Giuseppe Gigliozzi

Introduzione all'uso del computer negli studi letterari

A cura di Fabio Ciotti

Bruno Mondadori

accentata maiuscola acuta il 144. Della *e accentata maiuscola grave* non si hanno notizie. Il fenomeno si spiega facilmente con il fatto che queste lettere sono state aggiunte agli originari 128 caratteri, ma ciò non toglie che questa situazione provochi quelle spiacevoli turbative a un corretto e naturale ordinamento alfabetico che tutti possiamo costatare consultando un elenco del telefono, dove, appunto, certi nomi schizzano nei posti più impensati.

La famiglia ISO 8859

Attualmente la maggior parte delle macchine e dei sistemi operativi si basano su *code set* a 8 bit e lo standard più diffuso è la famiglia ISO 8859 nella quale troviamo la tavola 8859-1, meglio nota come ISO Latin 1, che contiene i caratteri necessari per scrivere in tutte le lingue neolatine e anglogermaniche. Per inciso si può ricordare che l'ISO Latin 1 è utilizzata dai sistemi operativi Windows e da varie piattaforme Unix, mentre del tutto peculiari sono state le scelte a lungo operate nei sistemi operativi Macintosh. Questo ha complicato non poco la vita all'utente che è costretto a tenere conto delle caratteristiche dei vari sistemi operativi nel momento in cui – poniamo – vuole trasferire un testo scritto con un *editor* in ambiente Windows su un computer Macintosh.

Unicode e ISO 10646

Negli anni novanta sono state avviate due iniziative parallele per sistemare in modo definitivo (si spera) il problema della babele informatica: la prima, gestita dal consorzio Unicode (un'organizzazione non profit in cui convergono numerosi produttori di sistemi informatici) e la seconda dall'ISO. Da queste iniziative sono nati Unicode e ISO 10646/UCS, due *coded character set* che sono per fortuna perfettamente allineati (le differenze riguardano solo aspetti tecnici) e che, con qualche ragione, hanno la presunzione di definirsi "universali". In effetti si tratta di sistemi di codifica dei caratteri che possono basarsi su 16 o persino 32 bit (in questo caso è possibile rappresentare potenzialmente oltre quattro miliardi di caratteri distinti!).

In realtà la questione di cosa si intenda per carattere in Unicode e ISO 10646 e di come essi vengano effettivamente rappresentati nella memoria del computer è piuttosto complessa (per chi è interessato, e paziente, rimandiamo a K. Whistler e M. Davies[1]); qui ci limiteremo a dire che, attualmente, sono stati codificati oltre 95 000 caratteri, tra cui si annoverano tutte le lingue occidentali, arabe e africane, e buona parte di quelle orientali.[2]

[1] K. Whistler, M. Davies, *Character Encoding Model*, Unicode Technical Report, 17, rev. 3.2, Unicode Consortium 2000.

[2] Un elenco completo è consultabile sul sito Web del consorzio Unicode all'indirizzo: <http://www.unicode.org/unicode/onlinedat/languages-scripts.html>

Anche se Unicode non viene ancora utilizzato da tutti i sistemi operativi e software esistenti (solo le versioni più recenti di Windows, per esempio, lo implementano) è molto probabile che nel futuro prossimo la sua adozione sarà sempre più vasta.

BIBLIOGRAFIA CONSIGLIATA

Gaylord H.E., *Character Representation*, in Ide N., Véronis J. (a c. di), *The Text Encoding Initiative: Background and Context*, Kluwer Academic Publishers, Dordrecht 1995, pp. 51-73.

Whistler K., Davies M., *Character Encoding Model*, Unicode Technical Report, 17, rev 3.2, Unicode Consortium, 2000. <http://www.unicode.org/unicode/reports/tr17>

3.1.2 La codifica del testo e i linguaggi di "mark-up"

I coded *character set* (qualunque essi siano) che ci permettono di compiere quella che possiamo definire l'"operazione nucleare" per l'utilizzazione dell'informatica nel settore umanistico (ma ovviamente anche in tutti gli altri campi), e cioè la scrittura, rappresentano l'elemento minimo, ma non sufficiente per la costruzione di quel "modello del testo" che vogliamo creare e al quale rivolgeremo le nostre domande per aver risposte sul testo stesso.

I limiti della codifica dei caratteri

L'insufficienza nasce da due ordini di problemi. Il primo concerne la natura stessa delle tabelle di codici fin qui realizzate: se è vero, infatti, che sono stati realizzati un buon numero di *code set* dedicati agli alfabeti nazionali, è anche vero che l'unico gruppo di caratteri presente in tutte le tavole è quello corrispondente all'alfabeto latino nell'ortografia dell'inglese. Questo provoca la non completa sovrapponibilità di tutti i codici, rendendo problematica la reale trasportabilità dei documenti, almeno fino al momento in cui Unicode non godrà di una diffusione universale.

Il secondo riguarda l'esigenza di realizzare un testo realmente leggibile dalla macchina, da tutte le macchine, e che conservi la maggior quantità possibile di informazione. La digitalizzazione di un testo non può essere affidata esclusivamente al sistema di codifica dei caratteri, anche se ci si ferma a un primissimo livello di trascrizione. Infatti, se la nostra riflessione sul testo si fa più attenta le cose si complicano (*vedi* 2.1 "La parola testo").

La natura complessa del testo

Come abbiamo visto, il libro che abbiamo sul tavolo ci appare subito duplice: da un lato il significato, dall'altro "l'oggetto-libro". Se poi lo apriamo non ci mettiamo molto a scoprire che lì dentro di libro ce n'è (come minimo) un altro. Prendiamo il caso

della prosa. Le righe scorrono sulla pagina finendo tutte alla stessa colonna e solo alcune appaiono leggermente diverse. Alcune cominciano un po' rientrate rispetto al margine sinistro, mentre altre finiscono prima del margine destro e sono per lo più chiuse da un punto o dai due punti. Il lettore avrà capito che abbiamo descritto la struttura classica di un testo in prosa: l'organizzazione in paragrafi.

