2

XML

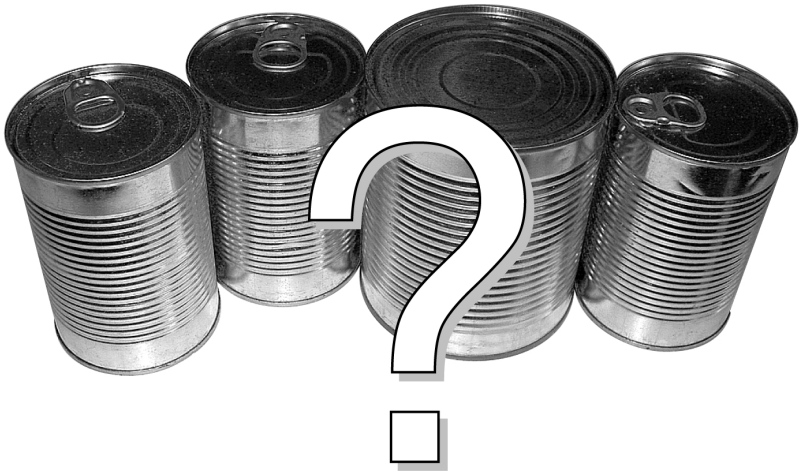
Un uovo, fin tanto che ha l'albume ed il tuorlo separati può essere usato per fare tutte quelle ricette nelle quali si richiede solo l'albume e solo il tuorlo. Basta sbatterlo un po' perché il nostro uovo passi in una condizione nella quale si potrà usare solo "sbattuto". L'uovo è sempre lo stesso: ciò che aumenta è l'entropia.

La stessa cosa vale per i contenuti. Tanto più sapremo tenere separate le varie componenti e mantenere la massima quantità possibile delle informazioni, tanto più potremo scegliere "cosa" pubblicare su "quale" media.

Si può pensare a XML come a una cura contro l'aumento di entropia!

L'etichetta e il pomodoro: un esempio di come già si usi la struttura delle informazioni nella vita di tutti i giorni

Pensiamo per un attimo ad un'operazione che avremo fatto mille volte: aprire una scatola di pomodori pelati ed utilizzarne il contenuto per realizzare la nostra ricetta preferita. Può sembrare strano, ma c'è una stretta connessione tra le operazioni che precedono e seguono lo svuotamento della lattina, ed il cross-media publishing. Il nostro scopo è di arricchire il piatto che stiamo preparando con un po' di "Pomodori Pelati Calabrì". Se pensiamo per un attimo all'indietro, a tutte le operazioni che ci hanno aiutato ad avere sopra il piano della cucina il nostro pomodoro pronto, queste, per la maggior parte, non hanno una connessione diretta con quella particolare verdura che interessa a noi.



Quale scegliere? Una sequenza di latte anonime non dà alcuna garanzia sulla tipologia del contenuto.

Proviamo a ripercorrere il viaggio. Scegliamo il negozio. Tra le varie possibili soluzioni optiamo per un centro commerciale, il "Tuttocasa". L'etichetta (che in questo caso è una vera e propria insegna luminosa) indica solo "Centro commerciale Tuttocasa": non c'è nulla che parla di pomodori pelati. Eppure noi decidiamo di andare lì. Tra i vari negozi che lo compongono, ci dirigiamo verso quello che ha l'etichetta (ancora un'insegna) "Maxi", sapendo che in quello c'è sicuramente ciò che ci interessa. All'interno del Maxi seguiamo l'etichetta per il reparto Alimentari, che stavolta non è un'insegna luminosa, ma la troviamo appesa in

alto in corrispondenza di una parte del negozio. Da casa fino a qui siamo arrivati "fidandoci" delle etichette, in quanto, fino ad ora di pomodori pelati non abbiamo visto neanche l'ombra. Tra le varie corsie del reparto Alimentari, ci infiliamo in quello che espone in alto il cartello "Cibi in scatola". Ci siamo vicini. Qui, se non ci fossero le etichette appiccicate ai barattoli, sembrerebbero tutti uguali. Noi cerchiamo quelli di un particolare tipo: i "Pomodori Pelati Calabrì". Tra i vari cibi in scatola, finalmente troviamo una bella lattina colorata con la scritta "Pomodori Pelati Calabrì". Però notiamo che ce ne sono due, quasi uguali, con la stessa scritta, ma una riporta una specifica di 250 grammi, un'altra di 500 grammi. Scegliamo la prima. Prendiamo il barattolo un po' ad occhi chiusi: non siamo sicuri di cosa ci sia dentro! Ma ci fidiamo delle etichette.

Le etichette ci hanno portato fino a questo scaffale, ancora una foto stampata su carta ci mostra i nostri prelibati pomodori. Però, ad essere sinceri, i pomodori veri non li abbiamo ancora visti. Portiamo via dallo scaffale un barattolo di latta con un foglietto colorato attaccato sopra, nella speranza di non avere strane sorprese.

Se facciamo un'analisi veloce della nostra giornata, tantissime scelte quotidiane sono basate su etichette, su metadati, su delle informazioni che marcano, o rimarcano, proprietà di oggetti.

Una volta a casa, con la nostra latta ancora chiusa, ci resta il dubbio. Ma, aperta la latta, tranquillamente ci fidiamo di buttare il contenuto nella pentola per preparare la nostra ricettina. Solo in quel momento abbiamo capito che fidarci delle etichette ci può aiutare a cercare, trovare e utilizzare il contenuto!!!

Per utilizzare uno specifico oggetto, spesso dobbiamo riferirci a delle informazioni che con questo non hanno alcun legame diretto, o meglio il legame c'è, ma è di tipo gerarchico. Ogni contenitore ha un'etichetta. La nostra esperienza ci aiuta a scegliere: dove c'è una certa etichetta, troveremo all'interno altri contenitori tra i quali scegliere il più conveniente. Questo procedimento si ripete fino a trovare un contenitore "speciale", che ha proprio il "contenuto" che cerchiamo! Ad esempio, sappiamo che "contenuti" nel centro commerciale ci sono vari negozi, un supermercato, un negozio di articoli sportivi, uno di calzature e così via.

Possiamo dire che centro commerciale, supermercato, reparto, corsia e scaffale, sono elementi "strutturali" della ricerca che stiamo facendo, nodi intermedi che non contengono direttamente i nostri "Pomodori Pelati Calabrì". La latta, oppor-

tunamente etichettata, invece, è un elemento "foglia", che contiene proprio solo il contenuto oggetto di tutto il discorso.

Proviamo ad immaginare uno scenario apocalittico dove manchino nodi di struttura, o addirittura le etichette sugli elementi foglia. Nel nostro caso il centro commerciale sarebbe una distesa omogenea di scatole e pacchi, alla completa rinfusa, dove non saremmo in grado di distinguere un barattolo di cibo per cani da uno di pelati, da uno di grasso per auto.

Può sembrare fuori luogo, ma, nel corso dei prossimi capitoli capiremo meglio questa affermazione: una pubblicazione orientata al mezzo, priva di un'adeguata struttura, è come questo caotico centro commerciale! Tutto alla rinfusa!

Un normale impaginato per la carta o per Internet, sia questo un file PDF generico, oppure una pagina Web, sono assimilabili ad un negozio dove la gente acquista i prodotti non in base ad un'etichetta che non hanno, ma, grazie all'esperienza, solo in funzione della forma del contenitore, o del fatto che questo sia posizionato in un certo punto del negozio, o vicino ad un determinato prodotto.

Se ci pensiamo bene, quando guardiamo una pagina di giornale, come facciamo a sapere che un testo in grassetto di un corpo elevato è il titolo dell'articolo?? L'esperienza ci insegna che c'è un certo equilibrio tra titolo, sottotitolo e corpo del testo. Generalmente sono raggruppati da elementi grafici che, in qualche modo, ne evidenziano l'unità (linee sottili, spaziatura adeguata, fondini). Se c'è un'immagine, questa deve essere collocata in prossimità del testo o, addirittura, al suo interno, facendolo scorrere. Tutte queste informazioni ci guidano a comprendere la "struttura" propria delle informazioni, ma lo fanno attraverso un "linguaggio", quello della forma grafica, che è molto facile per l'essere umano, ma abbastanza difficile per i computer. Per far digerire ad uno strumento "meccanico", come un elaboratore elettronico, informazioni che per noi risultano naturali, dobbiamo riuscire a schematizzarle, ad inscatolarle in un qualcosa che gli permetta di capirle. O meglio, se non proprio capirne il significato complessivo, almeno, essere in grado di gestirle e di farne ciò che noi vogliamo.

Ora vedremo come avrebbe fatto un computer per realizzare questo percorso.

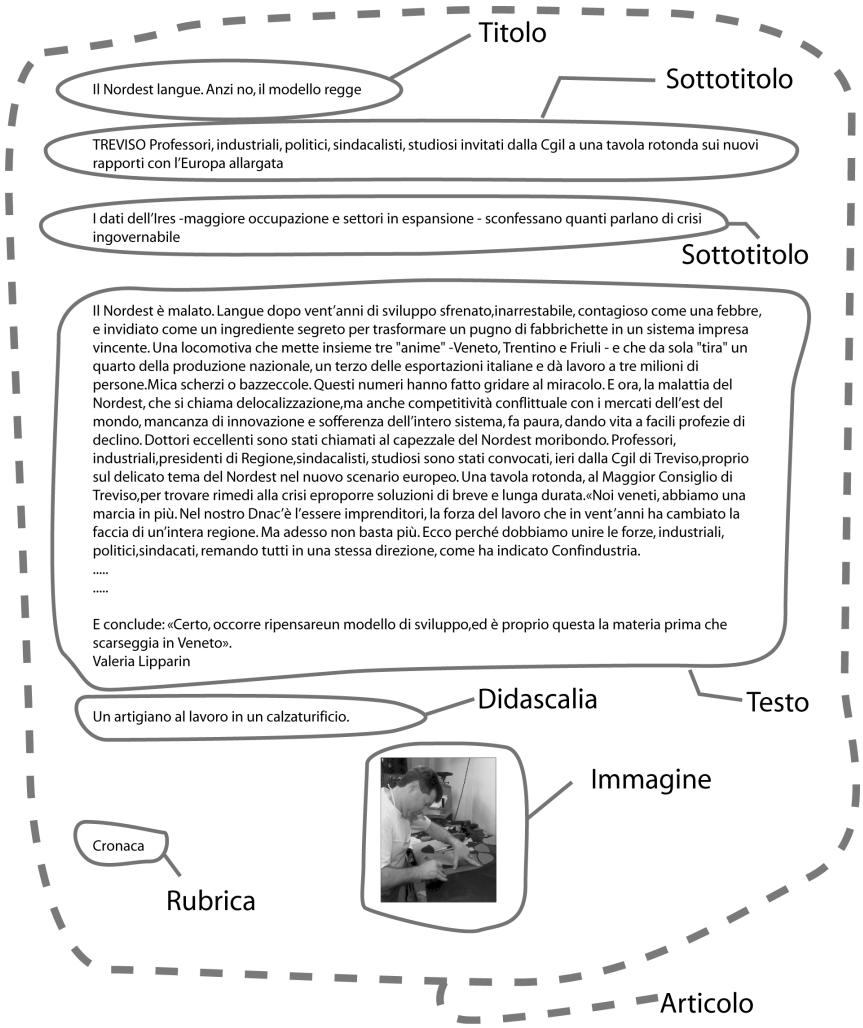
In una pagina di giornale il significato dei vari elementi è noto al lettore grazie alla sua esperienza. La forma, in modo implicito, riesce a trasmettere all'utente com'è fatta la struttura dell'informazione-contenuto che essa contiene.

