Introduzione ad XML

Fabio Vitali

Introduzione

Qui esaminiamo alcuni aspetti di XML, in particolare sintattici e di filosofia d'uso:

- Vantaggi di XML
- Applicazioni XML
- Sintassi dei DTD
  - Elementi
  - Attributi
  - Entità
  - ~~Notazioni~~
  - ~~Marked section~~
- Altre caratteristiche sintattiche di XML
- XML e Whitespace
- Analisi dei contenuti

XML

XML (Extensible Markup Language [sic!]) è un meta-linguaggio di markup, progettato per lo scambio e la interusabilità di documenti strutturati su Internet.

XML prevede una sintassi semplificata rispetto a SGML, e definisce contemporaneamente una serie piuttosto lunga di linguaggi associati: uno per i link, uno per i nomi di tag, uno per i fogli di stile, uno per la descrizione di meta-informazioni, ecc.

XML si propone di integrare, arricchire e, nel lungo periodo, sostituire HTML come linguaggio di markup standard per il World Wide Web.

Perché XML?

HTML nacque come un DTD di SGML (non proprio!!!), che permetteva di mettere in rete documenti di un tipo molto specifico, semplici documenti di testo con qualche immagine e dei link ipertestuali.

Con il successo del WWW, HTML venne iniziato ad usare per molti scopi, molti più di quelli per cui era stato progettato.

Si iniziò ad abusare dei tag di HTML per gli effetti grafici che forniva, più che per gli aspetti strutturali o semantici.

Si iniziarono a desiderare elaborazioni sofisticate sui dati HTML, elaborazioni che non era possibile fornire.

Si iniziò a trovare limitata la capacità grafica di HTML, anche abusando dei tag.

Perché non SGML?

SGML ha molti pregi, ma ha dalla sua una complessità d'uso e di comprensione notevole. Inoltre, a SGML mancano caratteristiche di notevole importanza per l'uso pratico, come link ipertestuali e specifiche grafiche.

L'avvento di HTML ha fatto capire come i linguaggi di markup siano ormai maturi per essere compresi dal largo pubblico, ma che la semplicità d'uso di HTML doveva costituire un elemento di partenza.

XML contiene tutte le caratteristiche di SGML che servono per creare applicazioni generali senza scendere nel livello di dettaglio e pedanteria richiesti da SGML.

I vantaggi di XML (1)

Documenti auto-descrittivi

- La scelta dei nomi degli elementi può essere fatta per facilitare la comprensione del ruolo strutturale dell'elemento.
- Inoltre, l'uso di un DTD può esplicitare le regole di composizione ed i rapporti possibili tra le varie parti dei documenti.

Struttura navigabile dei documenti

- La rigida struttura ad albero e l'assenza di regole di minimizzazione rendono semplice la visualizzazione e l'analisi della struttura del documento, e la possibilità di visualizzare il documento è indipendente dal foglio di stile che vi si applica.

I vantaggi di XML (2)

Platform-independence

- XML è uno standard aperto, e chiunque può realizzare strumenti che lo usino come formato di dati.

Facile convertibilità a formati Web

- La totale interdipendenza tra XML, SGML, HTML etc. fa sì che la conversione tra formati interni e formati per il Web sia facile.

I vantaggi di XML (3)

Strutturazione gerarchica dei documenti

- Esistono molti formati di dati generici per l'intescambio di dati, ma sono tutti organizzati linearmente. XML permette strutture ad albero.

Ripetibilità degli elementi

- XML permette di definire formalmente elementi ripetibili. Questo permette strutture più flessibili e complesse di altri formati di dati

Content model misti

- XML trova un punto di equilibrio tra i formati dati per l'interscambio di dati e i formati per la strutturazione di documenti di testo. I content model misti (elementi che possono contenre sia altri elementi che testo) infatti permettono di inserire caratterizzazioni semantiche non soloper interi elementi, ma anche all'interno di elementi di testo contenitori (ad esempio, i paragrafi).

Quali applicazioni XML?

Data Interchange

- Ogni volta che più programmi si debbono scambiare dati, ci sono problemi di compatibilità. Ogni programma ha le proprie assunzioni in termini di caratteri, separatori, ripetibilità di elementi, differenza tra elementi vuoti e assenti, ecc.
- XML si propone come la sintassi intermedia più semplice per esprimere dati anche complessi in forma indipendente dall'applicazione che li ha creati.

Document publishing

- XML è ideale come linguaggio per esprimere documenti strutturati o semi strutturati, e per esprimerli in maniera indipendente dalla loro destinazione finale.
- Lo stesso documento XML può essere preso e trasformato per la stampa, il Web, il telefonino, l'autoradio.