Possiamo dire che in un testo a stampa troviamo due autori: l'autore del significato del testo e l'autore di un "oggetto-libro". Quando memorizziamo e codifichiamo, memorizziamo e codifichiamo quel preciso libro che abbiamo in mano. Quando ci poniamo alla costruzione del "modello del testo" (che, ricordiamo, è una struttura munita di un punto di vista) dobbiamo decidere e "dichiarare" quale dei tanti testi stiamo analizzando e le procedure che intendiamo seguire.

Esiste un punto in cui entrambi gli "autori" del testo concordano nell'attribuire senso a un segnale grafico che la pagina manda al lettore. È quello che la maestra chiamava "punto e a capo". "Hai finito un discorso e allora fai punto e a capo." In quel punto la volontà dell'autore dell'aspetto semantico del testo (che ha individuato la chiusura di un'unità coerente del proprio argomentare) e la scelta dell'editore (che a quel punto decide che la riga non rispetta la giustificazione, quanto spazio deve esserci tra un paragrafo e l'altro e così via) si sovrappongono. Tocca a noi scegliere quale delle due intenzioni rappresentare. Potremmo infatti dire "quest'ultima riga è più corta" o "qui finisce un paragrafo", ma per dirlo (il che non significa che coincida con il *carriage return* o il *line feed* con cui generalmente si chiude una linea in un file di testo) dobbiamo costruirci uno strumento adatto.

Word processor e Editor

Le possibilità in questo settore assomigliano a quelle presenti nel campo dei *word processor* e degli *editor*. Da un lato troviamo programmi che sfruttano le sempre maggiori possibilità grafiche degli elaboratori per tentare di mostrare sul video esattamente ciò che apparirà poi sulla carta stampata. Sono programmi che si rifanno a una filosofia che viene definita *wysiwyg*, e che significa: *What You See Is What You Get* ("Quello che vedi è quello che ottieni"). Dal lato opposto abbiamo programmi che si disinteressano dell'aspetto che il documento assume sul video per concentrarsi sul risultato finale. I file destinati a questo tipo di programmi sono composti dal testo che si vuole comporre (ovviamente) e da una serie di istruzioni che indicano al *device* di output (normalmente la stampante) che cosa fare.

Questi programmi vengono definiti *batch* in quanto (come facevano i primi elaboratori) dividono l'operazione in due fasi: 1) la compilazione del programma (il testo da questo punto di vista è un programma composto da istruzioni e argomenti); 2) l'esecuzione del programma (e cioè la realizzazione dell'output).

I linguaggi di mark-up

Come in tutte le cose della vita la ragione non sta mai da una sola parte e così i programmi *wysiwyg*, mentre facilitano fortemente il lavoro dell'utente proponendosi come mezzo rapido e "amichevole" per un operatore non necessariamente espertissimo, legano l'elaborazione del testo a un determinato programma e a un determinato sistema rendendone spesso problematica la portabilità. Allo stesso tempo i programmi *batch* rendono più faticosa la vita a chi elabora il testo, ma, utilizzando generalmente file scritti con un normale *editor*, rendono più agevole la trasportabilità dei prodotti.

La linea vincente – com'è ovvio per quanto siamo venuti dicendo finora – è la seconda, in quanto ci permette di identificare un vero e proprio linguaggio di *mark-up* contenente tutti i simboli che ci servono per rappresentare il nostro testo.

Questi linguaggi, che in italiano vengono definiti "linguaggi di marcatura o di codifica del testo", hanno i loro lontani parenti in quei "segnacci" che i professori facevano sui nostri compiti quando li correggevano. Segno blu: errore grave; segno rosso: poteva andar peggio. Punto di riferimento più vicino sono quei segni che un qualunque autore fa sul dattiloscritto o sulle bozze per mandare informazioni relative alla stampa al tipografo. Una parola sottolineata indica una parola da stampare in corsivo, una X indica una cancellatura e così via.

Questi sistemi di marcatura sono veri e propri linguaggi in quanto sono composti da una serie di *tag* (che potremmo tradurre "marche" e che costituiscono le "parole" del linguaggio) e da una sintassi che regola i rapporti tra loro.

Ovviamente i primi "linguaggi di marcatura" ad essere ideati sono stati quei linguaggi che si occupavano dell'aspetto tipografico del testo. Era evidente che – se non si accettava l'ottica del *wysiwyg*, ma si riteneva più proficuo lavorare su documenti-bozza che dovevano poi passare attraverso un programma che producesse l'output definitivo in modalità *batch* (ecco i *batch formatter*) – bisognava individuare una serie di istruzioni che consentissero alla periferica di generare il documento nella forma voluta.

Si poteva così immaginare un *tag* che dicesse alla macchina che il testo da lui marcato (e fino al *tag* di chiusura) doveva essere stampato in corpo 14, in neretto, centrato e che dopo il *tag* di chiusura bisognava lasciare tre righe bianche. Il lettore avrà riconosciuto che stiamo descrivendo quella parte della pagina che riconosciamo come un titolo.

Linguaggi di mark-up procedurali

Questi linguaggi si concentrano su quell'aspetto del testo che abbiamo affidato alla responsabilità di un secondo autore che affianca l'autore dell'aspetto semantico e che, potremmo dire sin-

tetizzando, sono orientati sul *documento* più che sul *testo*. Questi linguaggi vengono definiti *linguaggi procedurali* perché la loro esecuzione attiva delle procedure tese a ottenere un determinato risultato.

Normalmente i *tag* di questi linguaggi sono contraddistinti da un segnale che li precede e che li differenzia dal testo normale. Per esempio, se scrivo la parola "corsivo" in un testo il programma "intenderà" che io sto scrivendo un pezzo sui vari tipi di font disponibili; ma se io scrivessi *\corsivo* e se il programma sapesse che il *backslash* (e cioè la barra) contraddistingue i comandi, potrebbe non scrivere la parola "corsivo" nel testo, ma cominciare a scrivere "in corsivo" le parole successive.

