuovamente attraverso la funzione ***output()***.

Ecco come si presenta la sintassi di queste due istruzioni:

```
push_output( tipo[, file] )
pop_output
```

In pratica, la funzione ***push_output()*** ridirige il flusso generato dalla funzione ***output()*** (che viene usata anche internamente a '**sgmlspl**'. Questo flusso può essere ridiretto verso oggetti differenti, identificati da una parola chiave che va indicata come primo argomento, come si vede nella tabella 51.121.

Tabella 51.121. Tipi di ridirezione della funzione ***push_output()***.

Tipo	Descrizione
'handle'	Ridirige verso un flusso aperto (*file handle* indicato nel secondo argomento.
'file'	Ridirige verso un file che viene creato per l'occasione.
'append'	Ridirige aggiungendo a un file esistente.
'pipe'	Ridirige inviando allo standard input del comando indicato.
'string'	Ridirige in un'area temporanea.
'nul'	Perde l'output.

La ridirezione verso un flusso di file già aperto, verso un file e verso un condotto è un concetto abbastanza intuitivo. In questi casi il secondo argomento indica il flusso, il file o il comando a cui si ridirige. Per esempio:

```
push_output( 'handle', MIO_FILE );
```

invia l'output verso il file già aperto con il nome '**MIO_FILE**';

```
push_output( 'file', "/tmp/pippo" );
```

genera il file '/tmp/pippo' e vi inserisce l'output;

```
push_output( 'append', "/tmp/pippo" );
```

accoda al file '/tmp/pippo';

```
push_output( 'pipe', "mail tizio" );
```

invia un messaggio di posta elettronica all'utente '**tizio**'.

Al contrario, il comando '**push_output('string')**' non prevede un secondo argomento e invia i dati in un'area indefinita, che può essere recuperata solo attraverso la funzione '**pop_output**'. La funzione '**pop_output**' serve in generale per concludere una ridirezione precedente, mentre quando si tratta in particolare di un flusso di output ridiretto verso questa area indefinita, restituisce quanto accumulato:

```
push_output('string');
…
…
$recupera = pop_output;
…
```

Infine, '**push_output('nul')**' serve a eliminare l'output senza poterlo recuperare.

51.2.3.4 Verifica sintattica secondo Perl

La verifica sintattica di uno script di '**sgmlspl**' può risultare difficile se non si usa un trucchetto: basta aggiungere la definizione di una funzione ***output()*** fittizia, come quella seguente:

```
#sub output {
#    local( $argument ) = $_[0];
#    print "$argument";
#}
```

L'esempio mostra delle righe commentate. Infatti, si tratta di inserire questa funzione fittizia solo nel momento in cui si vuole eseguire un'analisi attraverso Perl, nel modo seguente:

```
$ perl -c pippo.spec [Invio]
```

Qui, si intende che 'pippo.spec' sia lo script da controllare.

51.2.4 Esempio di un mini-sistema SGML

Il modo migliore per comprendere come si possono mettere insieme i vari tasselli di un sistema di composizione che parte dall'SGML, è quello di studiare un esempio elementare, che possa essere esteso facilmente. Si vuole arrivare a generare una trasformazione del sorgente SGML in LaTeX.

Quello che serve è: un DTD, che viene rappresentato dal file '`relazione.dtd`'; un catalogo, rappresentato dal file '`catalogo`'; una serie di file contenenti le entità standard ISO indispensabili e adatte a LaTeX, rappresentate dai file '`ISOlat1.tex`', '`ISOnum.tex`' e '`ISOdia.tex`'; e infine un file di rimpiazzo ASP, rappresentato dal file '`mappa.tex`', oppure un file di specifiche per '**`sgmlspl`**'.

51.2.4.1 Il DTD

Si vuole realizzare un tipo di documento molto semplice, adatto per scrivere delle relazioni banali, composte da un titolo, una data, un corpo più o meno lungo e da una o più firme.

```
<!ENTITY % ISOlat1 PUBLIC "ISO 8879:1986//ENTITIES Added Latin 1//EN">
%ISOlat1;

<!ENTITY % ISOdia PUBLIC "ISO 8879:1986//ENTITIES Diacritical Marks//EN">
%ISOdia;

<!ENTITY % ISOnum PUBLIC
        "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN">
%ISOnum;

<!ENTITY space " ">
<!ENTITY null "">

<!shortref mappaglobale
        "BB" space
        "&#RS;B" null
        "B&#RE;" space
        "&#RS;B&#RE;" null
        "&#RS;&#RE;" null
        "#" num
        "%" percnt
        "@" commat
        "[" lsqb
        "]" rsqb
        "^" circ
        "_" lowbar
        "{" lcub
        "|" verbar
        "}" rcub
        "~" tilde >

<!ELEMENT relazione     - - (titolo?, data, contenuto)>
<!ELEMENT titolo        - o (#PCDATA)>
<!ELEMENT data          - o (#PCDATA)>
<!ELEMENT contenuto     - o (paragrafo+, firma+)>
<!ELEMENT paragrafo     - o (#PCDATA)>
<!ELEMENT firma         - o (#PCDATA)>

<!usemap mappaglobale relazione>
```

Come si può osservare dall'esempio proposto, inizialmente vengono acquisite le entità standard, utilizzando un riferimento pubblico, secondo gli standard. Successivamente vengono definite delle entità aggiuntive e quindi una mappa di sostituzione (*shortref*).

Nella parte finale vengono definiti i vari elementi, a cominciare da quello che ha lo stesso nome del DTD, abbinando l'elemento più esterno all'unica mappa di sostituzione che sia stata definita.

51.2.4.2 Il catalogo

Il catalogo serve a individuare i file corrispondenti alle entità standard e al DTD stesso. Si tratta di poche righe (si osservi il fatto che non è stato definito un identificatore pubblico per il DTD, dal momento che si tratta di un lavoro poco importante).

```
PUBLIC "ISO 8879:1986//ENTITIES Added Latin 1//EN" "ISOlat1.tex"

PUBLIC "ISO 8879:1986//ENTITIES Diacritical Marks//EN" "ISOdia.tex"

PUBLIC "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN" "ISOnum.tex"

DOCTYPE "relazione"             "relazione.dtd"
```