Cosa si fa con XML? (1)

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

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

Cosa si fa con XML? (2)

Accesso a database eterogenei

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

Cosa si fa con XML? (3)

Computazioni client-side

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

Cosa si fa con XML? (4)

Viste selettive

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

Cosa si fa con XML? (5)

Agenti Web (ora: Web applications)

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

Quando scegliere XML? (1)

Quali sono le condizioni per adottare XML in un progetto?
Ovviamente:

- E' nuovo
- E' di moda
- E' compatibile con Web e con .NET
- Può essere imposto dal committente
- Può essere imposto dai partner

Ma ci sono almeno quattro ***buone*** ragioni per XML:

- Produzione di documenti automatici
- Gestione indipendente di produzione e uso di dati
- Elaborazione di dati con aspetti strutturali complessi
- Elaborazione di dati strutturati in contenitori semi-strutturati

Quando scegliere XML? (2)

Produzione di documenti automatici

- XML è la soluzione in assoluto più elegante (anche se ad oggi ancora faticosa) per integrare collezioni di dati strutturati sul Web.
- Documenti dinamici, che mescolano blocchi testuali con output tabellari di informazioni strutturate, sono facilmente esprimibili in XML, e gli strumenti attuali si concentrano su questo, per il momento.
- Integra e sostituisce le tecnologie server-side di accesso ai dati: ASP, PHP, server-side Javascript, ecc.

Quando scegliere XML? (3)

Gestione indipendente di produzione ed uso di dati

- Spesso l'interscambio di dati avviene all'interno di un workflow controllato e noto. In questo caso, ***data producers*** e ***data consumers*** sono creati ad hoc per lo specifico flusso informativo. XML è una complicazione inutile.
- Tuttavia esistono delle situazioni in cui non c'è progettazione integrata di producer e consumer. In questo caso, un'adeguata progettazione del producer facilita molto il lavoro di tutti i possibili consumer
- XML è strutturato, auto-esplicativo, enfatizza la descrizione del dato più che del suo scopo nella elaborazione. E' quindi ideale per le situazioni in cui l'elaborazione non è nota in anticipo.

Quando scegliere XML? (4)

Elaborazione di dati con aspetti strutturali complessi

- I database utilizzano le relazioni per ogni tipo di esigenza: dalla descrizione di connessioni logiche tra entità concettualmente diverse, alla gestione di dati strutturati in maniera complessa.
- Ad esempio, è complicato gestire, in una tabella, record con un numero variabile di campi, o situazioni alternative complesse.
- XML prevede strutture con blocchi ripetuti, alternativi, facoltativi. La descrizione di queste strutture in XML è molto più ***naturale*** che con DB relazionali.

Quando scegliere XML? (5)

Elaborazione di dati in contenitori semi-strutturati

- A volte l'informazione ha uno stato naturale semi-strutturato (e.g., documenti testuali), al cui interno esistono informazioni atomiche su cui è necessario attivare computazioni.
- La soluzione classica è di estrarre le informazioni atomiche, metterle in un DB tradizionale, e buttare via il contenitore naturale. Questo ha il grosso difetto di eliminare il contesto e omogeneizzare in maniera forzata informazioni organizzate diversamente.
- XML permette di inserire all'interno di strutture documentarie (pensate per la visualizzazione) tag di natura descrittiva utilizzabili per elaborazioni sofisticate.

Un esempio: XMLNews (1)

XMLNews definisce il contenuto testuale e le meta-informazioni di notizie da agenzia stampa. E' una parte dello standard denominato ***News Industry Text Format*** (NITF), sviluppato dal ***International Press Telecommunications Council*** e dalla ***Newspaper Association of America***.

XMLNews è composto di due parti:

- XMLNews-Story è un DTD XML per descrivere in maniera variamente arricchita il testo delle notizie
- XMLNews-Meta definisce il formato delle meta-informazioni per notizie d'agenzia. E' conforme al Resource Description Framework (RDF), e on si riferisce solo alle notizie testuali, ma anche a immagini, video-clip, ecc.

Un esempio: XMLNews (2)

XMLNews-Story: il testo di una notizia di agenzia è diviso in tre parti: l'head contiene informazioni di organizzazione, mentre il body è a sua volta diviso in intestazione e contenuto.

```
<?xml version="1.0"?>
<nitf>
  <head> <title>Colombia Earthquake</title> </head>
  <body>
    <body.head>
      <hedline><hl1>143 Dead in Earthquake</hl1></hedline>
      <byline><bytag>By Jared Kotler, AP </bytag></byline>
      <dateline>
        <location>Bogota, Colombia</location>
        <story.date>January 25 1999 7:28 ET</story.date>
      </dateline>
    </body.head>
    <body.content> … </body.content>
  </body>
</nitf>
```


Un esempio: XMLNews (3)

XMLNews-Story: Il body ha un markup minimale di struttura del testo:

```
<?xml version="1.0"?>
<nitf> <head> … </head> <body> <body.head> … </body.head>
    <body.content>
      <p>An earthquake struck western Colombia on Monday,
         killing at least 143 people and injuring more than
         900 as it toppled buildings across the country's
         coffee-growing heartland, civil defense officials
         said.
      </p>
      <p>The early afternoon quake had a preliminary
         magnitude of 6, according to the U.S. Geological
         Survey in Golden, Colo. Its epicenter was located
         in western Valle del Cauca state, 140 miles west
         of the capital, Bogota.
      </p>
    </body.content> </body>
</nitf>
```


Un esempio: XMLNews (4)

XMLNews-Story: Però è possibile in qualunque momento aggiungere informazioni inline:

```
<p>An <event>earthquake</event> struck <location>western
<country>Colombia</country></location> on <chron
norm="19990125">Monday</chron>, killing at least 143
people and injuring more than 900 as it toppled buildings
across the country's coffee-growing heartland,
<function>civil defense officials</function> said.</p>
```

Questo permette di arricchire la storia con molte informazioni e in maniera semi-automatica:

- **Nella ricerca**: è possibile cercare tutto quello che è successo in Colombia, o cosa è successo in una certa data.
- **Nella presentazione**: un provider potrebbe fornire semi-automaticamente dei link o delle cartine della Colombia.
- **Nell'organizzazione delle news**: è possibile cercare tutti i terremoti effettivi, e non le notizie che ne usano la parola, magari figurativamente.