La marcatura

Alla base di tutto sta il concetto di "marcatura". Ipotizziamo di avere un file di testo senza formattazione, stampato su carta. Questo contiene un titolo, un sottotitolo, il corpo dell'articolo, e qualche testo specifico da evidenziare. Per specificarne le varie parti, e comunicare ad un operatore le modifiche da apportare, prendiamo un pennarello e facciamo un cerchio attorno alle varie parti, segnando "cosa è cosa" con zone evidenziate in pennarello e marcatori. Il gioco è fatto. Abbiamo effettuato la marcatura!! Può sembrare banale, ma il concetto di fondo al cross-media publishing sta proprio qui! È necessario sapere cosa è cosa, distinguere in modo appropriato ogni pezzo di contenuto, marcandolo con un nome che lo descriva, ma che, con questo, non ha fisicamente nulla a che fare.

Che relazione c'è tra l'etichetta di carta con stampato il disegno dei "Pomodori Pelati Calabrì" ed i pomodori stessi? L'etichetta è carta e inchiostro. Il frutto è un prodotto commestibile. Nessun legame, se non il fatto che l'etichetta dice cosa è contenuto dentro la scatola. Allo stesso modo, la marcatura di un testo dice che cos'è quel testo. Stesso ragionamento vale per le immagini e per ogni altro tipo di "pezzo", o meglio "contenuto" da pubblicare (ogni elemento che noi abbiamo già imparato a chiamare "asset").

Useremo *XML*: *eXtensible Markup Language*, o *linguaggio di marcatura estensibile/estendibile*. Non fatevi confondere da altre sigle note. XML non ha quasi nulla



Marcare gli elementi e dire cosa sono è l'operazione che sta alla base di tutta la strategia cross-media. È un'operazione che facciamo abitualmente: ci basta imparare come effettuarla in modo che il computer possa capirci.

da condividere con HTML, se non il fatto di essere, come questo, un linguaggio di marcatura. HTML è nato per descrivere l'ipertesto, e oggi, a tutti gli effetti è un linguaggio per indicare ad un browser come le pagine Web devono apparire. Possiamo quasi dire che HTML ha molto più da spartire con PostScript e PDF, essendo, come questi, un linguaggio di descrizione della pagina.

XML invece è un "linguaggio di marcatura del contenuto"!! Con XML possiamo marcare qualsiasi cosa, come con un pennarello possiamo evidenziare qualsiasi

pezzo di testo, e indicare di cosa si tratta. Dobbiamo vedere XML come una "lingua" (italiano, inglese, spagnolo ...) che ha delle sue regole grammaticali e un vocabolario, e attraverso la quale possiamo descrivere oggetti, pensieri, situazioni e così via. XML, a rigore, ci permette di descrivere tutto. Come, generalmente, possiamo fare anche con una lingua moderna. Ma, di particolarmente originale, c'è il fatto che non esiste un vero e proprio vocabolario: l'insieme delle parole che si possono usare può essere inventato di volta in volta, a seconda delle necessità. Ci sono alcune regole grammaticali. Come approcciandosi ad una qualsiasi lingua, ci sono vari livelli, dal semplice linguaggio da turista a quello accademico, anche per XML ci si può limitare alle regole base, oppure approfondire e mettere regole su regole. Per fortuna, per le esigenze del CMP, le regole da sapere sono veramente poche, meno di quelle che servirebbero in inglese per riuscire a chiamare un taxi e farsi portare in centro città.

I tag

Sono il mattone di XML, non in senso che sono di "difficile digestione", bensì sono come l'elemento base di ogni grande costruzione realizzata dall'uomo. Un *tag* è una sequenza che deve rispettare una sintassi precisa. Ce ne sono di due tipi, *di apertura* e *di chiusura*, e si distinguono solo perché nel secondo è presente uno slash "/".

In formato XML, per marcare un testo come titolo, dobbiamo usare due tag titolo, uno di apertura e uno di chiusura, e tra questi mettere il titolo vero e proprio. Ad esempio:

```
<titolo>Divina commedia</titolo>
```

oppure

```
<autore>Dante Alighieri</autore>
```

Ecco finalmente formalizzata la nostra latta: la latta e l'etichetta stanno ai "Pomodori Pelati Calabrì" come i tag "titolo" o "autore" stanno a "Divina commedia" e a "Dante Alighieri".

Il lettore attento noterà subito come prima non si parlasse solo del contenitore vero e proprio dei pomodori, ma anche dei vari contenitori del contenitore. Giustamente dobbiamo specificare con altri tag anche le strutture o nodi di livello superiore. Una buona notizia è che tag di struttura e tag foglia, o di contenuto, si distinguono solo per ciò che contengono, non per particolari regole sintattiche. Unica attenzione da prestare è che ogni contenitore aperto sia anche chiuso.

Quindi, per formalizzare l'esempio dei pelati, in puro XML:

```
<centri_commerciali>
  <tuttocasa>
    <maxi>
      <alimentari>
        <cibi_in_scatola>
          <pomodori_pelati_calabri>
            POMODORI
          </pomodori_pelati_calabri>
          <fagioli_di_lamon>
            FAGIOLI
          </fagioli_di_lamon>
          <cibo_per_cani_canefelice_>
            CIBO_CANEFELICE
          </cibo_per_cani_canefelice_>
        </cibi_in_scatola>
        <surgelati>
        ....
        ....
        </surgelati>
        <latticini>
        ....
        ....
        </latticini>
      </alimentari>
      <casalinghi>
      .....
      .....
      </casalinghi>
      <detersivi>
      .....
      .....
      </detersivi>
    </maxi>
    <scarpasport>
    .....
    .....
    </scarpasport>
    <tuttofaidate>
    ....
    ....
    </tuttofaidate>
  </tuttocasa>
  <milanocasa>
  ....
  ....
  </milanocasa>
  <romacasa>
  ....
  ....
  </romacasa>
</centri_commerciali>
```

Il tag per l'informazione è come l'etichetta per la latta. Se riconosciamo l'etichetta sappiamo anche dire cosa c'è dentro la latta.

Per ovvie ragioni di spazio l'albero non è completo, ma, ad ogni nodo abbiamo posto alcune possibili scelte (indicate coi puntini di sospensione ...).

Tra tutti i centri commerciali (Tuttocasa, Romacasa e Milanocasa) abbiamo scelto Tuttocasa. Tra i vari negozi che Tuttocasa contiene (Maxi, Tuttosport e Casafaidate) scegliamo Maxi. Qui, tra i vari reparti (Alimentari, Casalinghi e Detersivi) puntiamo sugli Alimentari. Ancora, scegliamo la corsia dei cibi in scatola (tralasciando latticini e surgelati), per scegliere, finalmente i nostri "Pomodori Pelati Calabrì" tra i vari cibi in scatola presenti.

Come in una lingua non c'è un unico modo per esprimere un concetto, così in XML non si è vincolati ad una sola ed unica maniera di descrivere la nostra sequenza di ricerca.

Ne proponiamo due possibili, tra le tante, e di seguito, evidenziamo le rispettive peculiarità.

```
<centri_commerciali>
  <centro>
    <nome>
      tuttocasa
    </nome>
    <negozi>
      <negozio>
        <nome>
          maxi
        </nome>
        <tipo>
          supermercato
        </tipo>
        <reparti>
          <reparto>
```

```
        <nome>
          alimentari
        </nome>
        <corsie>
          <corsia>
            <nome>
              cibi_in_scatola
            </nome>
            <prodotti>
              <prodotto>
                <nome>
                  Pomodori Pelati Calabri
                </nome>
                <peso>
                  250 g
                </peso>
                <scadenza>
                  15-12-2009
                </scadenza>
                  ...
                  ...
              </prodotto>
              <prodotto>
                <nome>
                  Fagioli di Lamon
                </nome>
                <peso>
                  150 g
                </peso>
                <scadenza>
                  15-12-2008
                </scadenza>
                  ...
                  ...
              </prodotto>
              <prodotto>
                <nome>
                  Cibo per cani Canefelice
                </nome>
                <peso>
                  500 g
                </peso>
                <scadenza>
                  15-12-2022
                </scadenza>
                  ...
                  ...
              </prodotto>
            </prodotti>
          </corsia>
        </corsie>
      </reparto>
    </reparti>
  </negozio>
 </negozi>
</centro>
</centri_commerciali>
```

In questo caso i nomi dei contenitori non sono i tag stessi. Si è preferito utilizzare un tag figlio per indicare informazioni specifiche. Ad esempio, per `<centro>` il tag `nome`, oppure per `prodotto`, ancora `nome`, `peso` e `scadenza`.