51.2.4.3 Le entità standard

Le entità standard, come tali, si possono recuperare già pronte un po' dappertutto. Eventualmente si può porre il problema di dover modificare le stringhe corrispondenti per il tipo di elaborazione che si intende fare. Di seguito vengono mostrati integralmente i file delle entità utilizzati in questo esempio. È il caso di ricordare che le stringhe di sostituzione sono pensate per LaTeX.

```
<!-- Questa versione del file ISOlat1 è ridotta rispetto
     all'originale dello standard ISO 8879.
     Per la precisione, sono state tolte le entità che esistono
     già negli altri file mostrati.
-->
<!-- Character entity set. Typical invocation:
     <!ENTITY % ISOlat1 PUBLIC
       "ISO 8879:1986//ENTITIES Added Latin 1//EN">
     %ISOlat1;
-->
<!ENTITY aacute CDATA "\'a"--=small a, acute accent-->
<!ENTITY Aacute CDATA "\'A"--=capital A, acute accent-->
<!ENTITY acirc  CDATA "\^a"--=small a, circumflex accent-->
<!ENTITY Acirc  CDATA "\^A"--=capital A, circumflex accent-->
<!ENTITY agrave CDATA "\`a"--=small a, grave accent-->
<!ENTITY Agrave CDATA "\`A"--=capital A, grave accent-->
<!ENTITY aring  CDATA "\aa{}"--=small a, ring-->
<!ENTITY Aring  CDATA "\AA{}"--=capital A, ring-->
<!ENTITY atilde CDATA "\~a"--=small a, tilde-->
<!ENTITY Atilde CDATA "\~A"--=capital A, tilde-->
<!ENTITY auml   CDATA '\"a'--=small a, dieresis or umlaut mark-->
<!ENTITY Auml   CDATA '\"A'--=capital A, dieresis or umlaut mark-->
<!ENTITY aelig  CDATA "\ae{}"--=small ae diphthong (ligature)-->
<!ENTITY AElig  CDATA "\AE{}"--=capital AE diphthong (ligature)-->
<!ENTITY ccedil CDATA "\c c"--=small c, cedilla-->
<!ENTITY Ccedil CDATA "\c C"--=capital C, cedilla-->
<!ENTITY eth    CDATA "\dh{}"--=small eth, Icelandic-->
<!ENTITY ETH    CDATA "\DH{}"--=capital Eth, Icelandic-->
<!ENTITY eacute CDATA "\'e"--=small e, acute accent-->
<!ENTITY Eacute CDATA "\'E"--=capital E, acute accent-->
<!ENTITY ecirc  CDATA "\^e"--=small e, circumflex accent-->
<!ENTITY Ecirc  CDATA "\^E"--=capital E, circumflex accent-->
<!ENTITY egrave CDATA "\`e"--=small e, grave accent-->
<!ENTITY Egrave CDATA "\`E"--=capital E, grave accent-->
<!ENTITY euml   CDATA '\"e'--=small e, dieresis or umlaut mark-->
<!ENTITY Euml   CDATA '\"E'--=capital E, dieresis or umlaut mark-->
<!ENTITY iacute CDATA "\'\i{}"--=small i, acute accent-->
<!ENTITY Iacute CDATA "\'I"--=capital I, acute accent-->
<!ENTITY icirc  CDATA "\^\i{}"--=small i, circumflex accent-->
<!ENTITY Icirc  CDATA "\^I"--=capital I, circumflex accent-->
<!ENTITY igrave CDATA "\`\i{}"--=small i, grave accent-->
<!ENTITY Igrave CDATA "\`I"--=capital I, grave accent-->
<!ENTITY iuml   CDATA '\"\i{}'--=small i, dieresis or umlaut mark-->
<!ENTITY Iuml   CDATA '\"I'--=capital I, dieresis or umlaut mark-->
<!ENTITY ntilde CDATA "\~n"--=small n, tilde-->
<!ENTITY Ntilde CDATA "\~N"--=capital N, tilde-->
<!ENTITY oacute CDATA "\'o"--=small o, acute accent-->
<!ENTITY Oacute CDATA "\'O"--=capital O, acute accent-->
<!ENTITY ocirc  CDATA "\^o"--=small o, circumflex accent-->
<!ENTITY Ocirc  CDATA "\^O"--=capital O, circumflex accent-->
<!ENTITY ograve CDATA "\`o"--=small o, grave accent-->
<!ENTITY Ograve CDATA "\`O"--=capital O, grave accent-->
<!ENTITY oslash CDATA "\o{}"--=small o, slash-->
<!ENTITY Oslash CDATA "\O{}"--=capital O, slash-->
<!ENTITY otilde CDATA "\~o"--=small o, tilde-->
<!ENTITY Otilde CDATA "\~O"--=capital O, tilde-->
<!ENTITY ouml   CDATA '\"o'--=small o, dieresis or umlaut mark-->
<!ENTITY Ouml   CDATA '\"O'--=capital O, dieresis or umlaut mark-->
<!ENTITY szlig  CDATA "\ss{}"--=small sharp s, German (sz ligature)-->
<!ENTITY thorn  CDATA "\th{}"--=small thorn, Icelandic-->
<!ENTITY THORN  CDATA "\TH{}"--=capital THORN, Icelandic-->
<!ENTITY uacute CDATA "\'u"--=small u, acute accent-->
<!ENTITY Uacute CDATA "\'U"--=capital U, acute accent-->
<!ENTITY ucirc  CDATA "\^u"--=small u, circumflex accent-->
<!ENTITY Ucirc  CDATA "\^U"--=capital U, circumflex accent-->
<!ENTITY ugrave CDATA "\`u"--=small u, grave accent-->
<!ENTITY Ugrave CDATA "\`U"--=capital U, grave accent-->
<!ENTITY uuml   CDATA '\"u'--=small u, dieresis or umlaut mark-->
<!ENTITY Uuml   CDATA '\"U'--=capital U, dieresis or umlaut mark-->
<!ENTITY yacute CDATA "\'y"--=small y, acute accent-->
<!ENTITY Yacute CDATA "\'Y"--=capital Y, acute accent-->
<!ENTITY yuml   CDATA '\"y'--=small y, dieresis or umlaut mark-->
```

```
<!-- (C) International Organization for Standardization 1986
     Permission to copy in any form is granted for use with
     conforming SGML systems and applications as defined in
     ISO 8879, provided this notice is included in all copies.
-->
<!-- Character entity set. Typical invocation:
     <!ENTITY % ISOnum PUBLIC
       "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN">
     %ISOnum;
-->
```

```
<!ENTITY half   CDATA "$\scriptstyle{1\over2}$"--=fraction one-half-->
<!ENTITY frac12 CDATA "\sfrac1/2"--=fraction one-half-->
<!ENTITY frac14 CDATA "\sfrac1/4"--=fraction one-quarter-->
<!ENTITY frac34 CDATA "\sfrac3/4"--=fraction three-quarters-->
<!ENTITY frac18 CDATA "\sfrac1/8"--=fraction one-eighth-->
<!ENTITY frac38 CDATA "\sfrac3/8"--=fraction three-eighths-->
<!ENTITY frac58 CDATA "\sfrac5/8"--=fraction five-eighths-->
<!ENTITY frac78 CDATA "\sfrac7/8"--=fraction seven-eighths-->

<!ENTITY sup1   CDATA "$^1$"--=superscript one-->
<!ENTITY sup2   CDATA "$^2$"--=superscript two-->
<!ENTITY sup3   CDATA "$^3$"--=superscript three-->

<!ENTITY plus   CDATA "$+$"--=plus sign B:-- >
<!ENTITY plusmn CDATA "$\pm$"--/pm B: =plus-or-minus sign-->
<!ENTITY lt     CDATA "$<$"--=less-than sign R:-->
<!ENTITY equals CDATA "$=$"--=equals sign R:-->
<!ENTITY gt     CDATA "$>$"--=greater-than sign R:-->
<!ENTITY divide CDATA "$\div$"--/div B: =divide sign-->
<!ENTITY times  CDATA "$\times$"--/times B: =multiply sign-->

<!ENTITY curren CDATA "\{curren\}"--=general currency sign-->
<!ENTITY pound  CDATA "\pounds{}"--=pound sign-->
<!ENTITY dollar CDATA "\$"--=dollar sign-->
<!ENTITY cent   CDATA "\cent{}"--=cent sign-->
<!ENTITY yen    CDATA "\{yen\}"--/yen =yen sign-->

<!ENTITY num    CDATA "\#"--=number sign-->
<!ENTITY percnt CDATA "\%"--=percent sign-->
<!ENTITY amp    CDATA "\&"--=ampersand-->
<!ENTITY ast    CDATA "*"--/ast B: =asterisk-->
<!ENTITY commat CDATA "@"--=commercial at-->
<!ENTITY lsqb   CDATA "["--/lbrack O: =left square bracket-->
<!ENTITY bsol   CDATA "$\backslash$"--/backslash =reverse solidus-->
<!ENTITY rsqb   CDATA "]"--/rbrack C: =right square bracket-->
<!ENTITY lcub   CDATA "$\{$"--/lbrace O: =left curly bracket-->
<!ENTITY horbar CDATA "{--}"--=horizontal bar-->
<!ENTITY verbar CDATA "$|$"--/vert =vertical bar-->
<!ENTITY rcub   CDATA "$\}$"--/rbrace C: =right curly bracket-->
<!ENTITY micro  CDATA "$\mu$"--=micro sign-->
<!ENTITY ohm    CDATA "$\Omega$"--=ohm sign-->
<!ENTITY deg    CDATA "$^\circ$"--=degree sign-->
<!ENTITY ordm   CDATA "\{ordm\}"--=ordinal indicator, masculine-->
<!ENTITY ordf   CDATA "\{ordf\}"--=ordinal indicator, feminine-->
<!ENTITY sect   CDATA "\S{}"--=section sign-->
<!ENTITY para   CDATA "\P{}"--=pilcrow (paragraph sign)-->
<!ENTITY middot CDATA "$\cdot$"--/centerdot B: =middle dot-->
<!ENTITY larr   CDATA "$\leftarrow$"--/leftarrow /gets A: =leftward arrow-->
<!ENTITY rarr   CDATA "$\rightarrow$"--/rightarrow /to A: =rightward arrow-->
<!ENTITY uarr   CDATA "$\uparrow$"--/uparrow A: =upward arrow-->
<!ENTITY darr   CDATA "$\downarrow$"--/downarrow A: =downward arrow-->
<!ENTITY copy   CDATA "\copyright{}"--=copyright sign-->
<!ENTITY reg    CDATA "\rcircle{}"--/circledR =registered sign-->
<!ENTITY trade  CDATA "(TM)"--=trade mark sign-->
<!ENTITY brvbar CDATA "\{brvbar\}"--=broken (vertical) bar-->
<!ENTITY not    CDATA "$\neg$"--/neg /lnot =not sign-->
<!ENTITY sung   CDATA "\{sung\}"--=music note (sung text sign)-->

<!ENTITY excl   CDATA "!"--=exclamation mark-->
<!ENTITY iexcl  CDATA "{!'}"--=inverted exclamation mark-->
<!ENTITY quot   CDATA '{\tt\char'\"}'--=quotation mark-->
<!ENTITY apos   CDATA "'"--=apostrophe-->
<!ENTITY lpar   CDATA "("--O: =left parenthesis-->
<!ENTITY rpar   CDATA ")"--C: =right parenthesis-->
<!ENTITY comma  CDATA ","--P: =comma-->
<!ENTITY lowbar CDATA "\_"--=low line-->
<!ENTITY hyphen CDATA "-"--=hyphen-->
<!ENTITY period CDATA "."--=full stop, period-->
<!ENTITY sol    CDATA "/"--=solidus-->
<!ENTITY colon  CDATA ":"--/colon P:-->
<!ENTITY semi   CDATA ";"--=semicolon P:-->
<!ENTITY quest  CDATA "?"--=question mark-->
<!ENTITY iquest CDATA "{?'}"--=inverted question mark-->
<!ENTITY laquo  CDATA "\guillemotleft{}"--=angle quotation mark, left-->
<!ENTITY raquo  CDATA "\guillemotright{}"--=angle quotation mark, right-->
<!ENTITY lsquo  CDATA "{'}"--=single quotation mark, left-->
<!ENTITY rsquo  CDATA "{'}"--=single quotation mark, right-->
<!ENTITY ldquo  CDATA "{''}"--=double quotation mark, left-->
<!ENTITY rdquo  CDATA "{''}"--=double quotation mark, right-->
<!ENTITY nbsp   CDATA "~"--=no break (required) space-->
<!ENTITY shy    CDATA "\-"--=soft hyphen-->
```



```
<!-- (C) International Organization for Standardization 1986
     Permission to copy in any form is granted for use with
     conforming SGML systems and applications as defined in
     ISO 8879, provided this notice is included in all copies.
-->
<!-- Character entity set. Typical invocation:
     <!ENTITY % ISOdia PUBLIC
       "ISO 8879:1986//ENTITIES Diacritical Marks//EN">
     %ISOdia;
-->
<!ENTITY acute  CDATA "\'"--=acute accent-->
<!ENTITY breve  CDATA "\u{"--=breve-->
<!ENTITY caron  CDATA "\{caron\}"--=caron-->
<!ENTITY cedil  CDATA "\c{"--=cedilla-->
<!ENTITY circ   CDATA "\^{}"--=circumflex accent-->
<!ENTITY dblac  CDATA "\{dblac\}"--=double acute accent-->
<!ENTITY die    CDATA '\"'--=dieresis-->
<!ENTITY dot    CDATA "\."--=dot above-->
<!ENTITY grave  CDATA "\`"--=grave accent-->
<!ENTITY macr   CDATA "\="--=macron-->
<!ENTITY ogon   CDATA "\{ogon\}"--=ogonek-->
<!ENTITY ring   CDATA "\accent23"--=ring-->
<!ENTITY tilde  CDATA "\~{}"--=tilde-->
<!ENTITY uml    CDATA '\"'--=umlaut mark-->
```