Un esempio: XMLNews (5)

XMLNews-Meta: Assieme ad ogni notizia, vengono scritte delle informazioni sulla notizia, che possono avere una distribuzione separata.

XMLNews-Meta permette di gestire insieme informazioni come:

- Informazioni sul contenuto della notizia (titolo, lingua, formato, ecc.)
- Informazioni sulle date della notizia: creazione, pubblicazione, scadenza, ecc.
- Informazioni sulla provenienza ed attendibilità della notizia
- Informazioni sui possessori dei diritti di distribuzione e copyright
- Informazioni di classificazione ed organizzazione
- Link a documenti connessi: versioni precedenti, seguenti, ed altre notizie connesse.

Cosa c’è con XML?

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

Noi di occupiamo, tra gli altri, di:

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

XML 1.0

- Una raccomandazione W3C del 10 febbraio 1998.
- È definita come un sottoinsieme di SGML
- URL ufficiale: http://www.w3.org/TR/REC-xml
- Traduzione ufficiale in italiano: **`http://www.iat.cnr.it/xml/REC-xml-19980210-it.html`**
- Molto più formalizzata della grammatica di SGML, usa una notazione formale, ***Extended Backus-Naur Form***.

Criteri di progettazione di XML (1)

Nel documento ufficiale di XML si elencano i seguenti obiettivi progettuali di XML:

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

Criteri di progettazione di XML (2)

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

Questi goal sono stati sostanzialmente raggiunti.

Criteri di progettazione di XML (3)

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

Criteri di progettazione di XML (4)

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

Criteri di progettazione di XML (5)

8. La progettazione XML deve essere formale e concisa.
   - La specifica di SGML è composta di un documento di oltre 300 pagine in stile ottuso e burocratico. Il manuale SGML ne richiede più di 600, e comunque non è leggibile facilmente.
   - Inoltre non è neanche immediatamente utilizzabile da un programmatore per realizzare tool (poche definizioni formali, difficile dedurre la grammatica del linguaggio).
   - La scelta di formalismi nitidi e pochi commenti ha permesso la creazione di una specifica XML notevolmente più corta (~40 pagg.) e immediatamente utilizzabile dai realizzatori di tool (sintassi BNF).
9. I documenti XML devono essere facili da creare.

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

Criteri di progettazione di XML (6)

10. Non ha importanza l'economicità del markup XML.
    - Le esigenze di economicità di markup (***terseness***) di SGML avevano portato all'adozione di molte pratiche di minimizzazione dei caratteri, che però rendevano i documenti poco leggibili e molto più complicati da parsare.
    - XML non ha meccanismi di minimizzazione, e dove si poteva scegliere tra economicità e chiarezza, si è scelta la chiarezza.

Esistono poi due obiettivi progettuali non riportati:

A. Supporto per l'internazionalizzazione
    - XML deve funzionare con tutti i set di caratteri.

B. Desperate Perl hacker
    - Il programmatore a cui viene imposto di eseguire un compito di modifica globale su una grande quantità di documenti e che riesce a farla applicando un qualche script semplice sulla struttura pulita dei documenti XML.

XML e Unicode

XML (come Java) abbandona completamente ASCII e le codifiche ad un byte, e si basa direttamente su Unicode.

Questo porta a due vantaggi nei riguardi dell'internazionalizzazione:

- È possibile scrivere documenti misti, senza ricorrere a trucchi strani per identificare la parte che usa un alfabeto dalla pare che ne adopera un altro.
- Un documento scritto in un linguaggio non latino non deve basarsi su parametri esterni per essere riconosciuto come tale, ma la codifica stessa dei caratteri lo identifica.

Documenti ben formati o validi

XML distingue due tipi di documenti rilevanti per le applicazioni XML: i documenti ***ben formati*** ed i documenti ***validi***.

In SGML, un DTD è necessario per la validazione del documento. Anche in XML, un documento è **valido** se presenta un DTD ed è possibile validarlo usando il DTD.

Tuttavia XML permette anche documenti **ben formati**, ovvero documenti che, pur essendo privi di DTD, presentano una struttura sufficientemente regolare e comprensibile da poter essere controllata.

Documenti XML ben formati

Un documento XML si dice ben formato se:

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

Parser validanti e non validanti

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

Sintassi dei DTD

- Una precisazione
- `<!DOCTYPE … >`
- `<!ELEMENT … >`
- `<!ATTLIST … >`
- `<!ENTITY … >`: Entità generali
- `<!ENTITY % … >`: Entità parametriche
- Altre caratteristiche di XML

Una precisazione

- I DTD XML e SGML sono molto simili. A parte minime modifiche, ogni DTD XML è anche un DTD SGML.
- A noi interessa soprattutto fare documenti SGML che possano passare per documenti XML, e viceversa.
- Quindi esponiamo come sintassi del DTD solo quella comune a entrambi i linguaggi. Qui non guardiamo la sintassi di SGML che non è stata ereditata da XML.
- In seguito guarderemo con esattezza le differenze tra XML e SGML.