Quello che segue potrebbe essere un esempio di file preparato per essere trattato da un interprete di linguaggio procedurale.

```
\nopagenumbers
\hsize = 32pc
\vsize = 44pc

\parskip = 1pc
\parindent = 0pt
\baselineskip = 12pt

\font\fone = amr10 scaled\magstephalf
\font\ftwo = amr7 scaled\magstep0
\font\fthree = amr10 scaled\magstep1

\def\tm{\raise 1.15ex\hbox{\ftwo TM}}
\def\MT{Micro\kern-.15em\TeX}

\centerline{\fthree Questo è il titolo dell'esempio}
\vskip 1pc
\centerline{Realizzato da Giuseppe Gigliozzi}
\vskip 1.5pc
```

Qui potremmo scrivere qualche riga di testo:

```
\vskip 2pc

\def\asi{\hbox to 24pt{\hfil$\bigtriangleup$\hfil}\hfil}
\def\logo{\leaders\hbox{\asi}\hfil}

\def\bsi{\hboxto24pt{\hfil$\bigtriangledown\bigtriangledown$\hfil}\hfil}
\def\blog{\leaders\hbox{\bsi}\hfil}
\centerline{$\bigtriangleup$}
```



```
\vskip -4pt
\centerline{$\bigtriangledown\bigtriangledown$}

\vskip 2pc
\centerline{Esempio per Letteratura & computer}
\centerline{Bruno Mondadori, 1997}
\centerline{Milano}
\bye
```

Linguaggi di *mark-up* dichiarativi

Diverso sarebbe stato se, nell'esempio, invece di "\centerline {\fthree Questo è il titolo dell'esempio}" avessi scritto: "*\Titolo*" oppure, al posto dell'ipotetico "*\corsivo*", "*\Emphasis*". Nel primo caso il mio interesse si sarebbe evidentemente spostato dall'aspetto grafico della riga del titolo all'aspetto sostanziale: *quella riga è un titolo* e cioè una parte di testo che gode di uno statuto diverso rispetto alle altre e che svolge una sua particolare funzione all'interno del testo. Nel secondo caso, piuttosto che sul fatto che una certa parola è resa in corsivo, avrei concentrato la mia attenzione sulla volontà dell'autore di dare una particolare enfasi a quella parola. Con una conseguenza: se mi fossi attenuto a una *codifica procedurale* (che punta al documento), nel momento in cui volessi fare delle ricerche utilizzando le codifiche inserite potrei chiedere a un qualunque programma di mostrarmi tutte le parole in corsivo, ma non potrei mai chiedergli di distinguere le parole enfatizzate da quelle rese in corsivo perché in lingua straniera. Sono entrambe in corsivo e il linguaggio procedurale si ferma a questo aspetto, mentre un *linguaggio dichiarativo*, che si preoccupa di dichiarare che un dato segmento testuale incarna in quel punto un elemento della struttura del testo, non potrebbe accontentarsi di questo e ci proporrebbe un *tag* apposito per le parole in lingua diversa da quella principale del testo: "*\foreign*" (potrebbe essere uno).

Riassumendo: i linguaggi procedurali – concentrati sul documento – offrono all'utente una serie di marcatori che gli permettono di segnalare aspetti soprattutto tipografici del documento e indicano alle opportune periferiche quale procedura attivare al momento dovuto (per questo sono spesso legati a certe periferiche con problemi di "portabilità"); i linguaggi dichiarativi – concentrati sul testo – forniscono una serie di marcatori che consentono di marcare elementi appartenenti alla struttura astratta del testo. Sono per questo indipendenti dai diversi *device*, mentre nulla esclude che vengano usati per ottenere output gradevoli di un documento. È possibile, cioè, marcare dichiarativamente anche aspetti grafici della pagina (salti-pagina, spazi bianchi, rientri ecc.).

Per intenderci: se si codifica esattamente l'aspetto grafico che deve avere un titolo (corpo 14, neretto, centrato, 12 punti tipografici dopo), quella riga, quel titolo si porterà sempre dietro quella sua rappresentazione grafica; se invece ci limitassimo a dire che quello è un titolo nulla ci vieterebbe di istruire il nostro programma di visualizzazione o di stampa al fine di renderlo in molti modi differenti.

Su questa via si aprono enormi possibilità a questi "linguaggi di marcatura". Se il linguaggio scelto è abbastanza potente è possibile rappresentare il testo (i suoi elementi e i rapporti che intercorrono tra loro) a vari livelli. Si possono costruire vari modelli del testo: accanto alla struttura editoriale possiamo immaginare di codificare la sua struttura grammaticale (per esempio i tempi verbali), la struttura retorica (le figure, ma anche la distribuzione del discorso diretto e indiretto), i luoghi; e poi è solo questione di voglia e fantasia.

E da ultimo (ma non per questo di minore importanza), la possibilità di utilizzare un testo trattato con un linguaggio di *codifica dichiarativa* per qualunque tipo di trattamento informatico. Come abbiamo visto, questi linguaggi non sono dedicati a un'applicazione in particolare e quindi, se siamo stati capaci di "dichiarare tutto quello che serve", lo stesso testo può essere analizzato dal punto di vista morfologico, lessicografico, narratologico. Potremmo realizzare le tradizionali "concordanze", ma potremmo anche analizzare la presenza dei vari personaggi nel racconto.

BIBLIOGRAFIA CONSIGLIATA

Buzzetti D., *Il testo "fluido". Sull'uso dell'informatica nella critica e nell'analisi testuale*, in Floridi L. (a c. di), *Filosofia e informatica*, Paravia, Torino 1996, pp. 85-93.