La struttura, che poi si ripete, è di questo tipo:

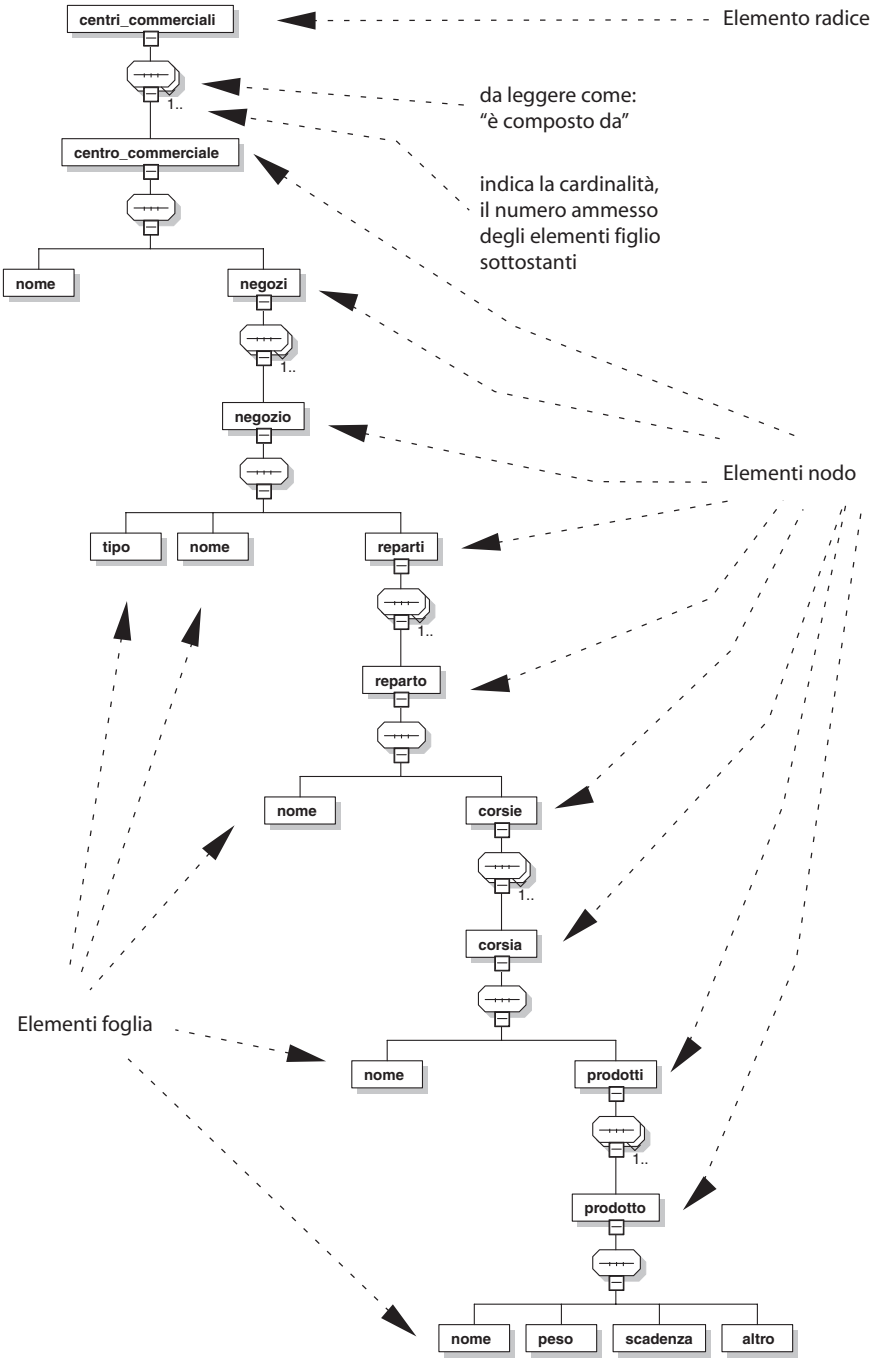
```
<centro>
  <nome>
    tuttocasa
  </nome>
  <negozi>
    ...
    ...
  </negozi>
</centro>
```

e

```
<prodotto>
  <nome>
     Fagioli di Lamon
  </nome>
  <peso>
    150 g
  </peso>
  <scadenza>
    15-12-2008
  </scadenza>
  ...
  ...
</prodotto>
```

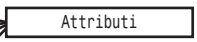
In questa modalità la struttura risulta un po' più complessa, ma molto più modulare. Allo stesso livello di `<nome>` potrebbero essere incluse con facilità anche tutta una serie di altre informazioni generali (come l'indirizzo, la categoria, la catena di cui fa parte e altri dati vari). Inserire tutte queste notizie in un unico tag, usando gli attributi, che vedremo tra un attimo, a volte non è possibile (o, almeno, non consigliabile). È quindi spesso necessario optare per qualche livello in più che riesca a mostrare con completezza i dati che dobbiamo rappresentare.

Una seconda alternativa consiste nell'utilizzo degli attributi. Un *attributo* è un'informazione aggiuntiva al tag. Non è necessariamente un elemento strutturale, ma caratterizza uno specifico tag rispetto agli altri della stessa categoria. Permette di rendere la struttura più snella e compatta, pur mantenendo tutte le informazioni e garantendo un certo grado di flessibilità.



Rappresentiamo con un albero, completo di elementi foglia e nodi struttura, una sequenza possibile che ci può guidare alla scelta dei prodotti desiderati.

```
<centri_commerciali>
  <centro nome= tuttocasa >
    <negozi>
      <negozio tipo=supermercato nome=maxi >
        <reparti>
          <reparto nome= alimentari>
            <corsie>
              <corsia nome=cibi_in_scatola>
                <prodotti>
                  <prodotto peso="250 g" scadenza=15-12-2009
                    Pomodori pelati calabri
                  </prodotto>
                  <prodotto peso="500 g" scadenza=15-12-2009
                    Pomodori pelati calabri
                  </prodotto>
                  <prodotto peso="150 g" scadenza=15-12-2008
                    Fagioli di Lamon
                  </prodotto>
                  <prodotto peso="500 g" scadenza=15-12-2022
                    Cibo per cani Canefelice
                  </prodotto>
                </prodotti>
              </corsia>
            </corsie>
          </reparto>
        </reparti>
      </negozio>
    </negozi>
  </centro>
</centri_commerciali>
```



Il problema di includere il nome del centro in questo terzo esempio è stato risolto con gli attributi nome:

```
<centro nome= tuttocasa>
```

Ad un tag possono essere abbinati un numero indefinito di attributi. Discriminando due tag uguali attraverso un attributo, non abbiamo più bisogno, come nel caso precedente, di includere un tag specifico a livello inferiore. Ad esempio `<centro nome=tuttocasa>` e `<centro nome=milanocasa>` sono tag uguali, allo stesso livello, però distinti dall'attributo `nome`. La stessa latta di pelati, in versione 250 gr e 500 gr può essere convenientemente distinta da un attributo `peso`. La capacità di scegliere quale delle due soluzioni sia più conveniente necessita un po' di esperienza. A questo livello di trattazione ci basti sapere che con XML, come con un qualsiasi altro linguaggio, è possibile descrivere oggetti uguali anche con combinazioni di parole diverse.

Prima di procedere, sottolineiamo solo un'esigenza: quando si presenta una sequenza di oggetti simili, è necessario raggrupparli e portare al livello superiore la

caratteristica che li accomuna. Quindi, `lista dei negozi`, ad esempio, sarà un insieme di singoli elementi di tipo `<negozio>`, inclusi in un tag sequenza a livello superiore di tipo `<negozi>`.

Pomodoro puro

Ora che abbiamo un'idea un po' più chiara di cosa possa significare descrivere una struttura, una sequenza di contenitori e contenuti, di nodi strutturali e di foglie, proviamo a soffermarci un attimo sull'utilizzo di tutto questo.

Che differenza c'è tra il pomodoro puro, ancora contenuto nella mia latta di "Pomodori Pelati Calabrì", e quello che ho già utilizzato per preparare la pizza o il ragù o le bistecche alla pizzaiola?? È abbastanza facile distinguere il pomodoro nella pizza, un po' più difficile nel ragù, ma, in questo caso, il senso del gusto ci può essere d'aiuto. Pomodoro è sempre pomodoro! Ma ha dei "gradi di lavorabilità" diversi a seconda di come lo utilizziamo. Mentre col pomodoro ancora nella latta possiamo decidere se usarlo per la pizza o per il ragù, dopo averlo usato per il nostro pentolone di ragù ci risulta estremamente difficile poterlo riprendere, estrarre e riutilizzare per la pizza!! Volendo fare un accenno più scientifico, si tratta di un problema di entropia: ordinare e separare costa molto di più che mescolare e omogeneizzare.

Dalla visione apocalittica di un centro commerciale destrutturato, distesa omogenea di barattoli e scatole anonime, siamo riusciti, a colpi di etichetta, a creare una struttura che ci aiuti a scegliere il nostro pomodoro preferito. Potremmo pensare di tenerlo in eterno nella latta, bello puro, così com'è, e potenzialmente pronto ad ogni utilizzo!!! E passeremo subito dal massimo del caos al massimo dell'ordine, dove tutto ha un suo specifico posto e nulla si mescola, si contamina con qualcos'altro.

Tra queste due visioni estreme si muove la nostra discussione sul cross-media publishing, dando ordine e struttura alle informazioni pure, per poi mescolarle con gli opportuni ingredienti di forma e creare squisite ricette di prodotti di comunicazione, abbinati ai media. Come il pomodoro, per formare la pizza, deve fondersi con mozzarella e origano, e per fare il ragù deve amalgamarsi con spezie e carne, così, l'informazione strutturata e pura, abbinata a differenti combinazioni di forma e ottimizzata per uno specifico media, diventerà un prodotto cartaceo o

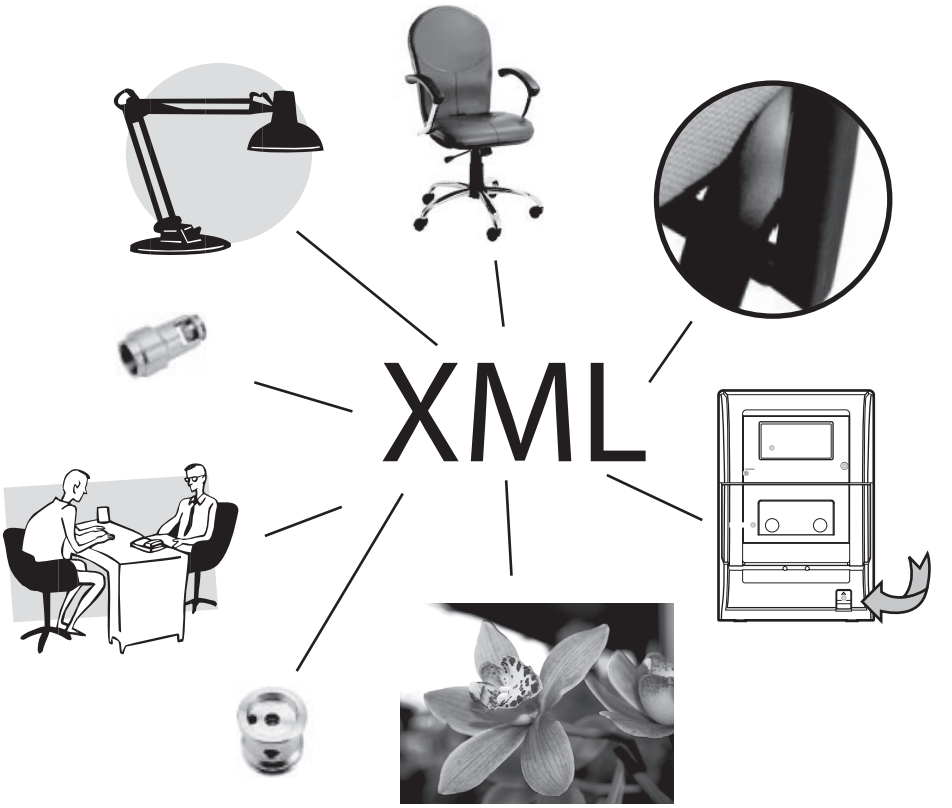
I prodotti puri, in mano al cuoco, possono assumere, a seconda delle ricette, sapori e colori particolari, come le informazioni pure a seconda del media. La ricetta che, dallo stesso prodotto, permetterà a chi la prepara di ottenere piatti diversi, sarà come la forma per lo specifico media, che dallo stesso contenuto farà nascere prodotti di comunicazione differenti, ma allo stesso tempo simili.

una pagina Web, mantenendo nel contempo un po' del suo "sapore" originale, e arricchendosi delle potenzialità intrinseche del mezzo.