La dichiarazione di tipo

- Il <!DOCTYPE … > è la dichiarazione del tipo di documento. Essa permette alle applicazioni SGML di determinare le regole sintattiche da applicare alla verifica e validazione del documento.
- La dichiarazione non è, ma contiene o fa riferimento alla Document Type Definition, o DTD, ove vengono elencati gli elementi validi e i loro vincoli.
- Il DTD può essere posto in un file esterno, internamente al documento, o in parte esternamente ed in parte internamente.

- **N.B.: In XML il nome del DOCTYPE deve essere il nome del tag radice.**

Dichiarazione del DTD: <!DOCTYPE … >

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

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

Specifica di elementi: <!ELEMENT …>

```
<!ELEMENT nome content-model >
<!ELEMENT para (#PCDATA | bold)+ >
```

In SGML:

```
<!ELEMENT nome ST ET content-model >
<!ELEMENT para - - (#PCDATA | bold)+ >
```

- ST & ET: minimizzazione del tag iniziale (ST) e finale (ET): può assumere i valori '-' (***obbligatorio***) o 'o' (***omissibile***). In XML mancano.
- Content-model: la specificazione formale del contenuto permesso nell'elemento, secondo una sintassi specifica di gruppi di modelli.

Content model

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

ANY, EMPTY, #PCDATA

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

  Def.: `<!ELEMENT HR EMPTY>`
  Uso: “`<HR/>`”
- #PCDATA: (Parsed Character Data): il contenuto testuale del documento. Include caratteri ed entità generali. È naturalmente in numero multiplo.

Separatori

Separano specifiche determinando l'ordine o l'obbligatorietà:

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

In SGML anche:

- **'&' (ampersand)**: richiede la presenza di entrambe le specifiche, ma in qualunque ordine.
  Es.: **(a & b)**: ci debbono essere sia a che b, ma in qualunque ordine.

Operatori di ripetizione (1)

Permettono di specificare se un gruppo può comparire esattamente una volta, almeno una volta, oppure zero o più volte.

- ***Niente***: la specifica precedente deve comparire esattamente una volta.
  Es.: c, (a, b): a e b devono comparire in quest'ordine esattamente una volta. È lecito solo: cab. Si dice che è una specifica ***richiesta*** e ***non ripetibile***.
- ***? (punto interrogativo)***: la specifica precedente può e può non comparire, ma solo una volta.
  Es.: c, (a , b)?: a e b possono comparire una volta, ma possono non comparire. Sono lecite: c, cab. Si dice che è una specifica ***facoltativa*** e ***non ripetibile***.

Operatori di ripetizione (2)

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

Element content semplice

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

Un elemento contiene solo altri elementi, senza parti opzionali.

Dentro all'elemento sezione ci deve essere un titolo, seguito da un abstract, seguito da un para.

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


Element content con elementi facoltativi o ripetuti

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

Un elemento contiene solo altri elementi, ma alcuni possono essere opzionali e altre in presenza multipla.

Dentro all'elemento sezione ci deve essere un titolo, seguito facoltativamente da un abstract, seguito da almeno (ma anche più di) un para.

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


Element content complesso

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

Un elemento contiene solo altri elementi, ma gli operatori di ripetizione e i separatori permettono sequenze complesse di elementi.

Dentro all'elemento sezione ci deve essere un titolo, seguito da almeno uno di abstract o para, che poi possono ripetersi in qualunque ordine e in qualunque numero.

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


Character content

```
<!ELEMENT para #PCDATA >
```

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

```
<para>Questo &egrave; lecito</para>
```


Mixed content

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

Un elemento contiene sia caratteri stampabili ed entità, sia altri elementi.

```
<para>Questo &egrave; un paragrafo lecito con alcune
<bold> parole in grassetto </bold> e poi <bold>
ancora altre </bold>. </para>
```

Nota: in XML il content model misto ha come **UNICA FORMA** la selezione ripetibile di elementi alternativi in cui il primo è #PCDATA. **Ogni altra forma genera errori nel parser**.

In SGML non è obbligatorio che abbia questa forma, ma è buono stile.

Lista di attributi: <!ATTLIST … >

La dichiarazione <!ATTLIST… > permette di definire una lista di attributi lecita ad un elemento SGML dichiarato in precedenza.

```
<!ATTLIST nome
      nome-attributo-1   tipo-1   default-1
      nome-attributo-2   tipo-2   default-2
      nome-attributo-3   tipo-3   default-3
      …
>
```


Attributo di tipo stringa

```
<!ATTLIST doc
     linguaggio     CDATA     “HTML” >
```

CDATA significa “character data”, e indica qualunque sequenza di caratteri (tranne “<“ e le virgolette già usate come delimitatore), ma non entità o elementi.

Attributi di tipo lista

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

Solo uno dei valori elencati nella lista può essere accettato. Ogni altro valore genererà un errore.