Id., *Archiviazione digitale dei dati e adeguatezza della rappresentazione del testo*, in "Schede Umanistiche", n.s., IX, 2, 1999, pp. 245-254.

Id., *Rappresentazione digitale e modello del testo*, in Aa.Vv., *Il ruolo del modello nella scienza e nel sapere*, Accademia Nazionale dei Lincei, Roma 1999, pp. 127-161.

Ciotti F., *Cosa è la codifica informatica dei testi?*, in Gruber D., Paoletto P. (a c. di), *Umanesimo e informatica*, Metauro, Pesaro 1997, pp. 55-85.

Id., *Testo rappresentazione e computer. Contributi per una teoria della codifica testuale*, in Nerozzi Bellman P. (a c. di), *Internet e le Muse*, Mimesis, Milano 1997, pp. 213-230.

Id., *Text Encoding as a Theoretic Language for Literary Text Analysis*, in Fiormonte D., Usher J. (a c. di), *New Media and Humanities: Research and Applications*, Oxford University Humanities Computing, Oxford 2001, pp. 39-47.

Id., *Manuale XML per le scienze umane. La forma del testo elettronico*, Laterza, Roma-Bari, in corso di stampa.

Coombs J.H., Renear A. e Derose S.J., *Markup and the Future of the Scholarly Text Processing*, in "Communications of the Association for Computing Machinery", XXX, 11, 1987, pp. 933-947.

McGann J., *Radiant Textuality. Literature after the World Wide Web*, Palgrave, New York 2001.

Renear A., *Out of Praxis: Three (Meta)Theories of Textuality*, in Sutherland K., *Electronic Text. Investigation in Method and Theory*, Clarendon Press, Oxford 1997, pp. 107-126.

3.1.3 SGML e XML

Nella numerosa famiglia dei linguaggi di *mark-up*, un ramo si è dimostrato particolarmente efficace per il trattamento di dati in ambito umanistico. Ci riferiamo ai linguaggi basati sullo *Standard Generalized Mark-Up Language* (SGML), la cui sintassi e logica di funzionamento è stata ereditata quasi completamente dal più recente *Extensible Mark-Up Language* (XML). Le differenze tra questi due linguaggi (ma sarebbe più corretto definirli metalinguaggi) sono assai poche, tutte di natura tecnica, e soprattutto hanno una rilevanza minima dal nostro punto di vista di utenti consapevoli e avanzati, ma non necessariamente di specialisti informatici. Poiché, per una serie di motivi tecnici ma anche commerciali, XML ha avuto una rapidissima diffusione e ha sostituito il suo progenitore in moltissimi contesti applicativi, nel prosieguo faremo riferimento ad esso, anche se quasi tutto ciò che diremo vale anche per SGML.

Extensible Mark-Up Language (XML)

L'*Extensible Mark-Up Language* si basa su un sistema di *mark-up* dichiarativo che punta a descrivere la struttura astratta di un documento piuttosto che il suo aspetto grafico, e a questo scopo consente la dichiarazione dell'insieme di elementi che lo compongono e delle loro relazioni. Paradossalmente potremmo dire che XML è un linguaggio "vuoto", riempito solo delle regole che permettono la sua stessa progettazione. XML, quindi, è un metalinguaggio che ci fornisce le regole per realizzare un numero indefinito di linguaggi di codifica. Ognuno di questi linguaggi, potremmo dire, corrisponde a un particolare modello di testo (o insieme di testi).

Se tutto può essere progettato *ex novo* e se lo scopo rimane sempre quello della realizzazione di un documento in MRF che garantisca la massima portabilità, è evidente che si deve garantire la condivisione del codice con cui si è codificato il testo da parte di tutti i possibili utenti. Questo scopo viene raggiunto dichiarando

"esplicitamente" tutti gli elementi che possono essere presenti in un documento (dai caratteri alfabetici ai marcatori utilizzati per la codifica) e le regole che governano le loro combinazioni.

La gestione dei caratteri in XML

Per definizione XML adotta come codice per la codifica dei caratteri lo standard Unicode. In linea teorica potremmo quindi ritenere che il problema della portabilità, almeno a questo livello "atomico", sia risolto. Tuttavia, come abbiamo notato, l'adozione di Unicode è ancora piuttosto limitata. Dunque, per evitare i rischi di perdita di informazioni, XML mette a disposizione una tecnica (ereditata da SGML) per rappresentare caratteri esterni al repertorio del vecchio codice ASCII (che ricordiamo, è l'unico *code set* che ragionevolmente possiamo considerare leggibile da ogni sistema informatico attualmente in uso). Questa tecnica si basa sull'uso di sequenze di caratteri noti, dette entità (*entity*), che "stanno per" altri caratteri ignoti al sistema.

Un esempio ci aiuterà a capire meglio questa tecnica. Poniamo di voler rappresentare nel nostro documento l'ormai famosa *e maiuscola accentata grave* (È). Chiaramente non la troviamo tra i 128 caratteri del codice ASCII e sappiamo che sono proprio quei 128 caratteri ad essere quelli maggiormente "capiti" dai computer di tutto il mondo. A questo punto la sintassi che regola la costruzione delle *entity* in XML ci fa sapere che si può descrivere qualunque carattere a patto di rispettare una precisa sintassi: la *e commerciale* (&) è il segnale di inizio codifica; il *punto e virgola* (;) quello di fine codifica. Tra i due segnali è possibile inserire il carattere da codificare (nel nostro caso *E*), assieme a una descrizione stabilita della codifica (nel nostro caso *grave*). Tutto questo ci porta a rappresentare il segno grafico *È* con il gruppo: &*Egrave*;. Ovviamente l'elenco delle *entity* legali e riconosciute sarà contenuto in un'apposita tabella che dovrà essere esplicitamente dichiarata.