XML sta nel mezzo. È un po' come la dispensa per il cuoco. La sorgente di prodotti semilavorati pronti per l'utilizzo definitivo. Per tutto ciò che non è testo, o informazione non assimilabile a testo, aiuteranno XML una serie di formati "ricchi", in un certo qual modo "puri", che garantiranno un elevato grado di lavorabilità per le immagini fotorealistiche e di sintesi (a matrice e vettori) e gli eventuali altri "asset". Di questo parleremo nel prossimo capitolo.

Schematizzare l'esperienza

Proseguendo nel nostro esempio, ora che è chiaro come un contenuto puro possa essere facilmente mescolato ad una forma per comporre prelibate ricette di comunicazione, ora che sappiamo che nel centro commerciale le cose sono ben ordinate in base alle etichette dei contenitori, ci chiediamo se in qualche modo si possa formalizzare anche l'esperienza che ci ha guidato da casa nostra alla latta di pelati nello scaffale, all'interno del centro commerciale.



XML è al centro degli asset, vettoriali e a matrice. A questi è connesso tramite link con riferimento relativo o assoluto. Viene allo stesso tempo mantenuto il legame della referenza e la purezza dell'informazione nel formato più consono a questa.

All'inizio abbiamo detto come l'esperienza ci aiuti a distinguere i negozi, i reparti e le categorie. In base all'esperienza facciamo le nostre scelte. Potremo dire che, in XML, c'è la possibilità di specificare uno "schema" che, dualmente, aiuti chi deve mettere le latte negli scaffali e chi debba cercarle per poi utilizzarle. È necessario uno strumento che dia ordine agli oggetti, e, da un lato, costringa chi è incaricato della distribuzione a mettere ogni cosa al suo posto, in attesa che qualcuno la scelga, dall'altro permetta al cliente del negozio, che vuole trovare quel determinato prodotto, di reperirlo con facilità, basandosi su un'appropriata sequenza di etichette specificata. Chi cerca deve poter muoversi di nodo in nodo, riferendosi a questo strumento, così da poter giungere fino a quella specifica foglia in modo univoco, e non rischiare di perdersi in qualche percorso inappropriato. A

livello formale, possiamo rispondere a questa esigenza con due strumenti: *DTD* (*Document Type Definition* o *definizione del tipo di documento*) e *XML Schema*.

Come già accennato nel precedente capitolo, parleremo solo dei secondi. Per le esigenze del cross-media publishing si potrebbero utilizzare in modo perfettamente equivalente. A nostro avviso, la sintassi delle DTD è un po' più ostica, essendo meno discorsiva e più simbolica. Lasciamo al lettore interessato la possibilità di approfondire.

NOTA

Col termine schema, in carattere normale (tondo), ci riferiremo al concetto di schema. Con *schema*, in corsivo, al file XML vero e proprio, all'*XML Schema*, che formalmente, definisce lo schema della nostra struttura di informazioni.

Una definizione sintetica di *schema* è la seguente:

vocabolario XML per esprimere le regole di impiego dei nostri dati.

Prima di affrontare l'esempio dei pelati, che potrebbe risultare troppo complesso, vediamo un caso semplice, con un albero a due livelli.

Dalle regole allo schema

Dobbiamo creare uno *schema* che descriva le caratteristiche generali di alcune persone: un'anagrafica semplificata. Specifichiamo le regole:

- l'anagrafica è composta da persone;
- ogni persona ha un nome, un cognome, una data di nascita, una città di nascita, un numero di figli;
- nome, cognome e città di nascita sono semplici testi;
- data di nascita è una data;
- numero di figli è un numero.

Ora analizziamo uno *schema XML* per rappresentare queste regole.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
 <xsd:element name="anagrafica">
  <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="persona" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="nome" type="xsd:string"/>
              <xsd:element name="cognome" type="xsd:string"/>
              <xsd:element name="data_di_nascita" type="xsd:dateTime"/>
              <xsd:element name="citta" type="xsd:string"/>
              <xsd:element name="n_figli" type="xsd:int"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Prima di tutto, non è il caso di spaventarsi!!!

Dobbiamo ricordare che XML è un linguaggio per descrivere i contenuti, che sta a metà tra gli esseri umani e i computer. Se pensiamo che questi ultimi lavorano solo con incomprensibili sequenze di numeri, il fatto che, in qualche modo, possano essere in grado di comprendere un linguaggio composto di parole "comprensibili" anche da noi umani dovrebbe rincuorarci abbastanza.

Con un po' di pazienza cerchiamo di capire cosa c'è dietro queste stringhe. Scopriremo che, appena imparate le prime regole formali, sarà abbastanza facile individuare l'albero di struttura e, di conseguenza, le regole che noi stessi avevamo definito. Le prime righe e l'ultima sono in un certo modo pre-confezionate. Devono essere così per forza.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
....
....
</xsd:schema>
```

La prima riga è una sorta di dichiarazione: questo è un file XML, che segue le indicazioni della versione 1.0 di XML. La seconda e l'ultima aprono e chiudono lo *schema*. Per essere più precisi, contengono completamente lo *schema*, quindi ne sono la "radice". La radice di un albero è il punto di partenza, le fondamenta su cui costruire la nostra struttura.

Come attributo del tag `schema` notiamo una strana stringa `xmlns:xsd="http://www.w3.org/2001/XMLSchema`. `xmlns` sta per *XML name space*, cioè *spazio dei nomi*, se vogliamo il "contenitore dei termini", l'insieme dei vocaboli, il *vocabolario*, appunto. In questo caso si specifica che questo *schema* utilizzerà dei termini dichiarati nella `xsd` (*XML Schema Definition*), che non è altro che un super insieme di vocaboli, definiti dal consorzio degli standard per Internet (World Wide Web Consortium, o W3C), in modo univoco per tutti. Fortunatamente, non ci servirà sapere altro. Per la creazione dei nostri *schemi* non utilizzeremo altri spazi dei nomi. Qui la sigla xsd, xs o altro che precede il nome del tag è solo un riferimento al vocabolario al quale il tag appartiene, e che ci sia o meno non interessa ai fini della nostra trattazione. Lasciamo al lettore interessato la facoltà di approfondire l'argomento.

In sostanza, ci basti sapere che ogni *schema* deve cominciare con queste due righe

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

e finire con questa

```
</xsd:schema>
```

L'anagrafica è composta da persone! Questo, in XML, si dice così:

```
<xsd:element name="anagrafica">
 <xsd:complexType>
  <xsd:sequence>
    <xsd:element name="persona" maxOccurs="unbounded">
```

Il nodo `"anagrafica"` sarà composto di elementi `"persona"`. Ce ne saranno da un minimo di 1 ad un massimo non specificato. Ogni persona ha un nome, un cognome, una data di nascita, una città di nascita, un numero di figli. In XML:

```
<xsd:element name="persona" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="nome" type="xsd:string"/>
      <xsd:element name="cognome" type="xsd:string"/>
      <xsd:element name="data_di_nascita" type="xsd:date"/>
      <xsd:element name="citta" type="xsd:string"/>
      <xsd:element name="n_figli" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Il tag di elemento-nodo `"persona"` è appunto un nodo di tipo complesso. È composto da una sequenza di elementi-foglia di tipo specifico, a seconda dei dati veri e propri che contiene. Quindi `"persona"` dovrà contenere un'occorrenza di `nome`, `cognome`, `data_di_nascita`, `citta` (accenti e spazi non sono ammessi nei no-

mi dei tag), `n_figli`. Mentre `nome`, `cognome` e `citta` conterranno del semplice testo, come specificato dall'attributo `type="xsd:string"`. Il tag `data_di_nascita` conterrà un testo in formato data, come notiamo da `type="xsd:date"`. Mentre `n_figli` sarà invece un numero `type="xsd:int"`. Ogni tag aperto trova poi il relativo tag di chiusura.

Riprendendo le nostre regole impostate:

- l'anagrafica è composta da persone;
- ogni persona ha un nome, un cognome, una data di nascita, una città di nascita, un numero di figli;
- nome, cognome e città di nascita sono semplici testi;
- la data di nascita è una data;
- il numero di figli è un numero.

Risulta evidente come lo *schema* così impostato non sia altro che una loro pedissequa formalizzazione. I tag possono contenere vari tipi di dato. Il più utilizzato nei flussi cross-media è senz'altro `string`, che permette di alloggiare comodamente ogni tipo di testo composto da lettere e numeri.

Riportiamo una lista non completa, solo per dare un'idea di cosa potrebbe essere parte del nostro *schema*. XML è estremamente flessibile, ed è addirittura possibile inventarsi dei tipi di dato ad hoc per ogni esigenza. Ci limitiamo a valutare quelli predefiniti per non annoiare il lettore con tecnicismi non necessari. A sinistra il nome del tipo di dato, a destra un esempio.