51.2.4.4 Rimpiazzo ASP

L'ultimo componente necessario è il file di rimpiazzo ASP per '**sgmlsasp**', oppure il file delle specifiche per '**sgmlspl**'. Vengono mostrati entrambi.

```
%
% mappa.tex
%
<relazione>     +       "\\documentclass{article}\n"
                        "\\begin{document}"             +

</relazione>    +       "\\end{document}"       +

<titolo>        +       "\n\\section{"
</titolo>               "}"             +

<data>          +       "\n"
</data>                 ""      +

<contenuto>
</contenuto>

<paragrafo>     +       "\n"
</paragrafo>            ""      +

<firma>         +       "\n"
</firma>                ""      +
```

```
#
# latex.spec
#
sgml( '<RELAZIONE>', sub {
    output "\n\\documentclass{article}\n";
    output "\\begin{document}";
});
sgml( '</RELAZIONE>', "\n\\end{document}\n" );

sgml( '<TITOLO>', "\n\n\\section{" );
sgml( '</TITOLO>', "}\n" );

sgml( '<DATA>', "\n\n" );
sgml( '</DATA>', "" );

sgml( '<CONTENUTO>', "" );
sgml( '</CONTENUTO>', "" );

sgml( '<PARAGRAFO>', "\n\n" );
sgml( '</PARAGRAFO>', "" );

sgml( '<FIRMA>', "\n\n" );
sgml( '</FIRMA>', "" );
```

51.2.4.5 I documenti SGML

Quello che segue è un esempio di documento SGML adatto al DTD e al catalogo che è stato definito sopra.

```
<!doctype relazione SYSTEM>
```

```
<relazione>
<titolo>Relazione introduttiva su SGML</titolo>

<data>31/12/1999</data>

<contenuto>

<paragrafo>SGML sta per Standard Generalized Markup
Language. bla bla bla... Perch&eacute;,... cos&igrave;...

<paragrafo>Bla, bla, bla....

<firma>Pinco Pallino</firma>

</contenuto>
</relazione>
```

51.2.4.6 I comandi necessari

Una volta scritto un testo SGML, la prima cosa da fare è la verifica di coerenza in base al DTD. Se il file da controllare fosse '`relazione.sgml`', si dovrebbe utilizzare il comando seguente (si presume che il file del catalogo sia collocato nella directory corrente).

```
$ nsgmls -s -c ./catalogo < relazione.sgml [Invio]
```

Una volta corretti gli errori, si può passare direttamente alla trasformazione in LaTeX, ma se lo si desidera, si può osservare l'output generato da '**`nsgmls`**'.

```
$ nsgmls -c ./catalogo < relazione.sgml [Invio]
```