La struttura del documento e la *Document Type Definition*

Siamo partiti dal particolare – dal piccolissimo, dalle lettere – per poi arrampicarci più agevolmente nella complessità della codifica di un intero testo.

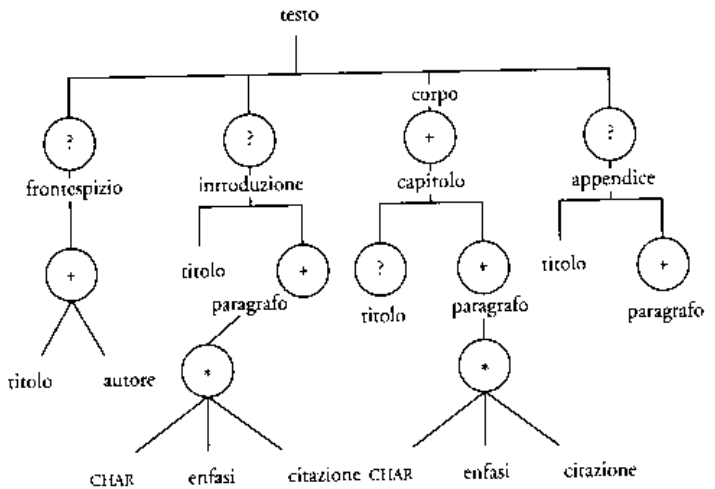
La struttura astratta di un testo (quella di cui si occupa programmaticamente XML) viene descritta dichiarando in una speciale tabella, detta DTD (*Document Type Definition*), gli elementi che la costituiscono e le relazioni che intercorrono tra questi elementi. Il linguaggio non indica alcuna prescrizione riguardo alla tipologia e al numero degli elementi, ma solo le regole sintattiche che regolano la loro definizione.

Nella DTD vengono dichiarati: il nome e il tipo degli *elementi* presenti nel testo (possono essere titolo, paragrafo, nota, pagina ecc.); la descrizione formale del *contenuto* di ciascun elemento, che può essere costituito da sottoelementi o sequenze di testo, e

l'ordine con il quale sottoelementi e sequenze di testo possono comparire (un capitolo potrà contenere un titolo e una sequenza di paragrafi; un paragrafo potrà contenere sequenze di testo semplice, parole evidenziate, o citazioni ecc.); gli eventuali *attributi* da assegnare a ogni elemento (un titolo potrà essere caratterizzato dalla propria resa grafica, una nota dalla sua posizione nella pagina); l'elenco delle eventuali *entità* (abbiamo già visto sopra la storia della "e maiuscola accentata grave").

Ovviamente ogni elemento (un titolo, per esempio) può comparire solo in un punto preciso della struttura del documento (mettiamo all'inizio di un capitolo) e non in altri (non all'interno di un capitolo). XML richiede, quindi, la descrizione della struttura del documento e questa struttura deve essere rigorosamente gerarchica. Ne consegue che ogni documento XML dovrà avere un elemento iniziale o *elemento radice*.

Di nuovo, un esempio ci aiuterà più di un discorso astratto. In un testo monografico in prosa possiamo trovare il frontespizio, una eventuale introduzione, la sezione del corpo testuale vero e proprio, che contiene una sequenza di capitoli, e una eventuale appendice; ogni capitolo potrebbe avere un suo titolo, seguito da una sequenza di uno o più capoversi (che in inglese si chiamano *paragraph*); questi a loro volta conterranno sequenze di caratteri, inframmezzate a citazioni o enfasi... Può bastare perché è già abbastanza complicato.



La forma di un documento XML.

La DTD, insomma, definisce le strutture e le regole da utilizzare per codificare il documento, ne costituisce la grammatica. Ogni

documento codificato, detto *Document Instance* deve contenere un riferimento alla sua DTD e deve rispettarne i vincoli (in realtà XML, a differenza di SGML, prevede il caso di un documento che sia sintatticamente "ben formato" ma non strettamente "valido" rispetto a una DTD. Ma per questi aspetti rimandiamo alla letteratura specialistica).

Il nostro file di testo comincerà, quindi, dichiarando che tipo di documento contiene e a quale DTD fa riferimento. Questo verrà fatto in un'apposita zona del documento che si chiama *Doctype Declaration*. Senza questa dichiarazione i programmi (di lettura, di formattazione o di analisi) non saprebbero quale tipo di *tag* e quali regole sintattiche vigono nel documento e non potrebbero "leggerlo". Un esempio di *Doctype Declaration* è il seguente:

```
<!DOCTYPE testo SYSTEM "/sgml/dtd/testo.dtd">
```

Dove: "<!" è il segno di apertura della *Doctype Declaration*; "DOCTYPE" è la parola chiave che indica il tipo di dichiarazione; "testo" è il nome della DTD; "SYSTEM" è la parola chiave che indica che la DTD è esterna al file (potremmo anche scriverla tutta nel file di testo, ma non è certo economico), ma si trova nello stesso computer ("sistema"); "/sgml/dtd/testo.dtd" è il *pathname* che permette di rintracciare il file che contiene la DTD ("testo.dtd"), ">" è il segno che chiude la dichiarazione.