Tipi di dato base, atomici, predefiniti

string	"Ciao mondo"
boolean	{vero, falso, 1, 0}
int	7
decimal	7.08
float	12.56E3, 12, 12560
duration	P1Y2M3DT10H30M12.3S
dateTime	formato: CCYY-MM-DDThh:mm:ss 2004-8-26 14:16
time	formato: hh:mm:ss.sss 14:16:32.127
date	formato: CCYY-MM-DD 2004-8-26
gYearMonth	formato: CCYY-MM 2004-8
gYear	formato: CCYY 2004
gMonthDay	formato: --MM-DD 8-26

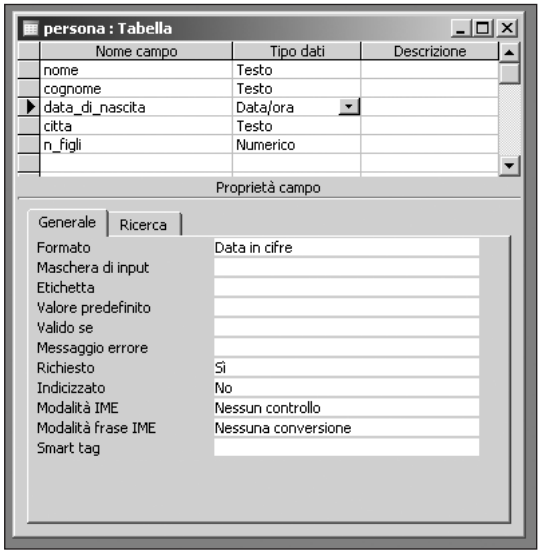
Il tipo `string` può contenere ogni tipo di testo destrutturato, composto da lettere, numeri e punteggiatura. Il tipo `boolean` può contenere dei valori booleani bi-

nari, il tipico sì/no, vero/falso, attivo/disattivo, presente/assente. I tipi `int`, `decimal` e `float` possono contenere vari tipi di numero, a seconda della precisione richiesta nei decimali dopo la virgola. Vari formati permettono di indicare il tempo. Da un numero di secondi a una data specificata in secoli, anni, mesi, giorni, ore, minuti, secondi e addirittura millesimi. Purtroppo l'ordine delle cifre segue le abitudini anglosassoni.

La tabella, primo esempio di struttura

Il lettore un po' smaliziato con la strutturazione dei dati si sarà già posto la domanda: "ma perché scomodare tante righe di complessa codifica quando basta fare una tabella?". L'osservazione è senz'altro appropriata!

Quando la struttura dei dati che andiamo a formalizzare è particolarmente semplice, risulta abbastanza intuitivo pensare ad un contenitore che ordini tutto su righe e colonne. In una tabella, le colonne danno il nome e specificano il tipo degli oggetti contenuti (campi), le righe sono le occorrenze degli elementi descritti (record). Notiamo subito, e ce ne renderemo conto sempre di più anche nei prossimi capitoli, che ci sono vari modi per rappresentare una certa struttura intrinseca all'informazione e che è abbastanza semplice passare da uno all'altro. Proviamo, ad esempio, ad utilizzare Access per realizzare la tabella dell'anagrafica.



Con Microsoft Access, uno dei database più conosciuti, è abbastanza semplice specificare le regole estrapolate nella fase di analisi per caratterizzare le informazioni su cui si lavora.

Nella semplice visualizzazione della struttura imporremo le regole che abbiamo formalizzato nello *schema*. Il risultato sarà una tabella che contiene una persona per ogni riga e una proprietà per ogni colonna.

Arrivare a realizzare una tabella di questo genere con Access è un'operazione alla portata di tutti. Con una semplice esportazione si può poi passare alla descrizione di queste informazioni secondo una codifica in XML.

Le ultime versioni di molti prodotti commerciali comprendono funzionalità di importazione ed esportazione in XML. In questo caso, Access permette di esportare sia il file XML dei dati, che una sua interpretazione dello *schema*.

Il file XML esportato è il seguente (dopo la rimozione di alcune righe fuorvianti e non necessarie):

```
<?xml version="1.0" encoding="UTF-8" ?>
<dataroot>
<persona>
    <nome>Pino</nome>
   <cognome>Pini</cognome>
   <data_di_nascita>1973-12-13T00:00:00</data_di_nascita>
    <citta>Milano</citta>
    <n_figli>2</n_figli>
</persona>
<persona>
   <nome>Marco</nome>
   <cognome>Galiazzo</cognome>
   <data_di_nascita>1971-05-22T00:00:00</data_di_nascita>
   <citta>Venezia</citta>
   <n_figli>0</n_figli>
</persona>
<persona>
   <nome>Rino</nome>
   <cognome>Vedi</cognome>
   <data_di_nascita>1950-01-01T00:00:00</data_di_nascita>
   <citta>Napoli</citta>
  <n_figli>4</n_figli>
</persona>
</dataroot>
```

Lo *schema* relativo (sempre esportato direttamente da Access) rappresenta tra le regole anche una serie di proprietà che Access imposta di default. Prima presentiamo la versione integrale, poi una versione "ripulita" dalle regole che a noi non interessano e che potrebbero infastidire il lettore non abituato a queste codifiche.

Versione integrale (come da esportazione diretta su file .xsd da Access):

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"xmlns:od="urn:schemas-microsoft-com:officedata">

<xsd:element name="dataroot">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="persona" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attribute name="generated" type="xsd:dateTime" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="persona">
  <xsd:annotation>
    <xsd:appinfo />
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="nome" minOccurs="1" od:jetType="text" od:sqlSType="nvarchar" od:nonNullable="yes">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:maxLength value="50" />
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>

      <xsd:element name="cognome" minOccurs="1" od:jetType="text" od:sqlSType="nvarchar" od:nonNullable="yes">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:maxLength value="50" />
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>

      <xsd:element name="data_di_nascita" minOccurs="1" od:jetType="datetime" od:sqlSType="datetime"
      od:nonNullable="yes" type="xsd:dateTime" />

      <xsd:element name="citta" minOccurs="1" od:jetType="text" od:sqlSType="nvarchar" od:nonNullable="yes">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:maxLength value="50" />
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>

      <xsd:element name="n_figli" minOccurs="1" od:jetType="longinteger" od:sqlSType="int" od:nonNullable="yes"
type="xsd:int" />
```

```
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

Versione snellita (senza ulteriori vincoli imposti di default dal programma):

```
<?xml version="1.0" encoding="UTF-8" ?>

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

  <xsd:element name="dataroot">
    <xsd:complexType>
      <xsd:sequence>
          <xsd:element ref="persona" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="persona">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="nome" type="xsd:string"/>
        <xsd:element name="cognome" type ="xsd:string"/>
        <xsd:element name="data_di_nascita" type="xsd:dateTime"/>
        <xsd:element name="citta" type ="xsd:string"/>
        <xsd:element name="n_figli" type="xsd:int" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Notiamo come la definizione del tag `"persona"` sia stata divisa in due parti: una prima dichiarazione come elemento sequenza in dataroot (la radice) alla quale poi fa riferimento una dichiarazione estesa che ne comprende tutti gli elementi.

Quando è necessario impostare a livello di *schema* quante occorrenze siano accettabili di un determinato tag, dobbiamo usare gli attributi `minOccurs` e `maxOccurs`. Se nessuno dei due è specificato, significa che lo *schema* impone la presenza di uno e uno solo di quei tag.

Ad esempio `<xsd:element name="nome" type="xsd:string"/>` è equivalente a `<xsd:element name="nome" minOccurs="1" maxOccurs="1" type="xsd:string"/>`.

Se volessimo specificare che il cognome può essere rappresentato da un minimo di nessun valore ad un massimo di tre, dovremmo usare qualcosa tipo `<xsd:ele-`

`ment name="cognome" minOccurs="0" maxOccurs="3" type ="xsd:string"/>`. Il termine `"unbounded"` indica invece la mancanza di un limite. Quindi `minOccurs="0" maxOccurs="unbounded"` permette che non ci sia alcun elemento, che ce ne sia uno, o addirittura un numero qualsiasi.

Nel capitolo 3 vedremo degli strumenti commerciali per definire *schemi* complessi. Purtroppo per questa fase della lavorazione delle informazioni, cioè per la formalizzazione della struttura, non sono ancora molto diffusi prodotti semplici, intuitivi e potenti. Nell'implementazione di un flusso di lavoro cross-media publishing sarà a volte necessario un piccolo intervento formativo perché qualche operatore acquisisca la capacità di trasformare le regole testuali in uno *schema* completo, efficiente e comprensibile. Uno strumento che, pur essendo in grado di fare molto di più, può essere avvicinato senza troppa fatica per la realizzazione di *schemi*, è senz'altro XMLSpy di Altova, che abbiamo utilizzato per realizzare molti dei grafici in questo testo, e del quale parleremo in modo più esteso nel prossimo capitolo.

XML e database

Il vantaggio nell'utilizzare XML come strumento informatico per la strutturazione dei contenuti risulta evidente quando si superi di poco la complessità della singola tabella. Nel caso di database relazionali, bisogna cominciare a tessere relazioni e legami tra celle e colonne. Potendo lavorare solo su due livelli per tabella è necessario impostare vincoli e c'è una rigidità abbastanza forte.

Con XML, si possono impostare negli *schemi* regole molto flessibili, e strutturare livelli su livelli risulta veramente agevole. Ad esempio, nell'anagrafica vista poco fa, volendo includere nel modello nome e cognome di eventuali figli, nel caso del database è necessario creare una nuova tabella contenente i figli e creare i vincoli referenziali tra figli e padri. Poi, l'esportazione non risulta sempre molto agevole. Con XML sarà sufficiente la seguente modifica allo *schema*, nel tag `"persona"`:

```
<xsd:element name="persona">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="nome" type="xsd:string"/>
      <xsd:element name="cognome" type ="xsd:string"/>
      <xsd:element name="data_di_nascita" type="xsd:dateTime"/>
      <xsd:element name="citta" type ="xsd:string"/>
      <xsd:element name="figlio" minOccurs="0" maxOccurs="unbounded" >
        <xsd:complexType>
          <xsd:sequence>
          <xsd:element name="nome_f" type="xsd:string"/>
          <xsd:element name="cognome_f" type ="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

e, direttamente nell'unico file XML si potranno includere gli eventuali figli:

```
<?xml version="1.0" encoding="UTF-8" ?>
<dataroot>

<persona>
   <nome>Pino</nome>
   <cognome>Pini</cognome>
   <data_di_nascita>1973-12-13T00:00:00</data_di_nascita>
   <citta>Milano</citta>
   <figlio>
     <nome_f>Giovanni</nome_f>
     <ccognome_f>Pini</cognome_f>
   </figlio>
   <figlio>
     <nome_f>Giuseppe</nome_f>
     <ccognome_f>Pini</cognome_f>
   </figlio>
</persona>

<persona>
   <nome>Marco</nome>
   <cognome>Galiazzo</cognome>
   <data_di_nascita>1971-05-22T00:00:00</data_di_nascita>
   <citta>Venezia</citta>
   <figlio />
</persona>

<persona>
  <nome>Rino</nome>
  <cognome>Vedi</cognome>
  <data_di_nascita>1950-01-01T00:00:00</data_di_nascita>
  <citta>Napoli</citta>
  <figlio>
    <nome_f>Lara</nome_f>
    <cognome_f>Vedi</cognome_f>
```

```
  <figlio>
    <nome_f>Mariolina</nome_f>
    <cognome_f>Vedi</cognome_f>
  </figlio>
  <figlio>
    <nome_f>Luisa</nome_f>
    <cognome_f>Vedi</cognome_f>
  </figlio>
  <figlio>
    <nome_f>Evelina</nome_f>
    <cognome_f>Vedi</cognome_f>
  </figlio>...
</persona>

</dataroot>
```

Pomodori strutturati: dal caos all'ordine attraverso la strutturazione

Dopo aver analizzato con attenzione questo semplice esempio di anagrafica, riprendiamo i pomodori, con tanto di centro commerciale, e cerchiamo di capire, in questo caso, come avremmo dovuto comportarci. Ecco le regole:

- partiamo dalla scelta del centro commerciale (Tuttocasa, Milanocasa, ...);
- ogni centro ha dei negozi di diverse categorie (Maxi, Scarpasport, Tuttofaidate, ...);
- ogni negozio ha dei reparti (Alimentari, Casalinghi, Tennis, Calcio, Bricolage, Ferramenta, ...);
- ogni reparto ha delle corsie (Cibi in scatola, Surgelati, Abbigliamento da tennis, Accessori da tennis, Attrezzi, Viti, Collanti, ...);
- in ogni corsia ci saranno le confezioni dei prodotti, che conterranno i prodotti in vendita.