Se si tratta dell'esempio di documento mostrato in precedenza, il risultato dovrebbe essere il seguente (una riga molto lunga appare spezzata per motivi tipografici).

```
(RELAZIONE
(TITOLO
-Relazione introduttiva su SGML
)TITOLO
(DATA
-31/12/1999
)DATA
(CONTENUTO
(PARAGRAFO
SGML sta per Standard Generalized Markup Language.\nbla ↩
↪bla bla... Perch\\'e,... cos\\'\\i{}...
)PARAGRAFO
(PARAGRAFO
-Bla, bla, bla....
)PARAGRAFO
(FIRMA
-Pinco Pallino
)FIRMA
)CONTENUTO
)RELAZIONE
C
```

Vale la pena di concentrare l'attenzione proprio sulla riga più lunga, nella quale si può osservare l'effetto delle sostituzioni delle entità standard, secondo le esigenze di LaTeX.:

```
(PARAGRAFO
SGML sta per Standard Generalized Markup Language.\nbla ↩
↪bla bla... Perch\\'e,... cos\\'\\i{}...
)PARAGRAFO
```

Il comando completo per ottenere una trasformazione in LaTeX, secondo il contenuto del file di rimpiazzo ('`mappa.tex`'), è il seguente:

```
$ cat relazione.sgml | nsgmls -c ./catalogo ↩
↪| sgmlsasp mappa.tex [Invio]
```

Ovvero, nel caso si utilizzi '**`sgmlspl`**':

```
$ cat relazione.sgml | nsgmls -c ./catalogo ↩
↪| sgmlspl latex.spec [Invio]
```

Il risultato ottenuto attraverso lo standard output, che andrebbe ridiretto opportunamente, potrebbe apparire come quello che segue.


```
\documentclass{article}
\begin{document}

\section{Relazione introduttiva su SGML}

31/12/1999

SGML sta per Standard Generalized Markup Language.
bla bla bla... Perch\'e,... cos\'\i{}...

Bla, bla, bla....

Pinco Pallino
\end{document}
```

51.2.5 Lo scalino successivo

Una volta capito come si possono utilizzare gli strumenti SGML comuni, si pongono subito due tipi di problemi: la gestione simultanea di più sistemi di composizione e l'astrazione dal problema della rappresentazione dei simboli che in uno qualunque dei sistemi di composizione richiederebbero codici speciali. Vengono analizzati questi due problemi separatamente.

51.2.5.1 Gestione simultanea di più sistemi di composizione

Quando si organizza un DTD allo scopo di costruire un sistema SGML per la composizione finale in più formati (PostScript, HTML ed eventualmente altro ancora), occorre definire quali siano gli obiettivi, stabilendo così anche i limiti che si devono imporre nel DTD (se si pretende di generare anche un risultato in forma di file di testo puro e semplice, le immagini potrebbero essere inserite nel documento solo in forma di «arte ASCII»).

Dopo il progetto del DTD e del modo in cui devono essere trasformati i vari elementi nelle diverse forme di composizione, si pone un ostacolo un po' fastidioso: le entità generali. Dal momento che queste dovrebbero essere definite in modo differente a seconda del tipo di composizione che si vuole ottenere, si rischia di dover gestire altrettanti cataloghi, dovendo fare riferimento a file differenti.

In un sistema SGML ben ordinato, ci dovrebbe essere un solo catalogo e il problema della distinzione delle entità generali si può ottenere attraverso l'uso delle sezioni marcate. Infatti, dal momento che i file delle entità esterne sono parte del DTD, si possono indicare anche altre istruzioni SGML oltre a quelle di definizione delle entità generali. Quello che segue è un estratto semplificato e abbreviato dal file delle entità esterne utilizzato da ALtools (un vecchio sistema di composizione di *Appunti Linux*).

```
<![CDATA[
<!ENTITY % EntitaASCII8 "IGNORE">
<!ENTITY % EntitaLaTeX "IGNORE">
<!ENTITY % EntitaHTML "IGNORE">

<![ %EntitaASCII8 [
    <!ENTITY excl   CDATA "!"-- exclamation mark -->
    <!ENTITY quot   CDATA '"'-- quotation mark -->
    <!ENTITY num    CDATA "#"-- number sign -->
    …
    …
]]>

<![ %EntitaLaTeX [
    <!ENTITY excl   CDATA "!"-- exclamation mark -->
    <!ENTITY quot   CDATA '{\tt\char'\"}'-- quotation mark -->
    <!ENTITY num    CDATA "\#"-- number sign -->
    …
    …
]]>

<![ %EntitaHTML [
    <!ENTITY excl   CDATA "!"-- exclamation mark -->
    <!ENTITY quot   CDATA '"'-- quotation mark -->
    <!ENTITY num    CDATA "#"-- number sign -->
    …
    …
]]>
```

Nella parte iniziale vengono dichiarate le entità parametriche ‘**EntitaASCII8**’, ‘**EntitaLaTeX**’ e ‘**EntitaHTML**’, tutte con la stringa ‘**IGNORE**’. In questo modo, in condizioni normali, nessuna delle istruzioni di definizioni delle entità generali verrebbe presa in considerazione. Per selezionare un gruppo soltanto, basterebbe che l’entità parametrica giusta contenesse la stringa ‘**INCLUDE**’. Per farlo si interviene direttamente nella riga di comando di ‘**nsgmls**’ (SP):

```
cat file_sgml | nsgmls -c catalogo -ientità_parametrica | ...
```

In pratica, con l’opzione ‘**-i**’ di ‘**nsgmls**’, si fa in modo di introdurre una dichiarazione del tipo

```
<!ENTITY % entità_parametrica "INCLUDE">
```

e questa prevale automaticamente su qualunque altra dichiarazione analoga (della stessa entità parametrica) in qualunque altra parte del DTD.

Per tornare all’esempio mostrato del file delle entità generali, si potrebbero selezionare le entità riferite alla trasformazione in LaTeX con un comando simile a quello seguente:

```
$ cat mio_file.sgml ↩
↪| nsgmls -c ./catalogo -iEntitaLaTeX | ... [Invio]
```

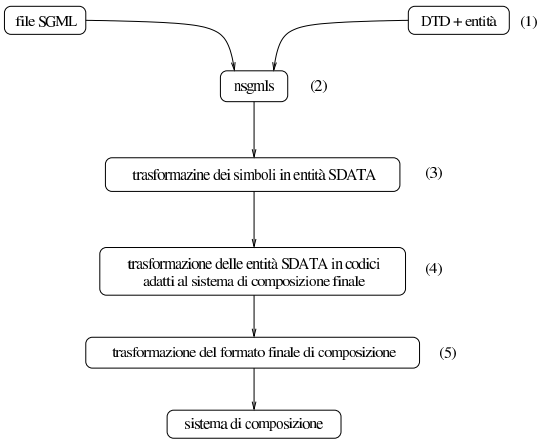
51.2.5.2 Insieme di caratteri

Attraverso le entità generali che si definiscono, è possibile fare in modo che il sistema di composizione finale riceva i codici adatti per tutti i simboli «strani» che si vogliono poter inserire. Tuttavia, spesso si vorrebbe poter scrivere liberamente utilizzando il minor numero possibile di macro ‘**&…;**’. Per la precisione, il minimo in assoluto è quello che richiede l’SGML stesso: occorre proteggere i simboli ‘**&**’, ‘**>**’ e ‘**<**’ (‘**&**’, ‘**>**’, ‘**<**’). Tutto il resto, non dà alcun fastidio all’analizzatore SGML, però i programmi di composizione potrebbero avere dei problemi differenti.

Anche senza uscire dai 7 bit dell’ASCII tradizionali, se si scrive qualcosa per LaTeX, non si possono usare direttamente caratteri normalissimi come ‘**#**’, ‘****’, ‘**$**’ e altri.

Per risolvere questo problema una volta per tutte, si utilizza una tecnica che impone una rielaborazione intermedia del risultato generato da SP dall’analisi del sorgente SGML. Questa tecnica si basa sull’uso di entità generali di tipo ‘**SDATA**’. Quando queste vengono sostituite dallo stesso analizzatore SGML, appaiono delimitate dalla sequenza ‘**\|**’, cosa che ne facilita l’individuazione da parte di un programma di rielaborazione.