Da notare che in SGML (ma non XML), ogni valore possibile in una lista di attributi dello stesso elemento deve essere unico. In SGML la definizione seguente genera un errore:

```
<!ATTLIST doc
   stato      (bozza | impaginato | finale) “bozza”
   posizione  (iniziale | mediana | finale) “iniziale”
>
```


Attributi di tipo predefinito

- `ID`: un identificativo univoco all'interno del documento.
- `IDREF` o `IDREFS`: un riferimento (o una lista di riferimenti) ad un identificativo definito altrove nel documento con un attributo ID. L'ID corrispondente deve esistere.
- `NMTOKEN` o `NMTOKENS`: un nome (o una lista di nomi). Un nome è una stringa di caratteri alfanumerici che include le lettere maiuscole e minuscole, i numeri e i caratteri ".", "-", "_", ":", ma non spazi, altri segni di punteggiatura, e altri caratteri particolari.

Attributi ID

```
<!ATTLIST X
      a        ID                    #REQUIRED
>
```

- Il tag iniziale di X può contenere un attributo chiamato “a”. Sono leciti solo valori unici su tutto il documento. L’elemento X assume un’identificabilità assoluta all’interno del documento: è un “luogo notevole” Poiché il valore deve essere sempre diverso, non è possibile specificare un valore di default.

- `<X a=“pluto”>adj</X><X a=“pippo”>bfg</X>`
- `<X a=“pluto”>adj</X><X a=“pluto”>bfg</X>`
  `<!-- errore  -->`

Attributi IDREF

```
<!ATTLIST X
      a        ID          #IMPLIED
      b        IDREF       #IMPLIED
>
```

- Il tag iniziale di X può contenere un attributo chiamato “a” ed uno chiamato “b”. I valori di “a” debbono essere unici.I valori di “b” debbono essere uguali ad un valore di “a” esistente da qualche parte nel documento.

- `<X a=“pluto”>adj</X><X b=“pluto”>bfg</X>`
- `<X a=“pluto”>adj</X><X b=“pippo”>bfg</X>`
  `<!-- errore  -->`

Valore di default letterale

L’attributo assume quel valore se non ne viene specificato un altro. Ad esempio, in

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

l’uso

```
<doc stato=“bozza”> Questo &egrave; un documento </doc>
```

e l’uso

```
<doc> Questo &egrave; un documento </doc>
```

sono uguali.

Valore di default #FIXED

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

Ad esempio, in

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

l'uso

```
<doc stato="finale"> Questo &egrave; un documento
</doc>
```

è un errore.

Valore di default #REQUIRED

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

Ad esempio, in

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

l'uso

```
<doc>Questo &egrave; un documento </doc>
```

è un errore.

Valore di default `#IMPLIED`

Non è obbligatorio specificare un valore per questo attributo. Se esiste, verrà considerato il valore fornito, altrimenti l'applicazione deve fornirne un valore proprio.

Gli attributi di tipo ID debbono avere un valore `#REQUIRED` (l'autore deve fornire un identificativo univoco ogni volta) o `#IMPLIED` (l'applicazione si preoccupa di fornire un identificativo interno).

Entità generali: <!ENTITY … >

Le entità generali sono elementi di contenuto definite nel DTD e richiamate nel documento.

Durante la lettura del documento i richiami delle entità vengono sostituite con il valore definito (espansione dell'entità)

Le entità generali sono permesse nel contenuto degli elementi e nei valori degli attributi.

Entità generali: definizione e uso

Definizione: `<!ENTITY nome valore>`
Uso: `&nome;`

Definizione (nel DTD):

```
<!ENTITY xml “Extensible Markup Language”>
<!ENTITY dataxml “5/3/2000”>
```

Uso (nel documento):

```
<para> <data n=“&dataxml;”>Presto</data> impariamo
l’&xml; </para>
```

Espansione:

```
<para> <data n=“5/3/2000”>Presto</data> impariamo
l’Extensible Markup Language </para>
```


Entità parametriche: <!ENTITY % … >

- Le entità parametriche sono elementi di specifica definiti nel DTD e usati nel DTD stesso, dopo la loro definizione.
- Servono per raccogliere in un'unica locazione definizioni di content model, o di attributi, o di valori comuni a molti elementi.
- Durante la lettura del DTD le entità parametriche vengono sostituite con il loro valore e questo viene usato per la definizione degli elementi.
- Invece del carattere '&' viene usato (anche nella definizione!!!) il carattere '%'.

Entità parametriche: definizione e uso

Definizione: `<!ENTITY % nome valore>`
Uso: `%nome;`

Definizione (nel DTD):

```
<!ENTITY % inline “(#PCDATA | bold)*”>
```

Uso (nel DTD):

```
<!ELEMENT para1 %inline;>
<!ELEMENT para2 (%inline; | italic)*>
```

Espansione:

```
<!ELEMENT para1 (#PCDATA | bold)*>
<!ELEMENT para2 ((#PCDATA | bold)* | italic)*>
```


Entità esterne

Non è necessario che il valore di espansione dell'entità sia specificato nella definizione.