Quindi, una versione orientativa dello *schema* potrà essere la seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 4 U (http://www.xmlspy.com) by Marco Galiazzo (Marco Galiazzo) -->
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="centri_commerciali">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="centro_commerciale">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nome"/>
```

```
<xs:element name="negozi">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="negozio">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="tipo"/>
            <xs:element name="nome"/>
            <xs:element name="reparti">
              <xs:complexType>
                <xs:sequence maxOccurs="unbounded">
                  <xs:element name="reparto">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="nome"/>
                        <xs:element name="corsie">
                          <xs:complexType>
                            <xs:sequence maxOccurs="unbounded">
                              <xs:element name="corsia">
                                <xs:complexType>
                                  <xs:sequence>
                                    <xs:element name="nome"/>
                                    <xs:element name="prodotti">
                                      <xs:complexType>
                                        <xs:sequence maxOccurs="unbounded">
                                          <xs:element name="prodotto">
                                            <xs:complexType>
                                              <xs:sequence>
                                                <xs:element name="nome"/>
                                                <xs:element name="peso"/>
                                                <xs:element name="scadenza"/>
                                                <xs:element name="altro"/>
                                              </xs:sequence>
                                            </xs:complexType>
                                          </xs:element>
                                        </xs:sequence>
                                      </xs:complexType>
                                    </xs:element>
                                  </xs:sequence>
                                </xs:complexType>
                              </xs:element>
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

Nota: lo spazio dei nomi questa volta è xs. XMLSpy usa un vocabolario per gli Schemi diverso da quello di Access, ma per il lettore non cambia nulla.

```
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Forse, in prima analisi, la composizione di questo *schema* potrà sembrare complessa. Ma se pensiamo che un albero di questo tipo, un po' ottimizzato, potrebbe essere sufficiente per la strutturazione di un intero centro commerciale, risulta evidente come un piccolo sforzo possa essere ampiamente ripagato.

A questo punto della trattazione, prima di passare ad analizzare i tre esempi di pubblicazione incrociata di cui abbiamo parlato nel primo capitolo, che saranno il nostro laboratorio di riferimento, cerchiamo di domandarci "che tipo di contenuto" sia possibile strutturare con XML.

Nell'esempio dei pomodori pelati, ci siamo limitati a mettere come foglia il `"nome"` del prodotto. Seguendo l'albero dalla radice, ad un certo punto si arriverà, per ognuno dei beni venduti in uno dei centri commerciali, a specificare il puro e semplice nome (o, al massimo, scadenza e peso), secondo l'indicazione del tag `<xsd:element name="nome" type="xsd:string"/>`.

Cosa dobbiamo fare se vogliamo catalogare, dopo tanta fatica, altre informazioni nel nostro XML ben strutturato? Apparentemente XML non ci lascia molta scelta. Un tag può contenere solo testo e, al massimo, valori numerici o booleani (sì/no). Tutto qua! E, per fortuna, questo testo è proprio ciò che ci basta per tutti i nostri desideri di comunicazione cross-media. Infatti, una particolare forma di testo è il "link", il riferimento esterno.

XML può riferirsi ad un qualsiasi oggetto esterno, posizionato virtualmente in ogni angolo dell'universo (o, almeno, in ogni sperduto bit accessibile attraverso una rete, e attraverso Internet in particolare). Quindi possiamo dire che XML gestisce in modo nativo il testo, e in modo referenziato qualsiasi altra cosa. Se, quindi, vorremmo poter vedere l'immagine dei nostri pomodori pelati, basterà riuscire in qualche modo a sapere dove si possa trovare una foto che li ritragga. Idem per l'intera etichetta che avvolge la latta. Conoscendo la posizione del file PDF che la descrive, basterà aggiungere, vicino al tag foglia `"nome"`, un tag "link_etichetta", per poter facilmente catalogare, e quindi accedere, alla versione digitale dell'etichetta.

Estendendo il concetto alla realtà tridimensionale, nulla mi vieta di inserire il posizionamento esatto in coordinate cartesiane della latta nella corsia e nel negozio. A livello di codifica, per riferirsi a qualcosa che non è possibile descrivere direttamente come testo XML, si useranno degli attributi per i tag. Ad esempio:

```
<immagine href="file://immagini/pomodoripelaticalabri.psd" />
```

punterà ad un file in formato nativo di Photoshop, indicando la posizione specifica rispetto al file XML a cui si riferisce. Questa è la modalità di *riferimento relativo*. Se ci fosse stato un percorso indicante un dominio Internet, ad esempio, avremmo avuto un *riferimento assoluto*:

```
<immagine href="http://www.tuttocasa.it/maxi/immaginipsd/pomodoripelaticalabri.psd" />
```

Quindi, per tirare le somme, con XML ci si può riferire ad ogni "pezzo" di realtà codificabile e, in qualche modo, referenziabile attraverso qualsiasi sistema si scelga. Questi "pezzi", come già accennato, saranno i nostri "asset", gli atomi della costruzione. Naturalmente, ai fini della discussione sulla pubblicazione incrociata, faremo riferimento a soli asset in forma digitale. Parleremo nel prossimo capitolo dei "formati ricchi", ai quali ci si aggancerà dalla struttura "pura" di XML. Infatti, non tutti i file digitali, come si è visto, hanno lo stesso grado di lavorabilità. Con XML, sarà sempre conveniente riferirsi prima a quelli che contengono il massimo di informazioni possibile ed il più alto grado di lavorabilità, poi, eventualmente, integrare versioni successive via via più "povere" o più vincolate ad uno specifico media.

Dal logo dei "Pomodori Pelati Calabrì", punteremo alla versione vettoriale o ad alta risoluzione, piuttosto che a quella presente in un'iconcina di un sito di commercio elettronico, ben sapendo che dalla prima si può sempre passare alla seconda, ma non è quasi mai possibile il contrario.

La struttura e lo schema ci hanno permesso di trovare i Pomodori Pelati Calabrì. Possiamo usarli per la pizza, le bistecche alla pizzaiola o il ragù. Allo stesso modo, informazioni pure e ben schematizzate possono essere usate in media diversi con ricette e sapori differenti.

Breve iter per exempla

Dopo tanta teoria eccoci finalmente ai nostri tre esempi. Per tutti e tre cercheremo di seguire i tre passaggi fondamentali: regole, *schema* e albero.

Primo esempio: InfoPersona

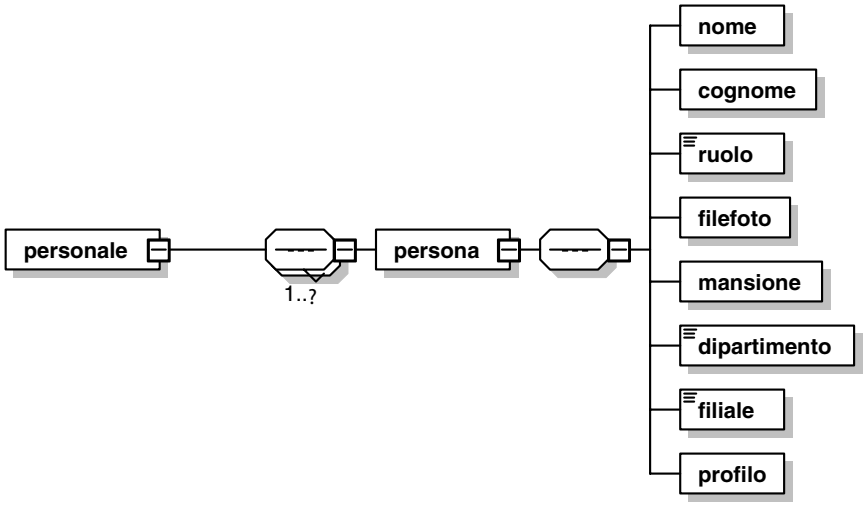
Regole:

- l'archivio è composto da persone;
- per ogni persona dobbiamo necessariamente indicare nome, cognome, ruolo, filefoto, mansione, dipartimento, filiale, profilo;
- nome, cognome, mansione sono semplici valori di testo;
- ruolo è un valore a scelta tra 5 possibili;
- dipartimento è un valore a scelta tra 6 possibili;
- filiale è un valore a scelta tra 4 possibili;
- filefoto è un valore di testo che si riferisce ad un file immagine.

Schema possibile ottenuto con XMLSpy:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 4 U (http://www.xmlspy.com) by Marco Galiazzo (Marco Galiazzo) -->
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="personale">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="persona">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nome"/>
              <xs:element name="cognome"/>
              <xs:element name="ruolo">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="manager"/>
                    <xs:enumeration value="supervisor"/>
                    <xs:enumeration value="specialist"/>
                    <xs:enumeration value="quadro"/>
                    <xs:enumeration value="operatore"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="filefoto"/>
              <xs:element name="mansione"/>
              <xs:element name="dipartimento">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="marketing"/>
                    <xs:enumeration value="ricerca e sviluppo"/>
                    <xs:enumeration value="logistica"/>
                    <xs:enumeration value="produzione"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="filiale">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="Venezia"/>
                    <xs:enumeration value="Padova"/>
                    <xs:enumeration value="Roma"/>
                    <xs:enumeration value="New York"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="profilo"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Unica notazione, la presenza delle restrizioni. Per rispondere all'esigenza di poter contenere in alcuni campi solo determinati valori (ruolo e filiale), è necessario definire un "tipo semplice" che non è altro che un elemento per il quale sono ammessi un ristretto numero di possibili valori. La semplice sintassi prevede la definizione del nome del tipo (`<xsd:simpleType name="filiale">`), la dichiarazione della tipologia di elementi accettabili (tutti testi: `<xsd:restriction base="xsd:string">`), e la lista dei possibili valori ammessi (`<xsd:enumeration value="manager"/>`, e così via). La rappresentazione ad albero permette di comprendere a colpo d'occhio la struttura, senza essere disturbati dai tecnicismi della formalizzazione scritta, ed è un utile strumento per valutare eventuali mancanze o ridondanze.



Un albero, completo di elementi foglia e nodi struttura. L'insieme "personale" è composto da un numero positivo di elemnti "persona". Le tre linee negli elementi foglia "ruolo", "dipartimento" e "filiale" indicano che ci sono delle restrizioni nei valori ammessi.

Secondo esempio: InfoHighTech

Regole (la presentazione completa si ritrova nel primo capitolo alle pagine 41-45)

- procedure di utilizzo da esprimere in 5 lingue (gli elementi multilingua dovranno avere uno specifico tag lingua);
- raggruppate per modello;
- del modello dobbiamo indicare nome, codice, foto generale, descrizione;
- per ogni procedura dobbiamo indicare livello operazione, nome operazione;
- i livelli di operazione possibili sono base, avanzato, supporto;
- ogni procedura può contenere svariate fasi;
- per ogni fase dobbiamo indicare nome, descrizione, immagine, schema e annotazioni;
- immagine e schema sono eventuali;
- per ogni schema dobbiamo indicare file schema e descrizione.

Schema possibile (l'estensione non deve preoccupare: le regole alla base sono sempre le stesse):

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 4 U (http://www.xmlspy.com) by Marco Galiazzo (Marco Galiazzo) -->
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="linea_prodotti">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="modello">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nome"/>
              <xs:element name="codice"/>
              <xs:element name="foto_generale"/>
              <xs:element name="descrizioni_generali">
                <xs:complexType>
                  <xs:sequence minOccurs="0" maxOccurs="unbounded">
                    <xs:element name="descrizione_generale">
                      <xs:complexType>
```