Figura 51.140. Passaggi per risolvere il problema dell’insieme dei caratteri.



In questo modo si perde il vantaggio di lasciare fare all’SGML la sostituzione delle entità, però ci si può limitare a intervenire solo dove serve.

Quando si decide di intraprendere questo tipo di approccio, occorre ricordare che l’elaborazione dell’output di ‘**nsgmls**’ deve evitare di intervenire negli elementi «letterali», ovvero quelli che anche nel sistema di composizione finale vengono presi e riprodotti tali e quali.

La descrizione seguente fa riferimento alla figura 51.140.

1. Le entità riferite ai simboli che possono creare problemi vengono definite in una qualche forma simbolica specificando il tipo ‘**SDATA**’. Per esempio, il carattere ‘**#**’ potrebbe essere definito nel modo standard:

   ```
   <!ENTITY num    SDATA "[num   ]"-- number sign -->
   ```

2. ‘**sgmls**’ elabora il file SGML e sostituisce le entità. Quando incontra per esempio la macro ‘**#**’, la trasforma in ‘**\|[num]\|**’.
3. Un programma di elaborazione successivo, quando incontra per esempio il carattere ‘**#**’, lo trasforma in quello che sarebbe stato generato se fosse stata usata la macro ‘**#**’; in pratica lo trasforma in ‘**\|[num]\|**’.
4. A questo punto, i simboli come ‘**#**’ che potrebbero provocare problemi sono stati trasformati tutti nella forma ‘**\|[num]\|**’. Quindi, un programma si deve occupare di trasformarli nel modo adatto al sistema di composizione a cui si devono poi dare in pasto i dati. Nel caso di LaTeX, la stringa ‘**\|[num]\|**’ viene sostituita con ‘**\\#**’. Nel risultato finale, LaTeX richiede solo la stringa ‘**\#**’, ma fino a che si resta nell’ambito del risultato generato da ‘**nsgmls**’, le barre oblique inverse devono essere raddoppiate.
5. Attraverso ‘**sgmlsasp**’, oppure ‘**sgmlspl**’, si genera il risultato finale da passare al sistema di composizione.

51.2.6 Organizzazione degli strumenti SGML in una distribuzione GNU/Linux

Non sempre le distribuzioni GNU/Linux si occupano di organizzare gli strumenti SGML, mentre questo sarebbe molto importante per tutti gli sviluppatori di programmi riferiti a questo standard e a quelli derivati. A questo proposito, vale la pena di osservare la distribuzione Debian che mette in pratica alcune buone idee.

Il problema fondamentale sta nello stabilire la collocazione dei DTD e dei file delle entità generali relative. Infine, si tratta di definire un catalogo unico per tutti questi DTD e per i file delle entità. I file dei DTD vengono collocati nella directory ‘`/usr/share/sgml/dtd/`’, mentre quelli delle entità si trovano nella directory ‘`/usr/share/sgml/entities/`’. A questo punto, per facilitare l’indicazione dei file nel catalogo, questo dovrebbe trovarsi opportunamente nella directory ‘`/etc/sgml/`’, con il nome ‘`catalog`’; così il file del catalogo può essere aggiornato senza interferire con la gerarchia ‘`/usr/`’ che deve poter essere innestata in sola lettura.

Avendo organizzato tutto in questo modo, ogni volta che si installa un nuovo pacchetto di strumenti SGML, questo dovrebbe provvedere ad aggiungere nel catalogo standard tutte le dichiarazioni che lo riguardano.

La base di questa struttura nella distribuzione Debian è costituita dai pacchetti ‘`sgml-base_*.deb`’ e ‘`sgml-data_*.deb`’.

51.2.7 perlSGML: analisi di un DTD

Quando si realizza un DTD per qualche scopo, potrebbe essere importante disporre di strumenti adatti alla sua analisi, per verificare la sua coerenza con l’obiettivo che ci si pone. Sono importanti a questo proposito i programmi di servizio del pacchetto perlSGML. Qui ne vengono mostrati solo alcuni.

In generale, per fare in modo che questi programmi di analisi funzionino correttamente, è opportuno che la directory corrente nel momento in cui si avviano corrisponda a quella in cui si trova il catalogo, in maniera tale che poi da lì, possa trovare le entità che fossero state collocate eventualmente in un file esterno. Se poi il file del catalogo non si chiama '`catalog`', occorre usare l'opzione opportuna per indicare il nome corretto.

Il primo programma che qui viene preso in considerazione a proposito di perlSGML è '**`dtd2html`**':

```
dtd2html [opzioni] file_dtd...
```

'**`dtd2html`**' genera un rapporto sui DTD elencati alla fine degli argomenti, in forma di ipertesto HTML.

Tabella 51.142. Alcune opzioni.

Opzione	Descrizione
`-help`	Emette un riepilogo dell'utilizzo del programma.
`-catalog` *catalogo*	Permette di indicare il nome del file contenente il catalogo SGML. In mancanza di questa opzione, viene cercato il file '`catalog`' nella directory corrente.
`-outdir` *directory*	Permette di specificare una directory diversa da quella corrente, nella quale vanno generate le pagine HTML.
`-ents`	Fa in modo che venga aggiunta una pagina HTML con l'elenco delle entità dichiarate nel corpo principale del DTD.
`-tree`	Fa in modo che venga aggiunta una pagina HTML con l'albero degli elementi SGML collegati tra loro in base alle dipendenze relative.

Segue la descrizione di alcuni esempi.

- `$` **`dtd2html dtd/mio.dtd`** [*Invio*]

  Analizza il file '`./dtd/mio.dtd`' utilizzando il catalogo '`./catalog`' e generando i file HTML nella directory corrente.

- `$` **`dtd2html -catalog catalogo dtd/mio.dtd`** [*Invio*]

  Come nell'esempio precedente, specificando che il catalogo è contenuto nel file '`./catalogo`'.

- `$` **`dtd2html -catalog catalogo -outdir /tmp dtd/mio.dtd`** [*Invio*]

  Come nell'esempio precedente, richiedendo che i file HTML siano creati nella directory '`/tmp/`'.

- `$` **`dtd2html -catalog catalogo -outdir /tmp -ents dtd/mio.dtd`** [*Invio*]

  Come nell'esempio precedente, richiedendo anche la generazione di una pagina dedicata alle entità dichiarate nel DTD.

- `$` **`dtd2html -catalog catalogo -outdir /tmp -ents -tree dtd/mio.dtd`** [*Invio*]

  Come nell'esempio precedente, richiedendo anche la generazione di una pagina contenente l'albero degli elementi.

Il programma '**`dtddiff`**' permette di confrontare due DTD, per conoscere le differenze di contenuto tra i due. Il risultato viene emesso attraverso lo standard output:

```
dtddiff [opzioni] file_dtd file_dtd
```

Tabella 51.143. Alcune opzioni.

Opzione	Descrizione
`-help`	Emette un riepilogo dell'utilizzo del programma.
`-catalog` *catalogo*	Permette di indicare il nome del file contenente il catalogo SGML. In mancanza di questa opzione, viene cercato il file '`catalog`' nella directory corrente.

Segue la descrizione di alcuni esempi.

- `$` **`dtddiff -catalog catalogo dtd/mio.dtd dtd2/mio.dtd`** [*Invio*]

  Confronta i DTD '`./dtd/mio.dtd`' e '`./dtd/mio2.dtd`', utilizzando il catalogo '`./catalogo`'.

51.3 Dichiarazione SGML

Fino a questo punto è stata ignorata la dichiarazione SGML, che in generale non dovrebbe essere un problema per l'utilizzatore, ma rappresenta pur sempre un elemento determinante per la comprensione della filosofia di questo linguaggio.