Per modularità, condivisione, o dimensioni eccessive, può essere comodo inserire il valore dell'entità in un file a parte, e definire l'entità come esterna usando la keyword SYSTEM.

```
<!DOCTYPE mydoc [
<!ENTITY % blocco-dtd SYSTEM “blocco.dtd”>
<!ENTITY grosso-testo SYSTEM “testo.sgm”>
%blocco-dtd;
]>
<mydoc>
  <p>&grosso-testo;</P>
</mydoc>
```


Altre caratteristiche di XML

- La dichiarazione XML
- Sezioni CDATA
- Rigore sintattico (case sensitivity, nessuna minimizzazione)
- Entità predefinite
- Gestione del white space
- Attributi riservati

Dichiarazione XML (1)

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

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

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


Dichiarazione XML (2)

Esistono esattamente tre valori che possono essere messi in una dichiarazione XML:

- Il parametro “version” identifica quale versione di XML si sta usando. Per il momento, l’unico valore possibile è “1.0”. Necessario.
- Il parametro “encoding” permette di specificare, se il dubbio può sorgere, quale codifica di caratteri viene usata per il documento. Facoltativo.
- Il parametro “standalone” permette di specificare se tutto il contenuto del documento è interno alla risorsa o se ne esiste parte anche all'esterno (ad esempio in un'entità posta nel DTD esterno). Facoltativo. Se è assente è false.

Sezioni CDATA

- A volte può essere comodo inserire un blocco di caratteri comprendenti anche ‘&’ e ‘<‘, senza preoccuparsi di nasconderli dentro ad entità.
- Si usa allora la sezione CDATA, che ha la seguente sintassi:

```
<![CDATA[ dati liberi comprendenti & e < ]]>
```

- L’unica sequenza di caratteri non accettabile è la sequenza ‘]]>’, che definisce la fine della sezione CDATA
- Il parser XML passa all’applicazione finale tutti i caratteri che trova fino alla sequenza]]>

```
<para>In HTML, “<![CDATA[ <h1>Questo &egrave; un
titolo</h1>]]>” indica un titolo </para>
```


Altre caratteristiche di XML (1)

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

```
<EMPTY/>
```

- **Case sensitivity**: in XML tutto il markup è case-sensitive (il maiuscolo è diverso dal minuscolo). È quindi necessario usare le maiuscole per ELEMENT, ATTLIST, ecc., e l’elemento <para> è diverso dall’elemento <PARA>.
- **Valori tra virgolette**: tutti i valori di tutti gli attributi debbono avere le virgolette (semplici o doppie, ma in maniera coerente), anche se numeri o appartenenti ad una lista di valori predefiniti.

Altre caratteristiche di XML (2)

- **Tag omissibili**: Non esiste il concetto di tag omissibili.
- **Entità predefinite**: sono pre-definite e non ridefinibili
  5 entità:

```
<!ENTITY lt     “&#60;”>
<!ENTITY gt     “&#62;”>
<!ENTITY amp    “&#38;”>
<!ENTITY apos   “’”>
<!ENTITY quot   ‘””>
```

- **Processing instructions**: la sequenza di chiusura di un’istruzione di elaborazione è ‘?>’:

```
<?Fine-pagina?>
```


Il white space

XML adotta convenzioni molto semplici e dirette per il white space:

- ***New line***: Per semplicità ed uniformità, XML trasforma ogni tipo di new line (CRLF, LF e CR) nel solo carattere LF.
- "***If it ain't markup, it's data***": Ogni white space che appare nel contenuto del documento è rilevante, e deve essere passato intatto all'applicazione.
- Tuttavia, un ***parser validante*** è tenuto a precisare all'applicazione quale white space è stato riscontrato in elementi con content model di tipo elemento, cosicché l'applicazione possa decidere cosa farne.