La struttura logica prevista dalla DTD viene rappresentata nel documento mediante un insieme di coppie annidate di *tag* ciascuna delle quali rappresenta un particolare elemento. Un esempio per chiarire: questo paragrafo si intitola "SGML e XML". Se facesse parte di un documento codificato in XML troveremmo scritto:

```
<titolo rend="bold"> SGML e XML</titolo>
```

Dove "<titolo>" è il *tag di apertura* dell'elemento; "rend" è un attributo che contiene l'indicazione che il testo andrebbe reso in "neretto"; "SGML e XML" è la sequenza di testo codificata; e "</titolo>" è il *tag di chiusura* dell'elemento. Come si vede i segni "<" e "</" aprono rispettivamente il *tag* di apertura e il segno ">" li chiude entrambi. Per indicare il fatto che un titolo sta dentro un capitolo dovremo fare attenzione a inserire la coppia di *tag* dell'elemento <titolo> all'interno della coppia di *tag* dell'elemento <capitolo>. Per indicare il fatto che i paragrafi seguono il titolo dovremo inserire le coppie di *tag* dell'elemento <paragrafo> di seguito alla coppia di *tag* dell'elemento <titolo> ma sempre all'interno della coppia di *tag* dell'elemento <capitolo>:


```
<capitolo>
   <titolo rend="bold"> SGML e XML</titolo>
   <paragrafo> ....</ paragrafo>
   <paragrafo> ....</paragrafo>
   <paragrafo> ....</paragrafo>
</capitolo>
```

Non è nostro compito addentrarci oltre nei meandri della sintassi XML (a questo scopo sono stati realizzati dei manuali molto completi, per lo più in inglese, ma si sa che nel mondo dell'informatica l'inglese impera); nostro compito è ribadire che codificare un testo in XML non consiste esclusivamente nell'inserire i *tag* che lo descrivono, ma significa anche rispettare dei precisi vincoli sintattici espressi nella DTD.

BIBLIOGRAFIA CONSIGLIATA

Burnard L., *A Gentle Introduction to XML*, in Sperberg-McQueen C.M., Burnard L. (a c. di), *Guidelines for Electronic Text Encoding and Interchange (TEI P4), XML-Compatible Edition*, I, TEI Consortium, Chicago-Oxford 2001, pp. 13-34.

Ciotti F., *Manuale XML per le scienze umane. La forma del testo elettronico*, Laterza, Roma-Bari, in corso di stampa.

Goldfarb C.F., *A Generalized Approach to Document Markup*, in *Proceedings of the ACM SIGPLAN-SIGOA Symposium on Text Manipulation*, ACM, New York 1981, pp. 68-73.

Id., *The SGML Handbook*, Oxford University Press, Oxford 1990.

Goldfarb C.F., Prescod P., *The XML Handbook*, Prentice Hall, Upper Saddle River 1998.

Herwijnen E.V., *Practical SGML*, Kluwer Academic Publisher, Boston-Dodrecht-London 1994.

Ide N., Véronis J. (a c. di), *The Text Encoding Initiative: Background and Context*, Kluwer Academic Publishers, Dordrecht 1995.

Megginson D., *Structuring XML Documents*, Prentice Hall, Upper Saddle River 1998.

Rusty Harold E., Scott Means W., *XML. Guida di riferimento*, Apogeo-O'Reilly, Milano 2001.

World Wide Web Consortium (W3C), *Extensible Markup Language (XML) 1.0. W3C Recommendation*, 1998.
<http://www.w3.org/TR/1998/REC-xml-19980210>

XML Cover Pages, <http://www.oasis-open.org/cover>

3.1.4 La "Text Encoding Initiative"

Come abbiamo visto, XML lascia la massima libertà nel costruire la migliore grammatica possibile per descrivere un determinato documento: chiunque possieda le necessarie conoscenze tecniche può scrivere una propria DTD. Ma questa libertà è limitata dalle difficoltà che si incontrano nella realizzazione di una simile operazione. Per tanto sarà senz'altro più economico utilizzare una delle DTD preesistenti.

L'influenza di SGML, prima, e di XML, ora, nel settore umanistico è dovuta soprattutto al successo del vasto e complesso schema di codifica messo a punto dalla *Text Encoding Initiative* (TEI), un progetto internazionale divenuto ormai un punto di riferimento ineludibile per chi si occupi di trattamento informatico dei testi.

Una breve storia della TEI

Sin dagli anni ottanta in campo umanistico si era avvertita l'esigenza di rispondere adeguatamente ai problemi di interscambiabilità e portabilità creati della proliferazione dei linguaggi di codifica, e di definire uno standard per la rappresentazione di testi su supporto digitale. Nel 1988, con la sponsorizzazione delle tre maggiori associazioni professionali nel campo dell'informatica umanistica e della linguistica computazionale – la Association for Computers and the Humanities (ACH), la Association for Computational Linguistics (ACH) e la Association for Literary and Linguistic Computing (ALLC) –, fu avviato un progetto internazionale per sviluppare uno schema di codifica che mettesse ordine nel settore. Questo progetto, denominato appunto *Text Encoding Initiative*, ha visto la partecipazione di studiosi provenienti da tutto il mondo. Per definire tale schema di codifica, in prima istanza venne individuato come linguaggio di base lo *Standard Generalized Markup Language* (SGML).

A partire dal 1989 la TEI ha sviluppato diverse versioni del suo schema di codifica, le cui specifiche provvisorie sono state pubblicate per la prima volta nel 1991 con il titolo *Guidelines for Electronic Text Encoding and Interchange, TEI P1*. Il lavoro della TEI ha poi visto molte successive revisioni che hanno portato alla pubblicazione di altri due manuali: uno nel 1992, *TEI P2*, nel quale la struttura della DTD è stata notevolmente rivista; e uno nel 1994 intitolato *TEI P3*. Nel 2000 i membri della TEI hanno deciso di rendere permanente il progetto, costituendo il TEI Consortium, un'organizzazione internazionale senza scopo di lucro fondata allo scopo di sostenere e sviluppare lo schema di codifica TEI.

Il primo risultato del TEI Consortium è stata la pubblicazione nel giugno del 2002 di una nuova versione dello schema di codifica e delle relative *Guidelines for Electronic Text Encoding and Interchange*, battezzata *P4*. L'aspetto più importante di questa nuova versione è la sua piena conformità con XML.