La dichiarazione SGML è qualcosa che viene prima del DTD; serve a definire la forma del sorgente e alcune caratteristiche del linguaggio utilizzato. Attraverso la dichiarazione si possono modificare molti comportamenti convenzionali, facendo anche cambiare aspetto notevolmente al linguaggio stesso. Tutto quello che è stato descritto di SGML fino a questo punto, fa affidamento sulla dichiarazione SGML raccomandata, ma volendo si potrebbero cambiare molte cose. Per fare un esempio pratico, XML può essere inteso come un modo di utilizzare SGML in base a una dichiarazione particolare, realizzata per le esigenze specifiche della pubblicazione di documentazione attraverso la rete.

La dichiarazione SGML si fa generalmente in un file apposito; tutte le direttive sono contenute all'interno di un'istruzione sola del tipo seguente:

```
<!SGML "ISO 8879:1986"
…
…
…
>
```

In pratica, nel modello mostrato, le direttive occupano il posto dei puntini di sospensione.

Si osservi che lo standard originale ISO prevedeva la definizione '**`8879-1986`**', che successivamente è stata modificata nel modo mostrato, ovvero '**`8879:1986`**'. Lo stesso ragionamento vale per gli altri standard ISO che prevedono l'indicazione dell'anno.

Esiste una variante recente allo standard ISO 8879:1996 e precisamente si tratta di cambiamenti pensati per facilitare la comunicazione attraverso la rete. La stringa che fa riferimento a questo standard esteso è:

```
"ISO 8879:1986 (WWW)"
```

La si ritrova in particolare nella dichiarazione dell'HTML 4.∗ e nell'XML.

In questo capitolo vengono mostrate solo alcune direttive che possono essere utili per capire il senso della dichiarazione SGML. Per approfondire lo studio di questo linguaggio, bisogna procurarsi la documentazione originale ISO.

51.3.1 Codifica

La codifica dei caratteri utilizzata nel sorgente SGML non può essere ignorata, soprattutto perché alcuni codici hanno significati speciali che vanno oltre il carattere vero e proprio. Le direttive riferite alla codifica del sorgente iniziano con la parola chiave '**`CHARSET`**' che delimita la sezione relativa:

```
CHARSET
    definizione_riferita_all'insieme_di_caratteri
    …
```

In generale, si inizia con la definizione di un insieme standard di riferimento, attraverso l'uso di un identificatore standard:

```
BASESET insieme_di_caratteri
```

L'identificatore che definisce lo standard è normalmente una stringa abbastanza dettagliata. L'esempio seguente definisce l'insieme di partenza corrispondente all'ISO 646:1983, ovvero all'ASCII tradizionale:

```
BASESET  "ISO 646:1983//CHARSET
         International Reference Version (IRV)//ESC 2/5 4/0"
```

La direttiva appare su due righe, ma si tratta solo di una possibilità e non di una necessità, tanto che in alcuni casi la si può vedere anche distribuita su tre righe. Dopo la definizione dell'insieme di partenza, si può descrivere nel dettaglio l'utilizzo e la conversione dei codici corrispondenti ai caratteri:

```
DESCSET
    inizio  quantità  {corrispondenza | UNUSED}
    …
```

Si osservi l'esempio:

```
    DESCSET
          0  9 UNUSED
          9  2  9
         11  2 UNUSED
         13  1 13
         14 18 UNUSED
         32 95 32
        127  1 UNUSED
```

Il primo numero indica il codice corrispondente al carattere iniziale di un raggruppamento composto da una sequenza di ***n*** caratteri; il secondo valore indica una quantità di caratteri che possono essere ignorati oppure anche trasformati, partendo dal codice rappresentato dal terzo valore.

Nell'esempio, i codici che vanno da 0 a 8, in decimale (nel senso che si tratta di valori espressi in base 10), non sono utilizzati; inoltre i codici da 9 a 10 vengono convertiti con il codice 9 e seguenti (in pratica non vengono convertiti affatto). In sostanza, ciò che mostra l'esempio non ha lo scopo di convertire alcunché, ma solo di filtrare codici inutili: vengono lasciati passare i caratteri grafici, a partire dallo spazio, oltre a *<HT>*, *<LF>* e *<CR>*. Volendo esprimere la cosa in modo più esplicito, si possono usare anche dei commenti descrittivi:

```
    DESCSET
          0  9 UNUSED
          9  1  9       -- HT --
         10  1 10       -- LF --
         11  2 UNUSED
         13  1 13       -- CR --
         14 18 UNUSED
         32 95 32       -- SP e altri caratteri grafici --
        127  1 UNUSED
```

La sequenza di direttive '**BASESET**' e '**DESCSET**' può anche essere ripetuta, quando dopo l'ASCII normale, i primi 7 bit, si vuole fare riferimento a qualcosa di più. Per esempio, la dichiarazione relativa alla codifica dell'HTML 3.2, si presenta come si vede di seguito:

```
CHARSET
     BASESET  "ISO 646:1983//CHARSET
               International Reference Version
               (IRV)//ESC 2/5 4/0"
     DESCSET  0   9   UNUSED
              9   2   9
              11  2   UNUSED
              13  1   13
              14  18  UNUSED
              32  95  32
              127 1   UNUSED
     BASESET  "ISO Registration Number 100//CHARSET
               ECMA-94 Right Part of
               Latin Alphabet Nr. 1//ESC 2/13 4/1"
     DESCSET  128  32   UNUSED
              160  96    32
```


Rispetto a quanto già visto si aggiunge il riferimento allo standard ISO 8859-1 (Latin 1). Si può vedere che vengono esclusi i primi 32 codici a partire dal numero 128, che non contengono simboli grafici utili.

51.3.2 Capacità

Per qualche ragione storica, che ormai non avrebbe più motivo di sussistere, è prevista una sezione attraverso la quale si definisce la capacità elaborativa dell'analizzatore SGML. Si tratta di stabilire dei limiti di spazio per la gestione di una serie di informazioni. In generale, non dovrebbe essere determinante la dimensione da dare ai vari attributi riferiti a questa capacità; tuttavia, si tratta di un'indicazione che rimane, per la quale si fa riferimento allo standard, oppure si indica semplicemente che non ci sono limiti. Nel primo caso si indica,

```
CAPACITY PUBLIC  "ISO 8879:1986//CAPACITY Reference//EN"
```

nel secondo soltanto

```
CAPACITY NONE
```

A titolo di esempio si mostra anche la direttiva relativa riferita all'HTML 3.2 e 4:

```
CAPACITY        SGMLREF
                TOTALCAP        150000
                GRPCAP          150000
                ENTCAP          150000
```

Si osservi la parola chiave '**SGMLREF**' che può essere usata anche altrove. Rappresenta il riferimento ai valori predefiniti SGML, prima di modificarli o integrarli con le richieste successive.

51.3.3 Ambito

La sintassi del linguaggio SGML può essere alterata in parte, attraverso una serie di direttive descritte nella prossima sezione. L'ambito della definizione della sintassi SGML può essere controllato attraverso la direttiva '**SCOPE**':

```
SCOPE DOCUMENT | INSTANCE
```

La direttiva '**SCOPE DOCUMENT**' indica che la sintassi si applica sia al DTD, sia al sorgente SGML; nell'altro caso, '**SCOPE INSTANCE**' si riferisce solo al sorgente, mentre il DTD va interpretato in base alla sintassi standard predefinita (la ***sintassi concreta di riferimento***).

Di solito si usa la direttiva '**SCOPE DOCUMENT**'.