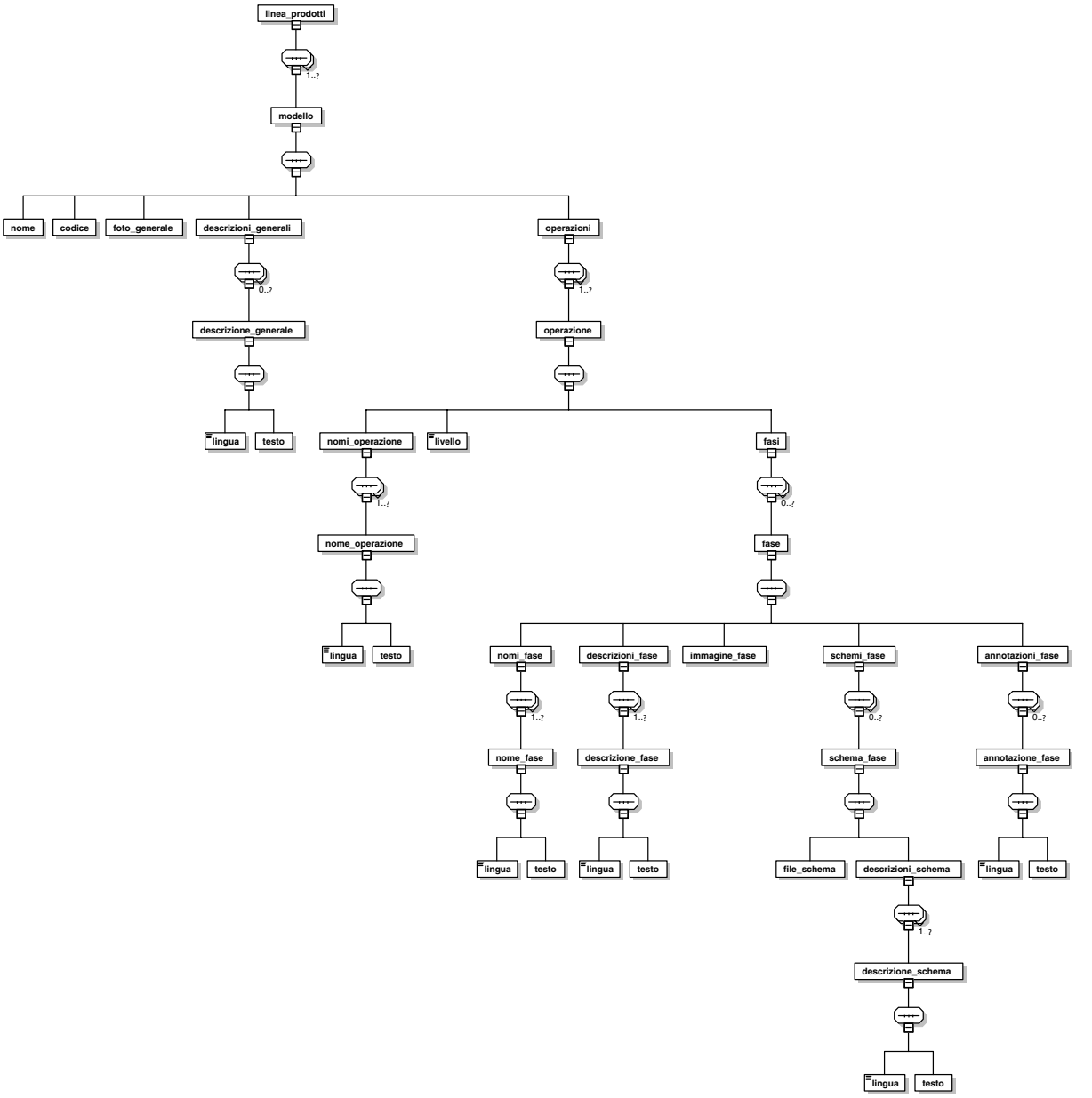
```
          <xs:sequence>
            <xs:element name="lingua">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="italiano"/>
                  <xs:enumeration value="inglese"/>
                  <xs:enumeration value="spagnolo"/>
                  <xs:enumeration value="francese"/>
                  <xs:enumeration value="tedesco"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
            <xs:element name="testo"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="operazioni">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="operazione">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="nomi_operazione">
              <xs:complexType>
                <xs:sequence maxOccurs="unbounded">
                  <xs:element name="nome_operazione">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="lingua">
                          <xs:simpleType>
                            <xs:restriction base="xs:string">
                              <xs:enumeration value="italiano"/>
                              <xs:enumeration value="inglese"/>
                              <xs:enumeration value="spagnolo"/>
                              <xs:enumeration value="francese"/>
                              <xs:enumeration value="tedesco"/>
                            </xs:restriction>
                          </xs:simpleType>
                        </xs:element>
                        <xs:element name="testo"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="livello">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="base"/>
                  <xs:enumeration value="avanzato"/>
                  <xs:enumeration value="supporto"/>
```

```
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="fasi">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="fase">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="nomi_fase">
              <xs:complexType>
                <xs:sequence maxOccurs="unbounded">
                  <xs:element name="nome_fase">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="lingua">
                          <xs:simpleType>
                            <xs:restriction base="xs:string">
                              <xs:enumeration value="italiano"/>
                              <xs:enumeration value="inglese"/>
                              <xs:enumeration value="spagnolo"/>
                              <xs:enumeration value="francese"/>
                              <xs:enumeration value="tedesco"/>
                            </xs:restriction>
                          </xs:simpleType>
                        </xs:element>
                        <xs:element name="testo"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="descrizioni_fase">
              <xs:complexType>
                <xs:sequence maxOccurs="unbounded">
                  <xs:element name="descrizione_fase">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="lingua">
                          <xs:simpleType>
                            <xs:restriction base="xs:string">
                              <xs:enumeration value="italiano"/>
                              <xs:enumeration value="inglese"/>
                              <xs:enumeration value="spagnolo"/>
                              <xs:enumeration value="francese"/>
                              <xs:enumeration value="tedesco"/>
                            </xs:restriction>
                          </xs:simpleType>
                        </xs:element>
                        <xs:element name="testo"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
```

```
  </xs:complexType>
</xs:element>
<xs:element name="immagine_fase"/>
<xs:element name="schemi_fase">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="schema_fase">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="file_schema"/>
            <xs:element name="descrizioni_schema">
              <xs:complexType>
                <xs:sequence maxOccurs="unbounded">
                  <xs:element name="descrizione_schema">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="lingua">
                          <xs:simpleType>
                            <xs:restriction base="xs:string">
                              <xs:enumeration value="italiano"/>
                              <xs:enumeration value="inglese"/>
                              <xs:enumeration value="spagnolo"/>
                              <xs:enumeration value="francese"/>
                              <xs:enumeration value="tedesco"/>
                            </xs:restriction>
                          </xs:simpleType>
                        </xs:element>
                        <xs:element name="testo"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="annotazioni_fase">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="annotazione_fase">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="lingua">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="italiano"/>
                  <xs:enumeration value="inglese"/>
                  <xs:enumeration value="spagnolo"/>
                  <xs:enumeration value="francese"/>
                  <xs:enumeration value="tedesco"/>
                </xs:restriction>
```

```
                                    </xs:simpleType>
                                  </xs:element>
                                  <xs:element name="testo"/>
                                </xs:sequence>
                              </xs:complexType>
                            </xs:element>
                          </xs:sequence>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Anche in questo caso più complesso l'albero favorisce la visuale d'insieme

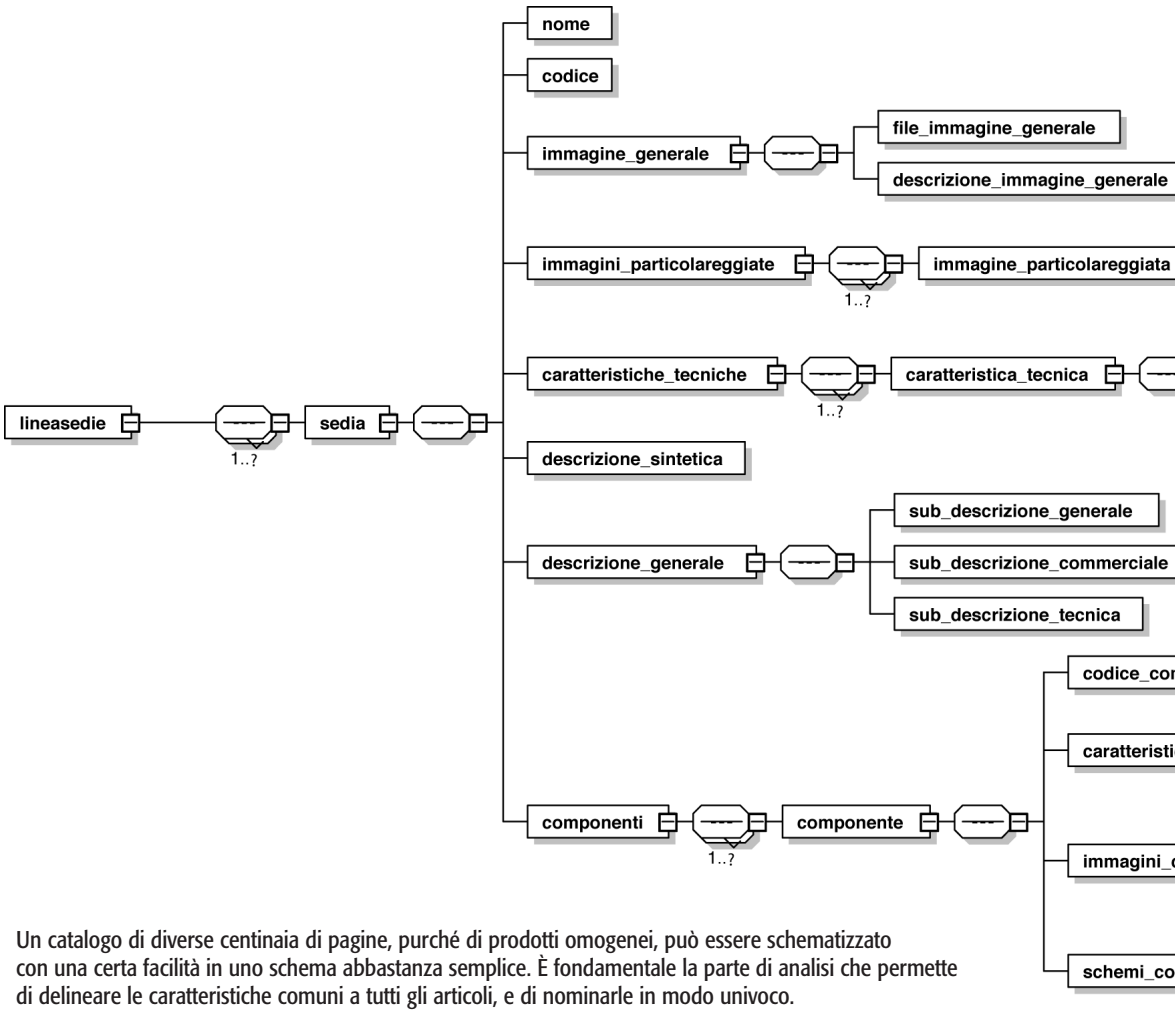