Il linguaggio di codifica TEI e la sua DTD

Le *Guidelines* definiscono un linguaggio di codifica espresso nella sintassi XML e basato su una complessa DTD. I principi che hanno orientato lo sviluppo dello schema di codifica TEI si basano sui fondamenti teorici dei linguaggi di *mark-up* di tipo dichiarativo, come SGML e XML, che prediligono la descrizione di strutture logiche e astratte del documento piuttosto che del suo aspetto fisico. Questa predilezione per una codifica di tipo dichiarativo e strutturale è stata rispettata nella maggior parte dei casi. Tuttavia, lo schema prevede anche elementi di tipo presentazionale, utilizzabili quando la scelta del *mark-up* descrittivo non è praticabile senza arrecare problemi o quando le esigenze di ricerca richiedono una forte aderenza del testo elettronico al suo originale cartaceo.

In estrema sintesi possiamo dire che un testo codificato in linguaggio XML conformemente alle specifiche della TEI è composto di due parti: la TEI *header* (rappresentata dall'elemento <teiHeader>) e la trascrizione vera e propria del testo (codificata con l'elemento <text>).

La TEI *header*

La TEI *header* fornisce informazioni analoghe a quelle contenute nella pagina del titolo di un volume: una descrizione bibliografica relativa al testo elettronico, una descrizione delle modalità di codifica, una descrizione non bibliografica del testo e la storia delle revisioni che ha subìto il documento.

Di nuovo, un esempio sarà più chiaro di una descrizione astratta. Quella che segue è la TEI *header* realizzata per la versione elettronica dell'edizione del 1942 de *L'amata alla finestra* di Corrado Alvaro:

```
<teiheader>
<filedesc>
<titlestmt>
<title>L'amata alla finestra: edizione elettronica</title>
<author>Corrado Alvaro</author>
<respstmt>
<resp>Edizione elettronica curata da</resp><name>Giuseppe Gigliozzi</name>
</respstmt>
</titlestmt>
<publicationstmt>
<publisher>CRILet &mdash; Centro Ricerche Informatica e Letteratura</pu-
    blisher>
</publicationstmt>
<sourcedesc>
```

```
<bibl>
<author>Corrado Alvaro</author>
<title>L'amata alla finestra</title>
<pubplace>Milano</pubplace>
<publisher>Bompiani</publisher>
<dare>1942</dare>
</bibl>
</sourcedesc>
</filedesc>
</teiheader>
```

Lasciamo all'attento lettore il compito di ritrovare in questa intestazione la dichiarazione relativa al documento elettronico e la descrizione del "documento sorgente" (quello cioè da cui si è partiti).

La codifica del corpo del testo

Il testo, a sua volta, può avere una sezione di "avantesto", una di "appendice" (*front* e *back*) e un corpo (*body*) vero e proprio. Il che ci porta alla seguente struttura generale:

```
<TEI.2> [elemento radice]
   <teiHeader> [contiene le informazioni della Tei Header] </teiHeader>
   <text> [testo]
      <front> [i materiali che precedono il corpo del documento] </front>
      <body> [il corpo del testo] </body>
      <back> [i materiali che seguono il corpo del documento] </back>
   </text>
</Tei.2>
```

Il corpo del testo a sua volta sarà diviso in sezioni ed eventuali sottosezioni: per esempio un romanzo in parti e capitoli, un poema in canti e strofe, un'opera drammaturgica in atti e scene. Queste partizioni vanno codificate mediante la serie di elementi <div1>, <div2> ... <div6> a seconda del loro livello gerarchico. Ogni sezione sarà a sua volta divisa in blocchi di testo che a seconda del genere saranno paragrafi (<p>), gruppi di versi e versi (<lg> e <l>), battute (<sp>), i quali a loro volta potranno contenere altri sottoelementi. Basti pensare che gli elementi definiti nella DTD sono oltre cinquecento, e che per molte tipologie di caratteristiche strutturali di un testo sono fornite molteplici modalità diverse di codifica.

Inoltre, la DTD della TEI ha una struttura fortemente modulare che prevede ampie possibilità di personalizzazione e di estensione, al fine di adattarsi a ogni esigenza di codifica testuale.

UN ESEMPIO DI TESTO CODIFICATO IN TEI

A titolo di esempio si veda il seguente brano, estratto da *Il Turno* di Luigi Pirandello (Bompiani, Milano 1959). Trattandosi di un frammento non sono visibili la TEI *header* e gli elementi di alto livello.

```
<div1 type="cap">
   <head rend="center">1</head>
   <p><q>Giovane d'oro, sì sì, giovane d'oro, <name type="pers">Pepè Alletto</name>!</q> il < name type="pers">Ravì</name> si sarebbe guardato bene dal negarlo; ma, quanto a concedergli la mano di <name type="pers">Stellina</name>, no via: non voleva se ne parlasse neanche per ischerzo.</p>
   <p><q>- Ragioniamo!</q></p>
   <p>Gli sarebbe piaciuto maritar la figlia col consenso popolare, come diceva; e andava in giro per la città, fermando amici e conoscenti per averne un parere. Tutti però, sentendo il nome del marito che intendeva dare alla figliuola, strabiliavano, strasecolavano:</p>
   <p><q>- <name type="pers">Don Diego Alcozèr</name>?</q></p>
   <p>Il <name type="pers">Ravì</ name type="pers"> frenava a stento un moto di stizza, si provava a sorridere e ripeteva, protendendo le mani:</p>
   <p><q>- Aspettate... Ragioniamo!</q></p>
   <p>Ma che ragionare! Alcuni finanche gli domandavano se lo dicesse proprio sul serio:</p>
   <p><q>- <name type="pers">Don Diego Alcozèr</name>?</q></p>
   <p>E sbruffavano una risata.</p>
   <p>Da costoro il <name type="pers">Ravì</name> si allontanava indignato, dicendo:</p>
   <p><q>- Scusate tanto, credevo che foste persone ragionevoli.</q></p>
   <p>Perché lui, veramente, ci ragionava su quel partito, ci ragionava con la più profonda convinzione che fosse una fortuna per la figliuola. E s'era intestato di persuaderne anche gli altri, quelli almeno che gli permettevano di sfogare l'esasperazione crescente di giorno in giorno.</p>
   <p><q>- Avete voluto la libertà, santo Dio! il re che regna e non governa, la leva per tutti, un esercito formidabile, ponti e strade, ferrovie, telegrafo, illuminazione: cose belle, bellissime, che piacciono anche a me: ma si pagano, signori miei! E le conseguenze quali sono? Due, nel caso mio. Numero uno: ho lavorato come un <emph rend="it">arcibue</emph>, tutta la vita, onestamente per mia disgrazia e non son riuscito a mettere da parte tanto da poter per ora maritare la figlia secondo il suo piacere, che sarebbe anche il mio. Numero due: giovanotti, non ce n'è: intendo dire di quelli che a un padre previdente possano assicurare, sposando, il benessere della figliuola: prima che si facciano una posizione, Dio sa quel che ci vuole; quando se la son fatta, pretendono la dote e fanno bene; senza posizione, in coscienza, quale padre affiderebbe loro la figlia? Dunque? Dunque bisogna sposare un vecchio, vi dico, se il vecchio è ricco. Di
```

```
giovani poi, volendo, alla morte del vecchio, ce n'è quanti se ne
vuole.</q></p>
   [...]
   </div1>
```