51.3.4 Sintassi concreta

La ***sintassi concreta*** è ciò che definisce i delimitatori dei marcatori SGML, il ruolo dei codici di controllo e altri dettagli riferiti alla sintassi SGML. In particolare si parla di sintassi concreta di riferimento quando si vuole indicare quella predefinita, ovvero quella a cui si fa riferimento di solito. Le direttive che compongono la definizione della sintassi concreta sono introdotte dalla sezione '**SYNTAX**', a cui spesso segue la stringa di un identificatore pubblico, per richiamare inizialmente una serie di caratteristiche standard che poi vengono alterate o integrate dalle direttive successive:

```
SYNTAX   PUBLIC  "ISO 8879:1986//SYNTAX Reference//EN"
```

51.3.4.1 Caratteri da evitare

La prima cosa che si specifica all'interno della dichiarazione della sintassi concreta è l'elenco dei numeri decimali corrispondenti ai codici, o caratteri, che non devono essere usati nel testo del sorgente. Questi non vengono passati all'applicazione successiva dall'analizzatore SGML. All'interno dei codici esclusi in questo modo ci possono essere comunque simboli o caratteri di controllo che servono in altri ambiti, come viene mostrato in seguito.

La codifica a cui si fa riferimento, non è quella ottenuta dopo la trasformazione con la direttiva '**DESCSET**' della sezione '**CHARSET**', ma quella della stessa direttiva della sezione '**SYNTAX**', come viene descritto tra poco.

La direttiva in questione è molto semplice; spesso, quando si tratta dell'ASCII, si utilizza direttamente l'esempio seguente:

```
SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
         16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
         127
```

51.3.4.2 Codifica nell'ambito della sintassi concreta

Nell'ambito della definizione della sintassi concreta, è necessario specificare nuovamente la codifica di partenza e la conversione eventuale. Tutto procede esattamente come è già stato visto in precedenza, nella sezione '**CHARSET**', con la differenza che in generale si preferisce lasciare tutto come si trova:

```
BASESET  "ISO 646:1983//CHARSET
         International Reference Version
         (IRV)//ESC 2/5 4/0"
DESCSET  0 128 0
```

L'esempio si riferisce al caso in cui si utilizzi solo l'ASCII. Comunque, si può osservare che la direttiva '**DESCSET**' non esclude alcunché e non trasforma alcun carattere.

51.3.4.3 Codici con funzioni speciali

Si possono definire alcuni codici con funzioni speciali, attribuendo loro un nome, a cui si accede con macro del tipo '***&#nome;***'. Spesso si fa uso di queste macro nel DTD, precisamente nelle mappe di sostituzione. Si dovrebbe ricordare che la macro '**&RE;**' fa riferimento convenzionalmente alla fine del record. Si osservi l'esempio seguente:

```
FUNCTION
    RE    13
    RS    10
    SPACE 32
    TAB   SEPCHAR 9
```

Si tratta della direttiva '**FUNCTION**' a cui segue la dichiarazione di una serie di nomi, abbinati al codice relativo. Si può osservare il caso del nome '**TAB**', a cui si aggiunge la parola chiave '**SEPCHAR**': si tratta effettivamente del carattere *<HT>*, che però, ai fini della sintassi concreta, viene tradotto con ciò che corrisponde a '**&SPACE;**', cioè uno spazio normale.

L'esempio mostra la definizione tipica di questa direttiva. Si può osservare che '**RE**' è abbinato a *<CR>*, per cui si suppone che il file sorgente SGML sia organizzato in modo da avere dei codici di interruzione di riga pari a *<CR><LF>*, come avviene nei sistemi Dos. Dipende molto dall'analizzatore SGML come funziona la cosa. In pratica, l'analizzatore potrebbe convertire autonomamente il file in questo modo, oppure potrebbe fare altre considerazioni.

51.3.4.4 Nomi

Attraverso la sottosezione '**NAMING**' è possibile definire quali caratteri possono essere usati nei «nomi». In questo modo si intendono i nomi degli elementi, delle entità, degli attributi e di alcuni tipi di valori da associare agli attributi.

In generale, si fa riferimento alle lettere latine dell'alfabeto inglese e alle cifre numeriche, tenendo conto che in generale è concesso solo di iniziare con una lettera. Per modificare questo assunto si interviene in direttive particolari, che limitano il primo carattere, oppure quelli restanti.

- `LCNMSTRT "`***caratteri_ulteriori***`"`

  `UCNMSTRT "`***caratteri_ulteriori***`"`

  Lower case name start, Upper case name start

  Descrivono rispettivamente il primo carattere minuscolo e maiuscolo. In generale, si indica semplicemente la stringa nulla, '**""**'.

- `LCNMCHAR "`***caratteri_ulteriori***`"`

  `UCNMCHAR "`***caratteri_ulteriori***`"`

  Lower case name characters, Upper case name characters

  Descrivono rispettivamente i caratteri successivi al primo, minuscoli e maiuscoli. In generale, si indica semplicemente la stringa nulla, '**""**'.

- `NAMESTRT `***elenco_codici***

  `NAMECHAR `***elenco_codici***

  Name start, Name characters

  Descrivono rispettivamente i codici utilizzabili nel primo carattere e in quelli restanti. Si usano queste direttive particolarmente nella definizione di XML.

- `NAMECASE`

  Si tratta di un'ulteriore sotto-sottosezione, con la quale si definisce la trasformazione o meno in maiuscolo:

  – `GENERAL YES|NO`

    in questo caso si controlla la conversione in maiuscolo di tutti i nomi, tranne le entità (nell'SGML tradizionale si attiva questa opzione);

  – `ENTITY YES|NO`

    si controlla la conversione in maiuscolo dei nomi di entità e dei loro riferimenti: le macro (nell'SGML tradizionale non si attiva questa opzione).

Nell'SGML normale si utilizza abitualmente la sezione '**NAMING**' nel modo seguente:

```
NAMING
    LCNMSTRT  ""
    UCNMSTRT  ""
    LCNMCHAR  "-."
    UCNMCHAR  "-."
    NAMECASE
        GENERAL     YES
        ENTITY      NO
```

In questo modo, si può osservare che i nomi possono contenere anche il trattino ('**-**') e il punto ('**.**'), ma non possono iniziare così; inoltre, tutti i nomi, tranne quelli delle entità, vengono convertiti in maiuscolo (si parla di ***normalizzazione***), per cui non fa differenza in che modo sono stati scritti.

51.3.4.5 Delimitatori

La sottosezione introdotta dalla parola chiave '**DELIM**' può servire per intervenire nella definizione dei delimitatori. In generale non si modifica nulla e ci si limita a confermare lo standard di riferimento, attraverso la parola chiave '**SGMLREF**':

```
DELIM
    GENERAL   SGMLREF
    SHORTREF  SGMLREF
```

A volte viene disabilitato l'uso delle mappe di sostituzione nel DTD, attraverso la direttiva '**SHORTREF NONE**', come avviene in XML.

Nell'HTML 4 e in XML è stata aggiunta la possibilità di indicare delle macro carattere nella forma '**&x*n*;**', per rappresentare i caratteri attraverso cifre esadecimali. Per ottenere questo risultato, dopo la direttiva '**GENERAL SGMLREF**', si aggiunge la dichiarazione di '**HCRO**':

```
        DELIM
            GENERAL   SGMLREF
            HCRO      "&#38;#x"
            SHORTREF  SGMLREF
```

Naturalmente, in XML ci sono poi altre aggiunte, che qui non vengono mostrate.