Un esempio mediamente complesso, la possibilità di visualizzare le varie strade possibili nella caratterizzazione del contenuto è senz'altro uno strumento importante. La mente umana, che spesso lavora per associazione di immagini, trova in vari casi congeniale una tale rappresentazione.

Terzo esempio: InfoSedie

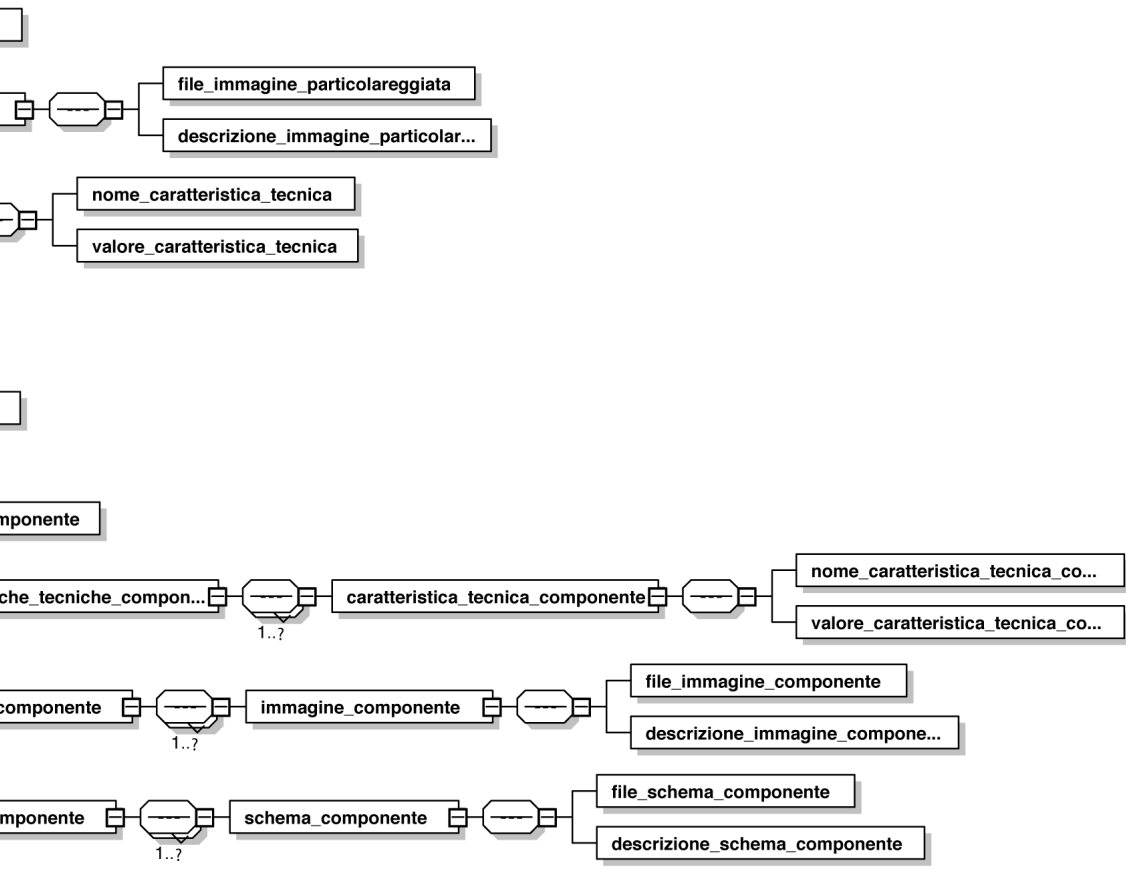
Regole (la presentazione completa si ritrova nel primo capitolo alle pagine 45-47)

- ogni sedia ha un nome, un codice, un'immagine generale, alcune immagini particolareggiate, alcune caratteristiche tecniche, una descrizione sintetica, una descrizione generale, un insieme di componenti;



Un catalogo di diverse centinaia di pagine, purché di prodotti omogenei, può essere schematizzato con una certa facilità in uno schema abbastanza semplice. È fondamentale la parte di analisi che permette di delineare le caratteristiche comuni a tutti gli articoli, e di nominarle in modo univoco.

- per ogni immagine (generale e particolareggiata) c'è un file e una descrizione;
- per ogni caratteristica tecnica c'è un nome e un valore (ad esempio "peso", 25 Kg);
- la descrizione generale è composta da una sub-descrizione generale, da una sub-descrizione commerciale, da una sub-descrizione tecnica;
- ogni componente è composto da un codice, da alcune caratteristiche tecniche, da alcune immagini e alcuni schemi;
- ogni caratteristica tecnica del componente ha un nome e un valore;



- ogni immagine del componente ha un nome file e una descrizione;
- ogni schema del componente ha un nome file e una descrizione.

Anche in questo caso proponiamo lo *schema*, che, se pur esteso, merita di essere analizzato con cura. In un flusso cross-media reale la complessità sarà spesso uguale o superiore a questa.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 4 U (http://www.xmlspy.com) by Marco Galiazzo (Marco Galiazzo) -->
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="lineasedie">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="sedia">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nome"/>
              <xs:element name="codice"/>
              <xs:element name="immagine_generale">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="file_immagine_generale"/>
                    <xs:element name="descrizione_immagine_generale"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="immagini_particolareggiate">
                <xs:complexType>
                  <xs:sequence maxOccurs="unbounded">
                    <xs:element name="immagine_particolareggiata">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="file_immagine_particolareggiata"/>
                          <xs:element name="descrizione_immagine_particolareggiata"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="caratteristiche_tecniche">
                <xs:complexType>
                  <xs:sequence maxOccurs="unbounded">
                    <xs:element name="caratteristica_tecnica">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="nome_caratteristica_tecnica"/>
                          <xs:element name="valore_caratteristica_tecnica"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
```

```
</xs:element>
<xs:element name="descrizione_sintetica"/>
<xs:element name="descrizione_generale">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="sub_descrizione_generale"/>
      <xs:element name="sub_descrizione_commerciale"/>
      <xs:element name="sub_descrizione_tecnica"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="componenti">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="componente">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="codice_componente"/>
            <xs:element name="caratteristiche_tecniche_componente">
              <xs:complexType>
                <xs:sequence maxOccurs="unbounded">
                  <xs:element name="caratteristica_tecnica_componente">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="nome_caratteristica_tecnica_componente"/>
                        <xs:element name="valore_caratteristica_tecnica_componente"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="immagini_componente">
              <xs:complexType>
                <xs:sequence maxOccurs="unbounded">
                  <xs:element name="immagine_componente">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="file_immagine_componente"/>
                        <xs:element name="descrizione_immagine_componente"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="schemi_componente">
              <xs:complexType>
                <xs:sequence maxOccurs="unbounded">
                  <xs:element name="schema_componente">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="file_schema_componente"/>
                        <xs:element name="descrizione_schema_componente"/>
                      </xs:sequence>
                    </xs:complexType>
```

```
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Si potrebbe pensare che la complessità è troppo elevata e difficilmente comprensibile da un operatore medio. Ma basterà un minimo di pratica e l'utilizzo di uno strumento visuale adatto (come XMLSpy), per far svanire ogni preoccupazione. Ricordiamo che si tratta solo di un modo formale per descrivere ciò che abbiamo imposto nelle regole enunciate subito prima. La visualizzazione ad albero ne facilita senz'altro la comprensione.

Nel prossimo capitolo vedremo come sia facile dare ordine attraverso XML ad una massa di dati considerevoli, alla quale siamo abituati ad accedere molto spesso in modo "sparpagliato", o, semplicemente legato alla forma. L'utilizzo dello *schema* ci guiderà nel trovare la giusta posizione per ogni informazione, sia in fase di inserimento, che poi, in fase di utilizzo, come lo schema mentale visto all'inizio ci ha aiutato a trovare i Pomodori Pelati Calabrì e ad utilizzarli in tante prelibate ricette diverse.