BIBLIOGRAFIA CONSIGLIATA

Bernard D.T. *et al.*, *SGML-Based Markup for Literary Texts: Two Problems and Some Solutions*, in "Computer and the Humanities", 1988, pp. 265-276.

Barnard D.T., Fraser C.A. e Logan C.M., *Generalized Markup for Literary Texts*, in "Literary and Linguistic Computing. Journal of the Association for Literary and Linguistic Computing", 3, 1988, pp. 26-31.

Burnard L., *Text Encoding for Information Interchange. An Introduction to the Text Encoding Initiative*, Document TEI J31, Oxford University Computing Services, Oxford 1995. <http://www.tei-c.org/Papers/J31>

Ciotti F., *Manuale XML per le scienze umane. La forma del testo elettronico*, Laterza, Roma-Bari, in corso di stampa.

Ide N., Véronis J. (a c. di), *The Text Encoding Initiative: Background and Context*, Kluwer Academic Publishers, Dordrecht 1995.

Sperberg-McQueen C.M., *Text in the Electronic Age: Textual Study and Text Encoding, with Exemples from Medieval Texts*, in "Literary and Linguistic Computing", 6/1, 1991, pp. 34-46.

Sperberg-McQueen C.M., Burnard L. (a c. di), *Guidelines for Electronic Text Encoding and Interchange (TEI P4). XML-compatible edition*, TEI Consortium, Chicago e Oxford 2001. <http://www.tei-c.org/P4X/>

TEI Website, <http://www.tei-c.org>

3.2 Gli archivi e le biblioteche

Un secondo settore nel quale l'informatica può risultare utile nel lavoro dello studioso di scienze umanistiche è la gestione (intesa come memorizzazione, elaborazione e utilizzazione) delle informazioni strutturate. Tutte le informazioni necessarie all'elaborazione di un qualsiasi lavoro possono essere efficientemente memorizzate, oltre che rapidamente ritrovate ed elaborate, in un archivio elettronico. Questo archivio, se abbastanza grande e di interesse generale, può poi essere messo a disposizione della comunità scientifica.

3.2.1 Le basi di dati

Qualche anno fa si era soliti parlare di *banche dati*, ponendo l'accento sull'aspetto conservativo e di *information retrieval* (rinve-

nimento delle informazioni) proprio di queste applicazioni; ora si preferisce la denominazione *basi di dati* che, invece, evidenzia l'aspetto fondante dell'informazione per qualsiasi operazione e ne mette in rilievo l'aspetto dinamico e relazionale.

La "gestione dell'informazione" viene qui intesa in maniera più restrittiva di come sia stato fatto finora (anche scrivere, correggere, stampare un testo è infatti una gestione dell'informazione). Dal punto di vista di questi *pacchetti applicativi*, l'informazione è un *dato* che viene memorizzato in una forma precisa, in un posto determinato e all'interno di una struttura predefinita in grado di renderne il più agevole possibile il ritrovamento e l'elaborazione.

I data base e la struttura dei dati

I programmi utilizzati per queste operazioni vengono normalmente chiamati *programmi di data base* e, per usufruire di una figura sufficientemente esplicativa, potremmo dire che tali programmi consentono il passaggio dalla scheda di archivio cartacea alla scheda elettronica.

La struttura centrale di questi programmi è il *record* (che può essere paragonato alla vecchia scheda) a sua volta costituito da *field* (campi, che contengono ben distinti tra loro gli elementi che sono contenuti nei vari record). Così una base di dati bibliografica sarà costituita di record che contengono informazioni sui vari volumi e ogni record sarà (per esempio) composto di un *campo-autore*, di un *campo-titolo*, di un *campo-data*, di un *campo-editore*, di un *campo-luogo-di-edizione*.

RECORD NUMBER	1
AUTORE	RIPELLINO, Angelo Maria
TITOLO	Praga Magica
DATA	1973
EDITORE	Giulio Einaudi Editore
LUOGO DI EDIZIONE	Torino

Definita questa struttura fondamentale, il programma per la gestione di una base di dati deve fornire una serie di funzioni elementari. Chi usa un data base deve, ovviamente, essere in grado di: 1) creare un archivio; 2) definire la struttura di un record; 3) inserire dati nell'archivio; 4) ritrovare ed elaborare questi dati; 5) eliminare i dati indesiderati; 6) distruggere l'archivio.

Gli attuali DBMS (*Data Base Menagement System*) sono, naturalmente, dotati di tutte queste funzioni a vari livelli di raffinatezza e complicazione. In questa sede ci soffermeremo solo sulla funzione di ricerca del *query language* (linguaggio di interrogazione) in quanto utile a mettere in luce la logica profonda di tali programmi.