51.3.4.6 Nomi riservati

Alcune nomi che hanno significati speciali possono essere modificati nella sottosezione '**NAMES**'. In generale, queste cose non si fanno, per cui si abbina semplicemente la dichiarazione predefinita: '**SGMLREF**':

```
        NAMES
            SGMLREF
```

51.3.4.7 Quantità

Nell'ambito della sintassi concreta è possibile definire il limite a una serie di quantità. Di solito non ci si preoccupa di queste cose, oppure si scrivono direttive per richiedere limiti molto elevati. Per fare riferimento allo standard, si utilizza la parola chiave '**SGMLREF**' come al solito:

```
        QUANTITY SGMLREF
```

Eventualmente si aggiungono le varianti che si ritiene necessario apportare. L'esempio seguente è tratto dalla configurazione predefinita di SP e appare evidente l'intenzione di estendere al massimo i limiti, anche senza spiegare nel dettaglio il significato di ogni parametro:

```
        QUANTITY SGMLREF
            ATTCNT    99999999
            ATTSPLEN  99999999
            DTEMPLEN  24000
            ENTLVL    99999999
            GRPCNT    99999999
            GRPGTCNT  99999999
            GRPLVL    99999999
            LITLEN    24000
            NAMELEN   99999999
            PILEN     24000
            TAGLEN    99999999
            TAGLVL    99999999
```

Con XML, o comunque con la dichiarazione «Web SGML», '**ISO 8879:1986 (WWW)**', è possibile usare una forma differente e più intuitiva per indicare che non si vogliono porre limiti:

```
         QUANTITY NONE
```

51.3.5 Proprietà

L'ultima sezione della dichiarazione SGML serve a raccogliere la definizione delle proprietà: '**FEATURES**'. Contiene in particolare tre sottosezioni intitolate rispettivamente '**MINIMIZE**', '**LINK**' e '**OTHER**'. Non è il caso di approfondire queste definizioni, a parte qualche direttiva che può essere interessante.

Per cominciare, conviene osservare la sezione '**FEATURES**' dell'HTML 4:

```
FEATURES
    MINIMIZE
        DATATAG  NO
        OMITTAG  YES
        RANK     NO
        SHORTTAG YES
    LINK
        SIMPLE   NO
        IMPLICIT NO
        EXPLICIT NO
    OTHER
        CONCUR   NO
        SUBDOC   NO
        FORMAL   YES
```


Nella sottosezione '**MINIMIZE**' è importante tenere in considerazione l'opzione '**DATATAG**', che in generale è bene sia disattivata come appare nell'esempio. Questa dovrebbe servire per specificare una stringa che nel testo deve essere presa in considerazione come una chiusura implicita di un elemento. L'opzione '**OMITTAG**' consente di utilizzare le regole di minimizzazione nel DTD.

La sottosezione '**OTHER**' permette di definire delle caratteristiche interessanti riguardo all'organizzazione del DTD, del sorgente e dei cataloghi. L'opzione '**CONCUR**' consente, se attivata, di gestire più DTD nello stesso documento. Ciò può servire quando è consentita l'aggregazione di più sorgenti che a loro volta utilizzano DTD differenti. Data la complessità che si creerebbe in questo modo, tale opzione viene disabilitata normalmente. L'opzione '**SUBDOC**' permette, se abilitata, di aggregare più sorgenti SGML assieme (che di solito condividono lo stesso DTD implicitamente); se si abilita l'opzione occorre aggiungere l'indicazione del numero massimo di livelli di annidamento a cui si può arrivare. L'opzione '**FORMAL**', se attivata, serve a richiedere l'uso corretto degli identificatori pubblici; se non è attivata, l'identificazione può avvenire in modo meno rigoroso.

L'esempio seguente mostra l'impostazione tradizionale di un sistema SGML:

```
    FEATURES

        MINIMIZE
            DATATAG   NO
            OMITTAG   YES
            RANK      YES
            SHORTTAG  YES

        LINK
            SIMPLE    YES 1000
            IMPLICIT  YES
            EXPLICIT  YES 1

        OTHER
            CONCUR    NO
            SUBDOC    YES 99999999
            FORMAL    YES
```

51.3.6 Applicazione di una dichiarazione SGML in pratica

La dichiarazione SGML può essere attribuita attraverso il catalogo, con la direttiva '**SGMLDECL**':

```
SGMLDECL "HTML4.dcl"
```

L'esempio mostra il riferimento al file '`HTML4.dcl`', contenente la dichiarazione SGML desiderata.

Potrebbe essere impossibile selezionare tra più dichiarazioni alternative. In tal caso, diventa necessario predisporre più cataloghi, uno per ogni tipo di dichiarazione che si intende utilizzare.

51.3.7 Esempio conclusivo

Per concludere viene mostrato un esempio completo di una dichiarazione SGML realizzata per poter utilizzare nel sorgente la codifica ISO 8859-1, che potrebbe essere adatta alle situazioni più comuni (appare anche la sezione '**APPINFO**' che non è stata descritta). Altri esempi possono essere ottenuti dal pacchetto SP sorgente, nel quale si può trovare anche la dichiarazione di XML.

```
<!SGML "ISO 8879:1986 (WWW)"
    CHARSET
        BASESET  "ISO 646-1983//CHARSET
                  International Reference Version
                  (IRV)//ESC 2/5 4/0"
        DESCSET
            0  9 UNUSED
            9  2  9
            11  2 UNUSED
            13  1 13
            14 18 UNUSED
            32 95 32
            127  1 UNUSED
        BASESET  "ISO Registration Number 100//CHARSET
```

```
                ECMA-94 Right Part of
                Latin Alphabet Nr. 1//ESC 2/13 4/1"
        DESCSET  128  32   UNUSED
                 160  96    32

    CAPACITY PUBLIC  "ISO 8879:1986//CAPACITY Reference//EN"

    SCOPE    DOCUMENT

    SYNTAX

        SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
                 15 16 17 18 19 20 21 22 23 24 25 26 27 28
                 29 30 31 127

        BASESET  "ISO 646-1983//CHARSET International
                 Reference Version
                 (IRV)//ESC 2/5 4/0"

        DESCSET
            0 128 0

        FUNCTION
            RE                  13
            RS                  10
            SPACE               32
            TAB       SEPCHAR   9

        NAMING
            LCNMSTRT  ""
            UCNMSTRT  ""
            LCNMCHAR  "-."
            UCNMCHAR  "-."
            NAMECASE
                GENERAL     YES
                ENTITY      NO

        DELIM
            GENERAL   SGMLREF
            HCRO      "&#38;#x"
            SHORTREF  SGMLREF

        NAMES    SGMLREF

        QUANTITY NONE

    FEATURES

        MINIMIZE
            DATATAG   NO
            OMITTAG   YES
            RANK      NO
            SHORTTAG  NO

        LINK
            SIMPLE    YES 1000
            IMPLICIT  YES
            EXPLICIT  YES 1

        OTHER
            CONCUR    NO
            SUBDOC    YES 99999999
            FORMAL    YES

    APPINFO NONE
>
```

51.4 Riferimenti

«

- C. M. Sperberg-McQueen, Lou Burnard, *Guidelines for Electronic Text Encoding and Interchange (TEI P4)*, in particolare il secondo capitolo: *A gentle introduction to XML*, http://www.tei-c.org/release/doc/tei-p4-doc/html/
- *The SGML Newsletter*, http://wayback.archive.org/web/2009*/http://architag.com/tag/
- Oasis, *Core Standards for Markup Language Technologies*, http://xml.coverpages.org/coreStandards.html

- Oasis, *Standard Generalized Markup Language (SGML)*, http://xml.coverpages.org/sgml.html
- Vidar Bronken Gundersen, Rune Mathisen, *SGML/XML character entity reference*, http://www.bitjungle.com/isoent/

[1] In questo momento può apparire strano l'uso di questa forma di eccezione. Tuttavia, per comprenderne meglio il senso, occorrerebbe conoscere come funzionano le entità parametriche che sono descritte più avanti. Con queste si può definire un modello del contenuto attraverso una sorta di variabile e, in tal caso, potrebbe essere conveniente l'indicazione di una o più eccezioni, sia in aggiunta che in detrazione.