Attributi per white space e lingua

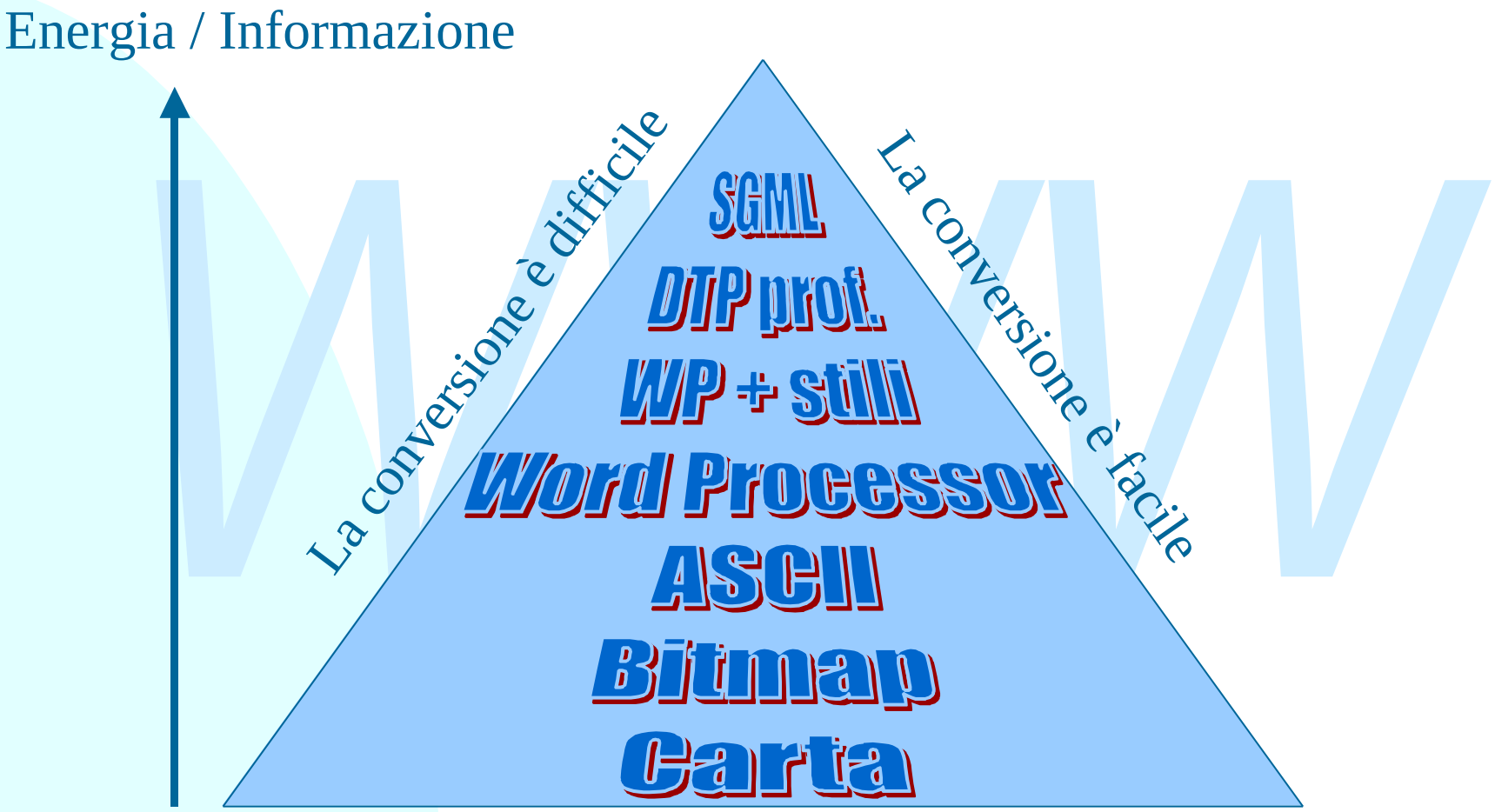
Esistono in XML due attributi riservati (ma da definire se usati):

- **xml:space** (valori possibili: “default” o “preserve”) permette all’autore di indicare all’applicazione se è opportuno che mantenga il white space
- **xml:lang** (valori possibili: i codici a due lettere di RFC 1766): permette all’applicazione di identificare la lingua in cui è scritto il contenuto di un elemento, per attivare funzionalità dipendenti dalla lingua:
  - Rendering corretto
  - Spell-checking
  - Full-text indexing
  - Editing

Analisi dei documenti

- Classificazione dei componenti
- Selezione dei componenti, costruzione della gerarchia, dei blocchi informativi e degli elementi di dati
- Identificazione delle connessioni
- Verifica e miglioramento iterativo delle specifiche

Perché convengono i linguaggi di markup?



Classificazione dei componenti

- La parte più importante del lavoro di progettazione di un'applicazione SGML è l'identificazione delle strutture e del significato delle parti dei documenti e delle loro relazioni.
- L'identificazione semantica dei componenti avviene quando si evidenzia l'esigenza di distinguere tra un tipo di dati ed un altro.
- Se due pezzi diversi di un documento contengono lo stesso tipo di informazione, li si deve considerare appartenenti alla stesso componente semantico, anche se sono separati tra loro.
- Viceversa, se due pezzi contengono due tipi diversi di informazione, o è necessario distinguerli in qualche maniera per i fini dell'applicazione, allora debbono essere distinti in due componenti semantici separati.
- Possiamo distinguere tra tre tipi di classificazione dei componenti di un documento: ***contenuto, struttura*** e ***presentazione***.

Classificazione basata sul contenuto

Si identificano i componenti per il significato che essi hanno, indipendentemente dalla loro posizione nel documento o dal loro aspetto grafico.

Ad esempio:

- indirizzi, città, codici postali;
- ricette, ingredienti, tempi di preparazione;
- termini, sviluppo grammaticale, significato.

Classificazione basata sulla struttura

Si identificano i componenti per il loro ruolo all'interno del documento, per il senso che hanno in quella posizione e in quella forma

Ad esempio:

- sezioni, capitoli, liste, paragrafi, titoli

Classificazione basata sulla presentazione

Si identificano i componenti per le variazioni nel modo in cui debbono apparire graficamente, senza implicazioni sul loro “vero significato”.

Ad esempio:

- Frasi con un determinato font o dimensione
- Blocchi da mantenere sulla stessa pagina
- Posti dove spezzare una pagina

Caratteristiche delle classificazioni

Questi tre modi di classificare i componenti di un documento sono presenti contemporaneamente bell'analisi di un documento. Persone diverse, sugli stessi documenti, possono identificare questa o quella classe a seconda di professione, ***forma mentis***, esigenze.

I tre tipi di classificazione hanno anche caratteristiche diverse di ***identificabilità, flessibilità*** e ***durata***.

Classificazione basata sul contenuto

È la classificazione più complessa da realizzare, ma la più flessibile, identificabile, flessibile e duratura.

Poiché identifico i componenti basati sul loro significato, è immediato identificare il senso di un componente.

La classificazione è indipendente dalla struttura del documento e dall'aspetto grafico, così posso cambiare idea su queste decisioni in qualunque momento, e anche fornire soluzioni diverse sugli stessi componenti.

Poiché un componente avrà sempre quel significato in qualunque contesto, anche cambiamenti di stile, organizzazione del documento ecc. non impediranno a questa classificazione di sopravvivere.

Classificazione basata sulla struttura

La classificazione strutturale identifica l'organizzazione di un documento in maniera sufficientemente generale, ma rozza per quel che riguarda il senso del documento.

È possibile in qualunque momento modificare la resa grafica degli elementi, ma non il loro ruolo nella struttura globale del documento.

Cambiamenti stilistici non preoccupano (il font, la larghezza di un paragrafo, l'esistenza o meno di un bordo in una tabella), cambiamenti strutturali importanti invece sì (ad esempio, passare da una forma a lista ad una a tabella, ecc.).

Classificazione basata sulla presentazione

In generale, ci sono più classi di contenuto che classi di presentazione. Per uniformità grafica e facilità di lettura molti componenti aventi significato diverso vengono resi graficamente nello stesso modo.

L'identificazione delle sole classi grafiche fa sparire immediatamente l'identificabilità di elementi di significato diverso ma resa grafica uguale.

Decisione successive di cambiamenti grafici di solo alcuni componenti, e non altri, saranno impossibili.

Utilizzi del documento per scopi diversi dalla presentazione (ad esempio, la creazione di un indice, l'inserimento in un motore di ricerca, ecc.), saranno impossibili.

Regole guida per la classificazione

Identificare il più possibile i componenti per il loro significato e contenuto. Richiede più lavoro ma ne vale la pena. È necessario, ovviamente, fermarsi ad un livello ragionevole di specificità.

Attribuire a questi componenti significati strutturali (tabelle, liste, paragrafi, organizzazioni gerarchiche tipo sezioni, sotto-sezioni, ecc.).

Specificare la resa grafica dei componenti. Quest'ultima specificazione tipicamente avviene al di fuori del contesto di SGML, con appositi strumenti e linguaggi di stylesheet.

Altri suggerimenti (1)

Scartare elementi puramente presentazionali: numero di pagina, elementi che si ripetono pagina dopo pagina (un logo, una decorazione, il nome di un capitolo) possono tipicamente essere aggiunti automaticamente dal formattatore e non è necessario considerarli come componenti del documento.

Identificare classi generali di informazioni. Anche se presenti in varie parti del documento, alcune informazioni possono avere lo stesso significato e lo stesso ruolo, e quindi debbono essere identificati nella stessa maniera.

Altri suggerimenti (2)

Identificare informazioni ripetute in varie parti del documento. Alcune informazioni (nomi propri o di organizzazioni, riferimenti ad immagini, date importanti, elementi ripetuti di una struttura, ecc.) debbono essere presenti in varie parti del testo in maniera identica, e debbono cambiare in maniera coerente. È utile avere un componente unico che registri una sola volta l'informazione da stampare, e venga usato ovunque necessario.

Identificare i componenti che provengono da sistemi informativi esistenti. Tipicamente un database ha già distinzioni di elementi basate sul contenuto. Se alcune informazioni provengono da un database è comodo e si risparmia tempo usare o basarsi sulla strutturazione dei dati già esistenti nel sistema informativo.

Tre grandi dubbi

Attributi o elementi?

- Nei documenti di testo, la distinzione è spesso chiara: elementi per contenuto, attributi per meta-informazioni. Nell'interscambio di dati è meno chiara: la destinazione di un URL è informazione o metainformazione?

Vincoli stretti o laschi?

- La tentazione esiste sempre di imporre tutto quello che si può imporre. Tuttavia in XML due possono essere gli obiettivi:
  - Identificare semanticamente gli elementi (tutti i titoli si chiamano TITOLO)
  - Uniformare la struttura (tutti gli INDIRIZZO hanno una VIA, un CAP, una CITTA, una PROV). Più è restrittiva la regola, più uniforme è il risultato.
- Qual è il vero obiettivo? Descrivere o regolare?

Content model misti: sì o no?

- I content model misti sono più difficili da trattare, descrivere, regolare. Tuttavia rappresentano una libertà di movimento che è importante lasciare agli autori.
- Un elemento fatto per essere letto (ad es. DESCRIZIONE) **deve** aver content model misto: enfasi, link ipertestuali, elementi semantici individuali ecc.

Conclusioni

Qui abbiamo parlato di XML, soprattutto per quanto non è derivato da SGML:

- Il senso di XML
- Usi innovativi di XML
- I criteri progettuali
- La distinzione tra documenti ben formati e validi
- Le principali differenze sintattiche
- Come analizzare documenti per trarne DTD

Riferimenti

Wilde's WWW, capitolo 7

Altri testi:

- Neil Bradley, ***The XML companion***, Addison Wesley 1998
- J. Bosak, ***XML, Java, and the future of the Web***, `http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm`
- T. Bray, J. Paoli, C.M. Sperberg-McQueen, ***Extensible Markup Language (XML) 1.0***, W3C Recommendation, 10 February 1998, **`http://www.w3.org/TR/REC-xml`**
- T. Bray, The annotated XML Specification,1998, **`http://www.xml.com/axml/testaxml.htm`**
- XMLNews Specifications, **`http://www.xmlnews.org/`**